Une méthode générique pour l'injection de fautes dans les circuits

Olivier Faurax[1,2] 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

Les microcircuits intégrés dans les cartes à puces dédiées à des applications de sécurité sont la cible d'attaques de plus en plus sophistiquées telles que les attaques en faute qui combinent perturbation physique et cryptanalyse. L'utilisation de la simulation pour la validation de ces circuits face à de telles attaques est limitée par le temps nécessaire pour effectuer les injections des fautes choisies. Il faut alors déterminer quels sont les points d'injection à privilégier. Dans la majorité des cas, ce choix est fait par l'utilisateur sur la base de sa connaissance des fonctionnalités du circuit. Dans cet article, nous proposons une méthode générique et semi-automatisable de réduction du nombre d'injections en prenant en compte le type des données manipulées par les registres, bascule par bascule.

1. Introduction

Une attaque en fautes 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 inter-dépendants à une faute près permettant d'extraire des données protégées par cryptanalyse. Les premiers travaux traitant des attaques en fautes sont les DFA (*Differential Fault Analysis*, analyse différentielle de faute) qui ont mis en évidence des attaques sur RSA [2] et sur DES [1]. Depuis, plusieurs attaques ont été proposées sur AES [5][3][9].

Pour concevoir un circuit résistant aux fautes, il est nécessaire de pouvoir anticiper les réactions du circuit en leur présence. Nous préconisons l'injection de fautes en simulation qui a l'avantage de se situer en amont de la fabrication, de générer des fautes reproductibles et d'être peu onéreuse.

Le principe est de simuler le comportement du circuit en présence du modèle de faute choisi (bit-flip, collage, etc.). En pratique, on utilise un simulateur auquel on fournit le modèle du circuit. On lance ensuite la simulation jusqu'à l'étape d'injection de faute que l'on choisit à un instant précis. On modifie alors l'état du circuit et la simulation continue. Pour finir, on analyse le comportement du circuit.

Cependant, la complexité des circuits actuels (nombre de portes, niveaux de métal, etc.) ne permet pas d'envisager la simulation de toutes les fautes possibles. Le nombre de points d'injection peut être assez élevé et il est d'autant plus important si on prend en compte le paramètre temporel. Cette complexité augmente encore si on considère la multiplicité des fautes. Pour des fautes simples, le temps de simulation est linéaire en nombre de points et de moments d'injection.

Dans notre travail, nous nous intéressons aux critères de choix des points d'injections par rapport à leur fonction dans le circuit. L'objectif est de pouvoir guider, *a priori*, la campagne d'injections vers les fautes qui ont la plus forte probabilité d'affecter le comportement du système.

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 justifierons ensuite notre système de pondération fonctionnelle des bascules critiques du circuit dans la section 3. Ensuite, une discussion de l'importance relative des fautes dans l'environnement de propagation de ces bascules critiques aura lieu dans la section 4. La section 5 présentera notre méthode générique et semi-automatisable. Pour finir, les perspectives de nos travaux seront exposés dans la section 6.

2. Travaux connexes

Les outils et méthodes d'injection de fautes en simulation existent depuis une vingtaine d'année. Nous présentons brièvement et de manière non-exhaustive certaines caractéristiques d'outils existants.

MEFISTO [8] est une des plus importantes contributions au domaine puisqu'il a permis de simuler des fautes multi-niveaux sur une modélisation en VHDL, en utilisant les techniques des saboteurs et des mutants.

VERIFY [10] propose une nouvelle façon d'injecter des fautes en proposant une syntaxe étendue des signaux VHDL, ce qui a l'avantage de ne pas nécessiter de recompilation comme les mutants de MEFISTO. Néanmoins, il faut utiliser un compilateur spécifique pour pouvoir prendre en compte ces extensions.

SINJECT [14] est un outil d'injection en simulation qui reprend le principe des mutants et des saboteurs de MEFISTO, en ajoutant le support des modélisations mixtes VHDL/Verilog non-synthétisables.

FITSEC [4] scinde le circuit entre une partie émulée sur FPGA et une partie simulée, ce qui accélère les injections. Cette combinaison accélère les tests d'un facteur 100 par

rapport à une simulation.

Ces outils permettent à l'utilisateur d'injecter des fautes en simulation sans pour autant l'orienter vers des injections préférentielles lui permettant d'optimiser ses calculs.

D'autres outils orientent l'utilisateur vers un sous-ensemble particulier d'injections possibles. Deux approches sont proposées : sélectionner les injections qui ont une grande chance de perturber le système (pour obtenir une borne inférieure du taux de couverture) ou sélectionner des injections représentatives de l'ensemble des fautes (pour obtenir un taux de couverture proche de celui du circuit).

DEPEND [6] limite le nombre de fautes en se basant sur une analyse de la charge de travail.

Güthoff et Sieh [7] proposent d'analyser l'exécution sans fautes (*golden run*) d'instructions exécutées sur un processeur pour ne simuler que les fautes sur les registres très utilisés en ne sélectionnant que les moments d'injection où la valeur du registre est significative.

La technique de l'extension de faute (*fault expansion*) [11] consiste à regrouper les fautes et à n'en simuler qu'un représentant. Cependant, cette méthode n'est réellement efficace que lorsque les groupes sont de grande taille par rapport au nombre de fautes total [12].

Notre approche se distingue des travaux précédents car elle se base sur le type des données manipulées aux entrées/sorties des bascules pour en déduire les endroits privilégiés d'injection de fautes.

3. Pondération fonctionnelle des bascules

Nous nous intéressons aux fautes transitoires qui provoquent un ou plusieurs changement d'état. L'état général d'un circuit est mémorisé dans l'ensemble de ses bascules. Les fautes prises en compte sont celles qui affectent ces éléments mémorisants.

Pour discerner les bascules critiques, notre approche consiste à associer à chacune d'elles une pondération représentant son impact sur le comportement du circuit. Cette pondération permettra d'orienter l'utilisateur vers le choix des bascules à simuler en priorité.

Notre pondération fait intervenir les facteurs structurels et fonctionnels du circuit.

Les facteurs structurels peuvent être déduits de la netlist du circuit : cône logique précédant la bascule, type de bascule, etc. Ils ne font pas partie des points traités dans cet article.

Les facteurs fonctionnels sont relatifs à l'utilisation de la bascule au regard de l'ensemble du circuit : type de donnée, valeur critique pour le déroulement du circuit, etc. Ces informations doivent être fournies par l'utilisateur puisqu'elles ne peuvent pas être extraites du circuit. Elles permettent ensuite de calculer la composante fonctionnelle de la pondération.

Nous considérons trois types de données critiques : les données secrètes, les données de contrôle et les données de sorties.

Une donnée secrète doit être isolée de l'extérieur du circuit pour ne pas révéler d'information. S'il existe une donnée observable (i.e. qui peut être lue) dont la valeur dépend d'une donnée secrète, alors cette dernière doit également être protégée. En effet, toute perturbation générant un résultat faux pourrait permettre d'extraire la donnée secrète par cryptanalyse.

Pendant le fonctionnement du circuit, les données de contrôle représentent les résultats de tests et déterminent le comportement du circuit. Une perturbation sur les bascules correspondantes peut avoir une incidence sur la sécurité, par exemple en validant une authentification fausse ou en permettant un accès à des ressources protégées.

Les données de sorties sont facilement observables. Une perturbation qui ne modifie qu'une partie du résultat facilite leur cryptanalyse. Dusart, Letourneux et Vivolo [3] le démontre sur AES en ne modifiant qu'un quart du résultat (4 octets sur les 16 du résultat).

La première étape de notre méthode de pondération (que nous développons dans la section 6) consiste à affecter une pondération bascules qui manipulent ces trois types de données.

4. Perturbations liées

Les bascules sont interconnectées les unes aux autres par des cônes logiques. Si une perturbation sur une bascule B (cf. figure 1) change le comportement du circuit, des perturbations sur les bascules rencontrées en suivant les chemins de propagation (A_i et C_i) peuvent aussi induire une modification du fonctionnement du circuit.

Les fautes produites sur ces bascules sont appelées des *perturbations liées* et se décomposent en *perturbations antérieures* et *perturbations postérieures*.

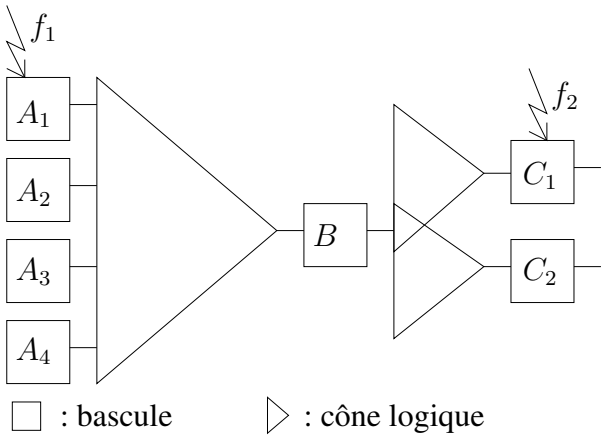

FIG. **1. Perturbations liées (f_1 : antérieure, f_2 : postérieure)**

4.1. Perturbations antérieures

Une faute injectée sur une des bascules d'entrée du cône logique de la bascule B peut avoir une influence sur la valeur de cette dernière. Ces bascules sont notées A_i et ces perturbations sont appelées *perturbations antérieures* car B est dans les chemins de propagation issus des A_i.

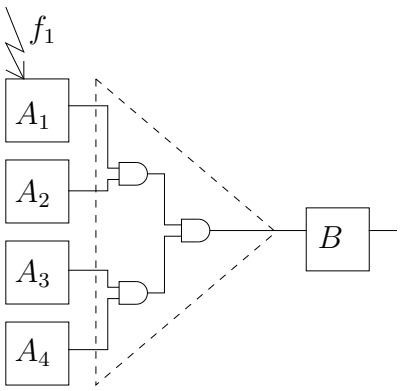

FIG. **2. Exemple de perturbation antérieure**

On observe sur le cas illustré par la figure 2 qu'un changement de la valeur d'une des bascules A_i est équivalent à une faute sur la bascule B.

Si B manipule une donnée secrète, une faute sur l'adressage nécessaire à la récupération de celle-ci correspond à une perturbation antérieure, puisque le comportement est modifié même si la bascule contenant la donnée secrète n'est pas directement touchée par la faute.

Si B manipule une donnée de contrôle, une perturbation antérieure à celle-ci modifiera le fonctionnement du circuit si le résultat du test est altéré. Il est même possible d'obtenir un blocage du circuit, ce qui est un comportement acceptable si aucun résultat erroné ne sort du circuit.

Une faute injectée proche des sorties causera une modification partielle du résultat qui permettra une attaque par analyse différentielle. De ce fait, on considère que les données de sorties issues de B sont des données critiques compte tenu des perturbations antérieures possibles.

Ainsi, la pondération d'une bascule B peut induire un phénomène en cascade sur les poids des bascules rencontrées en remontant les chemins de propagations (A_i).

4.2. Perturbations postérieures

De même, perturber les bascules de sortie des cônes logiques dont B est une des entrées va potentiellement changer l'exécution d'une partie du circuit. Ces bascules seront notées C_i et ces perturbations seront appelées *perturbations postérieures* car les C_i sont dans les chemins de propagation passant par B.

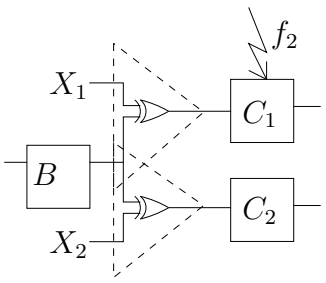

FIG. **3. Exemple de perturbation postérieure**

On observe sur le cas illustré par la figure 3 qu'un changement de la valeur d'une des bascules suivant B à le même impact sur une partie du circuit (celle dépendant de C_1) qu'une faute sur B.

Par exemple, une bascule contenant un résultat intermédiaire dépendant d'une donnée secrète peut révéler des informations si elle est perturbée. Les travaux de Yen et Joye [13] montrent que l'injection d'une *safe error* (i.e. erreur qui ne produit pas un résultat faux) sur un résultat intermédiaire lors d'une exponentiation permet de récupérer des informations secrètes.

Une perturbation postérieure sur des données de contrôle peut dérouter une partie du circuit de son exécution normale. Le circuit peut alors se retrouver dans un état imprévu, voire illégal au regard des spécifications de sécurité requises.

En prenant en compte les perturbations postérieures, une pondération sur une bascule B entraîne une pondération sur les bascules des cônes logiques dépendants de celle-ci (c'est-à-dire les C_i).

4.3. Combinaison des pondérations

Les pondérations vont s'atténuer en descendant ou remontant les chemins de propagations tout en se combinant entre elles, ce qui fera apparaître les points d'injection potentiellement intéressants.

Les attaques sur AES [5][3][9] se basent sur des fautes proches de la sortie qui ont la propriété d'être également proches des clés de tours (secrètes) : la faute est en même temps une perturbation antérieure à la sortie et une perturbation postérieure à des données secrètes.

5. Méthode de détermination des pondérations

Lors de la première étape, l'utilisateur fournit la liste initiale des couples {bascule, pondération} pour les bascules manipulant des données secrètes, de contrôle ou de sorties. Dans l'exemple de la figure 1, cela correspond à la liste des bascules B du circuit et de leurs pondérations respectives.

À partir de ces informations et de la netlist du circuit, la deuxième étape consiste à calculer les pondérations des bascules liées (A_i et C_i). Ces couples {bascule, pondération} sont alors ajoutés à la liste. Cette deuxième étape est réitérée jusqu'à l'obtention d'une liste stable (c'est-à-dire qui n'est pas modifiée entre 2 itérations).

On obtient ainsi comme résultat un liste de couples {bascule, pondération} qui guide l'utilisateur dans le choix des injections de fautes à simuler pour valider le circuit. Le déroulement du calcul est décrit plus formellement par l'algorithme 1.

Cette méthode est générique puisqu'elle ne fait pas d'hypothèse sur le type de circuit considéré. De plus, à l'inverse de la première étape qui nécessite l'intervention de l'utilisateur, la seconde étape peut être rendue entièrement automatique.

6. Conclusion & perspectives

Dans ce travail, nous proposons les fondements d'une méthode originale permettant de valider le comportement d'un circuit face aux attaques en fautes. Cette méthode

Algorithme 1 Algorithme de pondération des bascules

ENTRÉE : $circuit \leftarrow$ description du circuit
ENTRÉE : $nouvelle\_liste \leftarrow$ ensemble des couples $(bascule\_critique, ponderation)$
ENTRÉE : $ancienne\_liste \leftarrow \{\}$
SORTIE : la liste des pondérations finales
  tant que $nouvelle\_liste \neq ancienne\_liste$ **faire**
    $ancienne\_liste \leftarrow nouvelle\_liste$
    pour chaque $bascule$ dans $circuit$ **faire**
      $tmp\_ponderation \leftarrow Calcul\_ponderation(bascule, circuit, ancienne\_liste)$
      si $tmp\_ponderation > 0$ **alors**
        $nouvelle\_liste \leftarrow nouvelle\_liste \cup \{(bascule, tmp\_ponderation)\}$
      fin si
    fin pour
  fin tant que
  return $nouvelle\_liste$

générique et semi-automatisable permet de cibler les points d'injection critiques rendant ainsi le temps de simulation acceptable.

Nous travaillons actuellement sur l'affectation des pondérations initiales des bascules en fonction de leur criticité (valeur des B) et l'influence de la pondération de B sur le calcul de celles des A_i et C_i dans le cas d'un modèle simplifié tel que décrit dans la figure 1. Par la suite, nous travaillerons sur un modèle multiple (interconnexions de modèles simplifiés) dont les pondérations seront déduites d'une combinaison de celles des modèles simplifiés.

La suite du travail consistera à faire des hypothèses sur les valeurs et les formules de calcul, puis à tester expérimentalement ces choix avec l'outil que nous développons à cet effet. Les résultats ainsi obtenus permettront de déterminer les différents facteurs de la pondération fonctionnelle.

Il faudra ensuite combiner la pondération fonctionnelle à la pondération structurelle, qui prend en compte la structure du circuit autour des points d'injection considérés.

Références

[1] 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.

[2] 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.

[3] 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/.

[4] A. R. Ejlali, G. Miremadi, H. R. Zarandi, G. Asadi, and S. B. Sarmadi. A hybrid fault injection approach based on simulation and emulation co-operation. In *DSN-2003 : Proceedings of the 2003 International Conference on Dependable Systems and Networks*, pages 479–488, Washington, DC, USA, 2003. IEEE Computer Society.

[5] 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.

[6] 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.

[7] J. Güthoff and V. Sieh. Combining Software-Implemented and Simulation-Based Fault Injection into a Single Fault Injection Method. In *FTCS '95 : Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing*, pages 196–206, Washington, DC, USA, 1995. IEEE Computer Society.

[8] 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.

[9] 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.

[10] 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.

[11] D. Smith, B. Johnson, I. Profeta, J.A., and D. Bozzolo. A method to determine equivalent fault classes for permanent and transient faults. pages 418–424, Washington, DC, USA, 1995. IEEE Computer Society.

[12] W. Wang, K. S. Trivedi, B. V. Shah, and I. Joseph A. Profeta. The impact of fault expansion on the interval estimate for fault detection coverage. In *FTCS '94 : Proceedings of the 24th International Symposium on Fault-Tolerant Computing (FTCS '94)*, pages 330–337, Austin, TX, USA, 1994. IEEE Computer Society.

[13] S.-M. Yen and M. Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. *IEEE Trans. Comput.*, 49(9) :967–970, 2000.

[14] 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.