PAFI : Outil d'analyse de circuit pour l'accélération de l'injection de fautes en simulation

Olivier Faurax[1,2] Assia Tria[1] Laurent Freund[1] Frédéric Bancel[3]
Traian Muntean[2]

[1]Centre Microélectronique de Provence Georges Charpak, Laboratoire SESAM*
Avenue des Anémones - Quartier Saint-Pierre, 13120 GARDANNE
[2]Université de la Méditerranée, Groupe "Systèmes Informatiques Communicants", 13288 MARSEILLE
[3]STMicroelectronics / Division Smartcard, Zone Industrielle de Rousset, 13106 ROUSSET Cedex
E-mail: faurax@emse.fr

Résumé

Une attaque en faute sur un circuit consiste à perturber physiquement une ou plusieurs parties de celui-ci en vue de le corrompre et d'en exploiter des résultats erronés. Le principe est d'exécuter des calculs interdépendants à une faute près, permettant d'extraire par cryptanalyse des données protégées. Pour se protéger de ce type d'attaque, la robustesse des circuits doit être estimée pendant la conception. Dans cet article, nous présentons PAFI (Prototype of Another Fault Injector), un outil d'analyse de la sécurité des circuits par injection de fautes en simulation.

1. Introduction

À haute altitude, des observations ont mis en évidence des modifications d'état dans des mémoires SRAM provoquées par des neutrons. Des fautes de ce type peuvent être reproduites par radiation [17].

Injecter des fautes dans un circuit peut aider à déjouer ses protections sécuritaires, comme le montre les attaques par DFA (*Differential Fault Analysis*, analyse différentielle de faute). Ce type d'attaque existe pour des cryptosystèmes tels que RSA [4], DES [3] et AES [12][7][18], avec des hypothèses d'injection plus ou moins réalisables en pratique, notamment en ce qui concerne le lieu, le moment et le type des injections de faute.

Pour se prémunir de ces attaques, il est nécessaire de tester le fonctionnement du circuit en présence de fautes. Cela peut être fait en reproduisant physiquement l'environnement produisant les fautes dans le circuit [14][1][16], en utilisant des fonctionnalités internes de test [10][2] ou en modifiant sa description [11]. Nous proposons une méthode d'injection de fautes basée sur la simulation. Cette approche peut être utilisée avant la réalisation physique du circuit, sans modifier sa description, tout en générant des fautes reproductibles.

Le principe général consiste à simuler une description du circuit conjointement au modèle de faute choisi (inversion, collage, etc.). En pratique, le simulateur déroule le fonctionnement du circuit jusqu'au moment d'injection, modifie l'état du circuit et continue le déroulement. Le déroulement après la faute, ainsi que le résultat final, sont analysés pour déterminer le comportement du circuit.

Les circuits deviennent de plus en plus complexes, avec un nombre de portes logiques de plus en plus important, ce qui rend la simulation de toutes les fautes possibles très coûteuse en temps de simulation. En effet, pour des fautes simples, le nombre de simulation à effectuer est linéaire en moments d'injection et en lieux d'injection. Pour des fautes multiples, l'évolution est exponentielle suivant la multiplicité des fautes.

Notre travail a pour but de réduire *a priori* le nombre de fautes à injecter en analysant la structure du circuit et en utilisant des propriétés du modèle de faute. Cette méthode a donné lieu à PAFI (*Prototype of Another Fault Injector*), un outil d'injection de fautes combinant l'analyse du circuit, l'automatisation de la campagne d'injections de faute et l'analyse des résultats.

Cet article est organisé de la façon suivante : une présentation des travaux connexes sur l'injection de fautes est proposée dans la section 2. Nous exposerons ensuite les modèles de circuit et de faute que nous étudions dans la section 3. Ensuite, une présentation de l'outil d'injection de fautes PAFI ainsi qu'une analyse de la sécurité d'une implémentation d'AES aura lieu dans la section 4. Pour finir, les perspectives de nos travaux seront exposés dans la section 5.

2. Travaux connexes

Plusieurs projets ont pour but de déterminer le comportement de circuits lors d'injections de fautes en simulation.

MEFISTO [15] utilisait des saboteurs (modification d'un signal) et des mutants (remplacement de com-

*Le laboratoire SESAM est une équipe mixte CEA-LETI / Ecole des Mines de Saint-Étienne basée sur le site Georges Charpak (Gardanne).

posant) pour injecter des fautes dans des descriptions hiérarchiques en VHDL. Il était basé sur un simulateur qui n'existe plus (VHDL Optimum). MEFISTO-L [5] est une amélioration pour tester les fonctionnalités de tolérance aux fautes et rendre MEFISTO indépendant du simulateur en utilisant exclusivement des modifications VHDL. Cette approche induit une recompilation du circuit pour prendre en compte les modifications apportées à la description d'origine.

VERIFY [19] est un outil qui étend la définition VHDL des portes pour refléter le modèle de faute utilisé. Cela implique d'utiliser une compilateur modifié pour prendre en compte ces définitions étendues des composants. De plus, le modèle de faute est basé sur les portes et l'outil ne permet pas les modèles plus généraux comme des injections multiples sur plusieurs portes au même moment.

DEPEND [13] utilise des objets C++ pour modéliser le circuit. Ces objets possèdent un comportement en faute intégré qui peut être changé. Le temps de simulation est réduit en utilisant la structure hiérarchique du circuit ainsi qu'en ignorant les fautes sans répercutions. Cependant, le modèle de faute est défini dans la bibliothèque d'objets C++ et ne peut pas être affiné pour chaque instance de composant.

SINJECT [20] est un outil d'injection qui utilise le principe des mutants et des saboteurs de MEFISTO, en ajoutant le support pour les descriptions de circuit mixte (VHDL/Verilog) non-synthetisable. Pour chaque injection possible, il est nécessaire d'ajouter un signal supplémentaire aux portes correspondantes.

Notre approche consiste à utiliser une description du circuit non-modifiée sous la forme d'une interconnexion de composants de base (*netlist*) pour pouvoir analyser et injecter des fautes de façon assez réaliste. La réduction du temps de simulation se fait en aidant l'utilisateur à sélectionner les fautes les plus probables au regard du modèle de faute choisi.

3. Modèles

3.1. Modèle de circuit

L'état d'un circuit synchrone au cycle d'horloge t dépend de ses entrées et de son état interne à $t-1$.

Nous étudions les circuits correspondants au modèle de Moore : après initialisation, le circuit n'évolue qu'en fonction de son état courant.

Le déroulement général d'un circuit dans notre modèle commence par l'initialisation de l'état interne du circuit par ses entrées (un ou plusieurs cycles).

Le circuit est modélisé comme un ensemble de bascules (points mémoire de 1 bit) interconnectées par des cônes logiques. Nous considérons que l'état interne du circuit se résume à l'état de chacune de ses bascules internes.

Dans notre modèle, les entrées ne sont utilisées que lors de l'initialisation du calcul. Ensuite, comme le jeu complet des données d'entrées a été fourni, le déroulement du circuit continue en utilisant exclusivement son état interne : c'est l'étape de calcul.

Une fois le calcul terminé, l'état interne est réinitialisé.

Ce modèle est réaliste pour les cryptosystèmes par blocs qui n'utilisent pas de chaînage[1]. De plus, il permet la parallélisation des simulations utilisant des jeux de données différents ou, dans notre cas, soumis à des fautes différentes.

3.2. Modèle de faute

Nous nous intéressons aux fautes transitoires qui ont un effet sur le résultat du circuit.

Injecter une faute lors de l'initialisation donne un résultat prévisible car c'est équivalent à changer les données d'entrées, ce qui présente peu d'intérêt dans le cadre d'une exploitation de faille de sécurité. De plus, la logique de contrôle n'est pas encore utilisée, elle n'est donc pas vulnérable. Par exemple, injecter dans un compteur de rondes ne produit pas d'erreur puisqu'il sera réinitialisé au début du calcul, c'est-à-dire à la fin de l'initialisation.

Injecter une faute lors du calcul peut conduire le circuit à un état interne imprévu et donc à un résultat inattendu. Cela présente un risque de sécurité puisque c'est le principe des attaques par DFA (*Differential Fault Analysis*). Dans ce contexte, les fautes considérées affecteront le circuit pendant la phase de calcul.

Actuellement, nous ne prenons en compte que les fautes simples (1 inversion par simulation). Les fautes multiples seront étudiées par la suite et nécessitent de redéfinir le type des fautes à injecter.

Dans notre modèle, une faute modifie la valeur d'un bit de façon transitoire (1 cycle d'horloge). 3 types de modifications de bit sont possibles : collage à 0, collage à 1 ou inversion. Le seul type de faute qui modifie à coup sûr la valeur est l'inversion. De plus, si un collage produit une modification, il est équivalent à une inversion puisque ce collage ne se produit que de manière transitoire. C'est le modèle de faute correspondant à l'effet physique SEU.

L'état du circuit pendant le calcul est mémorisé dans l'ensemble de ses bascules. Chaque bascule n'est pas mise à jour instantanément : il y a un temps de propagation à travers les cônes logiques vers les bascules.

Nous nous intéressons aux fautes provoquées par un ajout de délai. Ce sont donc les bascules contenant le résultat des cônes dont les temps de traversée sont les plus longs qui sont le plus susceptibles d'être perturbées.

Ce type d'information peut être obtenu par analyse statique de la description du circuit en utilisant des informations de temps de traversée pour les composants de la

[1] Cependant, le chaînage peut être considéré comme faisant partie de l'initialisation

technologie utilisée lors de l'élaboration de la *netlist*. Actuellement, pour chaque bascule, nous prenons en compte le temps de traversée maximum du cône logique correspondant.

4. PAFI : Prototype of Another Fault Injector

PAFI (figure 1) est un outil d'analyse de la fiabilité et de la sécurité des circuits par injection de fautes en simulation et sans modification du circuit.

4.1. Description

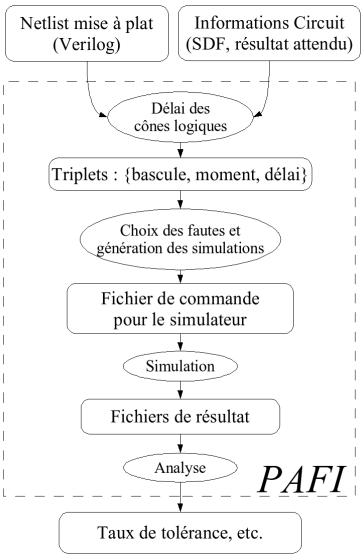

FIG. **1. PAFI**

À partir de la *netlist*, PAFI effectue une analyse statique pour extraire des informations (par exemple, les points d'injection possibles) qui l'aideront dans son choix des endroits d'injection.

Suite à cette analyse, l'utilisateur peut paramétrer sa campagne d'injection en fonction du nombre de simulations qu'il souhaite réaliser.

PAFI génère alors un ensemble de scripts de commandes pour que le simulateur réalise la campagne d'injection. Le simulateur utilisé actuellement est NCSim, mais PAFI est modulaire (figure 1) pour permettre le support d'autre simulateurs.

Pour la simulation, le circuit à tester est instancié dans un environnement qui fournit les entrées au circuit et écrit des fichiers contenant les données de sortie ainsi que les infomations nécessaires à l'interprétation des résultats, comme une comparaison avec les valeurs attendues. Par exemple, ce fichier peut contenir la comparaison entre le résultat fauté et le résultat théorique.

PAFI contient également un analyseur de résultats qui permet de mettre en évidence les caractéristiques du comportement du circuit lors des injections de fautes. Les figures 2 et 3 sont des exemples de graphiques issus de PAFI.

PAFI est conçu pour être un outil générique car il ne fait pas d'hypothèses sur la nature du circuit (cryptosystème, CPU, etc.). Il est donc possible de l'utiliser dans un cadre plus large que le modèle vu dans la partie 3.1, en redéfinissant également le modèle de faute de la partie 3.2.

De plus, la fonction qui extrait les informations de la description du circuit est adaptable aux modèles de fautes voulus. Il est ainsi possible de privilégier les injections selon des critères de sécurité, en sélectionnant les endroits d'injection suivant des propriétés algorithmiques connues (comme nous l'avons proposé dans [8][9]), ou des critères physiques, en se basant sur les fautes observées physiquement sur les circuits.

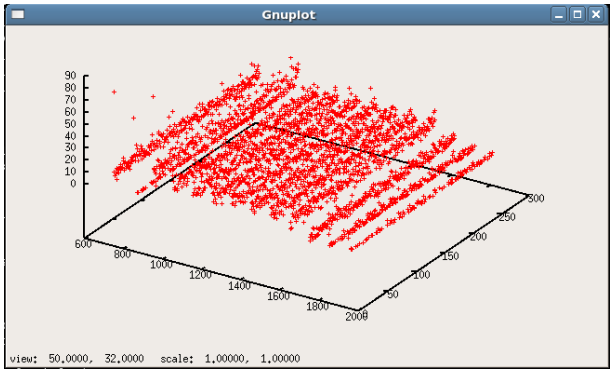

FIG. **2. PAFI : visualisation des sorties**

4.2. Cas d'étude : une implémentation d'AES

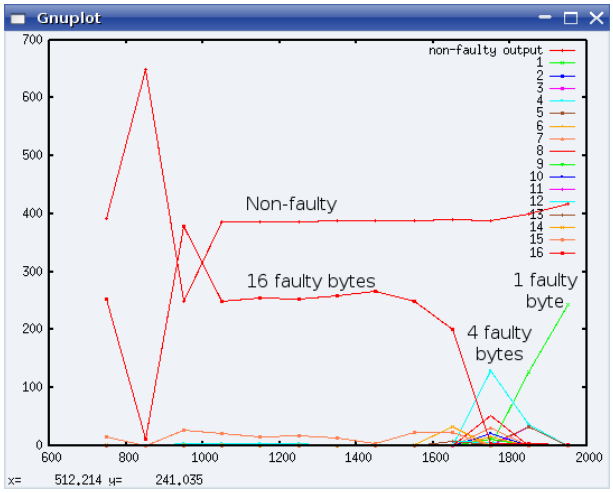

FIG. **3. Octets faux en sortie de l'AES**

Nous avons utilisé PAFI pour tester la sécurité d'une implémentation de l'algorithme de cryptographie AES[6], dont le résultat est sur 16 octets.

La majorité des fautes injectées (60,2%) n'ont pas d'effet sur le résultat du circuit et 27,6% des fautes

génèrent un résultat complètement différent du résultat attendu (16 octets faux). Certaines injections réalisées à la fin du calcul génèrent des fautes sur 4 octets, ce qui vérifie les propriétés de diffusion de l'AES. Ces résultats sont illustrés sur la figure 3.

De plus, nous avons pu valider le caractère aléatoire des sorties après injection, visible sur la figure 2.

Une campagne exhaustive nécessite 665 injections par unité de temps. L'utilisation d'un modèle de faute en délai réduit à 195 le nombre d'injection à effectuer, soit un gain d'un facteur 3,4.

5. Conclusion & Travaux futurs

Dans cet article, nous avons présenté PAFI, un outil d'analyse et d'injection de fautes en simulation dans les circuits. PAFI ne nécessite pas de modifications du circuit à tester.

Actuellement, PAFI analyse la structure du circuit en effectuant une analyse statique. Dans la suite de notre travail, nous allons ajouter une analyse dynamique en prenant en compte l'état interne du circuit lors d'une simulation sans faute.

La combinaison des analyses statique et dynamique nous permettra de renforcer les critères de réduction du nombre de fautes à simuler et donc de prendre en compte des modèles de fautes multiples.

Références

[1] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell. Fault Injection for Dependability Validation : A Methodology and Some Applications. *IEEE Trans. Softw. Eng.*, 16(2) :166–182, 1990.

[2] A. Benso, P. Prinetto, M. Rebaudengo, and M. S. Reorda. EXFI : a low-cost fault injection system for embedded microprocessor-based boards. *ACM Transactions on Design Automation of Electronic Systems.*, 3(4) :626–634, 1998.

[3] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In B. S. K. Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[4] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. *Lecture Notes in Computer Science*, 1233 :37–51, 1997.

[5] J. Boué, P. Pétillon, and Y. Crouzet. MEFISTO-L : A VHDL-Based Fault Injection Tool for the Experimental Assessment of Fault Tolerance. In *FTCS '98 : Proceedings of the The Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing*, page 168, Washington, DC, USA, 1998. IEEE Computer Society.

[6] J. Daemen and V. Rijmen. AES Proposal : Rijndael, 1998.

[7] P. Dusart, G. Letourneux, and O. Vivolo. Differential Fault Analysis on A.E.S. Cryptology ePrint Archive, Report 2003/010, 2003. http ://eprint.iacr.org/.

[8] O. Faurax, L. Freund, F. Bancel, and T. Muntean. Une méthode générique pour l'injection de fautes dans les circuits. In *Actes des 9èmes Journées Nationales du Réseau Doctoral en Microélectronique*, May 2006. ISBN : 978-2-9527172-0-5.

[9] O. Faurax, L. Freund, A. Tria, T. Muntean, and F. Bancel. A generic method for fault injection in circuits. In *IWSOC 2006 : Proceedings of the 6th IEEE International Workshop on System-on-Chip for Real-Time Applications*, pages 211–214. IEEE Circuits and Systems Society, 2006.

[10] P. Folkesson, S. Svensson, and J. Karlsson. Evaluation of the Thor Microprocessor Using Scan-chain-Based and Simulation-Based Fault-Injection, 1997.

[11] J. Francq, P. Manet, and J.-B. Rigaud. Material Emulation Of Faults On Cryptoprocessors. In *Proceedings of Sophia Antipolis forum of MicroElectronics (SAME) 2006*, 2006.

[12] C. Giraud. DFA on AES. In H. Dobbertin, V. Rijmen, and A. Sowa, editors, *Advanced Encryption Standard - AES, 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers*, volume 3373 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2004.

[13] K. K. Goswami, R. K. Iyer, and L. Young. DEPEND : A Simulation-Based Environment for System Level Dependability Analysis. *IEEE Trans. Comput.*, 46(1) :60–74, 1997.

[14] U. Gunneflo, J. Karlsson, and J. Torin. Evaluation of error detection schemes using fault injection by heavy-ion radiation. In *Proceedings of the 19th International Symposium on Fault Tolerant Computing, (FTCS-19), IEEE, Austin, Texas, USA*, pages 340–347, 1989.

[15] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson. Fault Injection into VHDL Models : The MEFISTO Tool. In *Proceedings of the 24th International Symposium on Fault Tolerant Computing, (FTCS-24), IEEE, Austin, Texas, USA*, pages 66–75, 1994.

[16] H. Madeira, M. Z. Rela, F. Moreira, and J. G. Silva. RIFLE : A General Purpose Pin-level Fault Injector. In *European Dependable Computing Conference*, pages 199–216, 1994.

[17] J. Olsen, P. Becher, P. Fynbo, P. Raaby, and J. Schultz. Neutron-Induced Single Event Upsets in Static RAMS Observed at 10 KM Flight Altitude. *IEEE TRANSACTIONS ON NUCLEAR SCIENCE*, 40(2), 1993.

[18] G. Piret and J.-J. Quisquater. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. In C. D. Walter, Çetin Kaya Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems − CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2003.

[19] V. Sieh, O. Tschäche, and F. Balbach. VERIFY : Evaluation of Reliability Using VHDL-Models with Embedded Fault Descriptions. In *FTCS '97 : Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS '97)*, pages 32–36, Washington, DC, USA, 1997. IEEE Computer Society.

[20] H. R. Zarandi, S. G. Miremadi, and A. Ejlali. Dependability Analysis Using a Fault Injection Tool Based on Synthesizability of HDL Models. In *DFT '03 : Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 485–492. IEEE Computer Society, 2003.