TP de sécurité des systèmes embarqués

Hugues de Valon
Paul Luperini
Lucas Mahieu

25 janvier 2017

Table des matières

1 Introduction

De nos jours, le chiffrement des données est essentiel pour assurer la sécurité des transferts de données et préserver la sécurité d'un pays, d'une entreprise ou la vie privée d'un individu.

L'algorithme AES (Advanced Encryption Standard) est un algorithme très utilisé en chiffrement des données car reputé incassable. Néanmoins, nous allons voir, au travers de deux exemples réalisés en travaux pratiques, que cette sécurité dépend grandement de l'implémentation qu'il en est fait dans un circuit électronique.

Dans un premier temps, nous allons réaliser une attaque par fautes sur un circuit AES utilisant une protection par redondance d'informations. Ensuite nous réaliserons une attaque par canaux auxiliaires sur un circuit AES n'ayant aucune protection spéciale.

2 TP 1

2.1 Introduction

2.2 Quelle protections sont mises en place contre les attaques par fautes ?

Bien que l'algorithme de l'AES soit incassable aujourd'hui, son implémentation peut permettre de récupérer des données sensibles par de nombreux moyens. Dans ce TP, on s'intéresse à une potentielle attaque par injection de fautes et a une contre-mesure de cette attaque. Pour mener un attaque par faute dans ce module AES, nous avons tout d'abord analysé le circuit et tenté de comprendre comment le système était protégé.

La sécurité du chiffrement tel qu'il est implémenté pour ce TP réside dans la redondance d'information. En effet, chaque calcul sur les octets du data_unit sont fait 2 fois, cela permet, à l'issue des deux calculs, de comparer que les 2 résultats sont identiques. Dans ce cas, aucune faute ne sera détectée. En revanche, si l'un des deux calculs ne donne pas le même résultat, un flag sera levé, et le calcul sera considéré comme erroné.

De plus, il est possible de doubler le nombre de calculs effectué par round, sans pour autant doubler le temps de calculs. En effet, d'après la figure **??** page **??** montre que durant les 6 premiers cycles 12 calculs sont effectués (sur fronts montant et descendant) et sur les 6 cycles suivants, les mêmes calculs sont à nouveau effectués. Ainsi si l'assaillant change une donnée du circuit, le calcul fait sur les 6 premiers cycles et celui fait sur les 6 derniers seront surement différents et une erreur sera levée.

2.3 Quelles sont les faiblesses de cette sécurité ?

Le problème avec cette façon de sécurisé l'AES est que si l'on change la valeur d'un octet pendant exactement 6 cycles, la valeur du premier calculs sera identique au deuxième calcul. Ainsi, il ne sera plus possible de faire remonter l'erreur. Cela permettrai à l'attaquant de faire changer un octet d'un

2.4 Implémentation de l'attaque par faute

En utilisant la faiblesse expliquée précedement, notre implémentation consiste à introduire une faute de retournement de bits sur le port `in_hi`, sur 8 bits, de la colonne 0. Nous nous basons sur un modèle réaliste d'injection de faute où, grâce à un laser, l'attaquant pourrait retourner les 8 bits de ce port.

```
in_hi_sig <= in_hi;
lin_0_in <= in_hi_sig XOR fault_sig;
```

Le signal `fault_sig` est remonté jusque dans le fichier top, comme un port du composant `aes_core`, pour pouvoir générer l'attaque. Grâce au XOR, un bit à 1 dans le signal `fault_sig` retournera le bit correspondant dans le signal `in_hi` et créera une faute. Précisons que nous n'attaquons que la première colonne du DataUnit et que ce même signal `fault_sig` est laissé à `"00000000"` pour les trois autres colonnes.
Nous avons tout d'abord testé cette implémentation en modifiant le test bench VHDL qui instancie notre IP. Nous avons rajouté un process `fault`, qui incrémente un compteur à chaque front montant et qui le remet à zéro à chaque début de chiffrement. Après un certain nombre de cycles, nous injectons la faute de retournement de bit (signal `fault_sig` à `"11111111"`), pendant exactement 6 cycles afin que les deux registres DDR soient affectés par l'attaque.
En utilisant les données fournies par la publication 197 du FIPS (Federal Information Processing Standard), c'est à dire le texte d'entrée, la clée et le texte chiffré, nous pouvons vérifier que notre implémentation fonctionne. Si c'est la cas, la donnée d'entrée ne doit jamais changer pendant le chiffrement, le signal de sortie de `aes_core`, `error`, ne doit jamais passer à 1 et la donnée de sortie doit être différente de la donnée attendue.
Nous avons obtenu de tels résultats (voir figures 1 et 2) la donnée chiffrée, n'est pas celle attendue, le signal `s_broken` reste toujours à 0 et la donnée en entrée ne change pas. On peut aussi voir que le signal modélisant le retournement de bit, `fault_sig`, n'est actif que sur 6 cycles.

2.5 Embarcation du code

2.6 Conclusion

3 TP 2

3.1 Introduction

Dans cette partie nous allons effectuer une attaque par canaux auxiliaires sur un chiffrement AES. Cette attaque s'appelle Differential Power Analysis ou DPA, et consiste à étudier la consommation électrique du système visé sur de multiples exécutions et d'en déduire par des procédés statistiques l'information visée, ici la clé de chiffrement.

3.2 Théorie

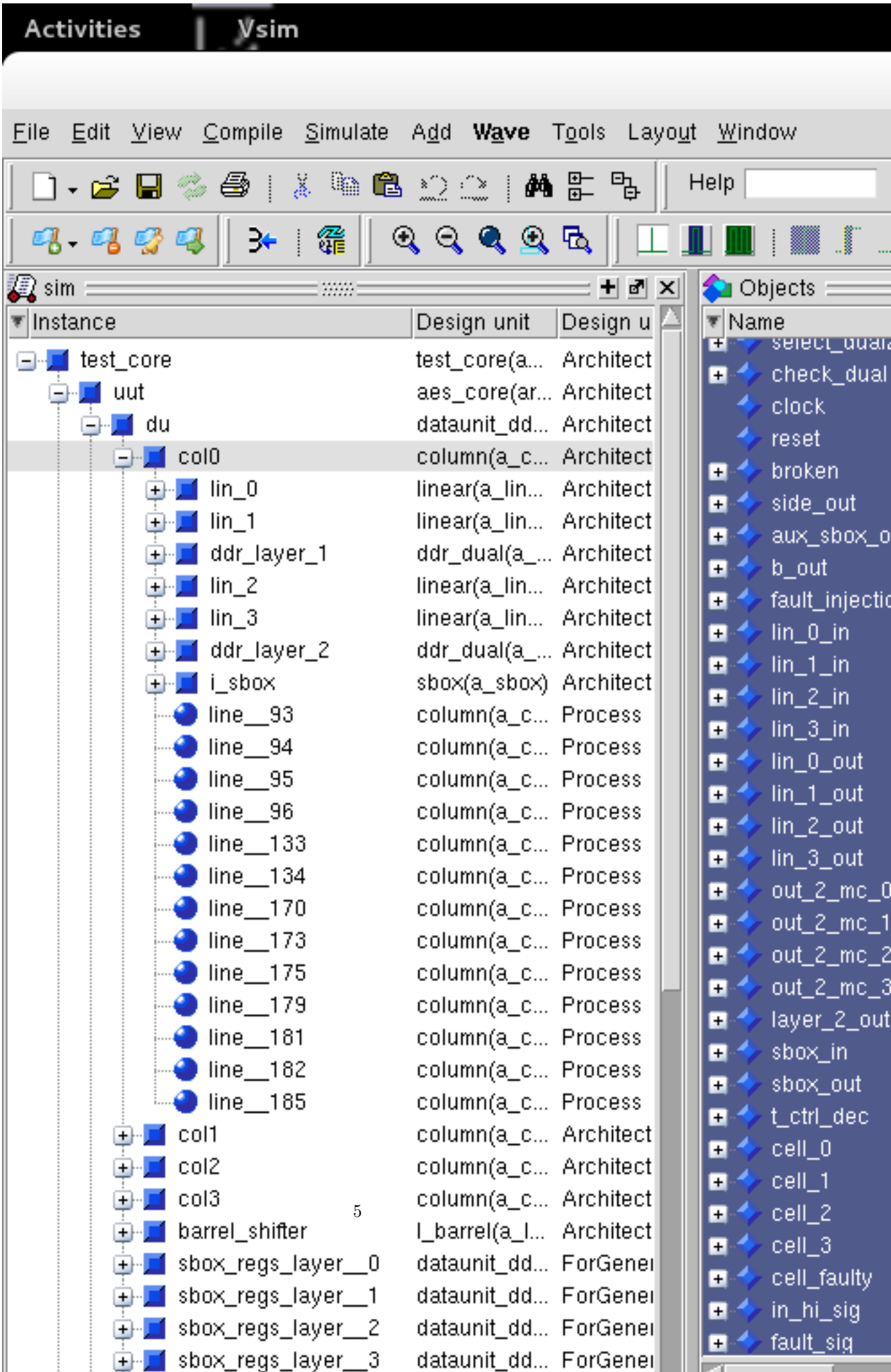
La sécurité d'un système de chiffrement dépend de l'algorithme employé et de son implémentation sur circuit électronique. Dans le cas de la DPA, nous allons utiliser des faiblesses d'implémentations pour en déduire la clé de chiffrement employée.

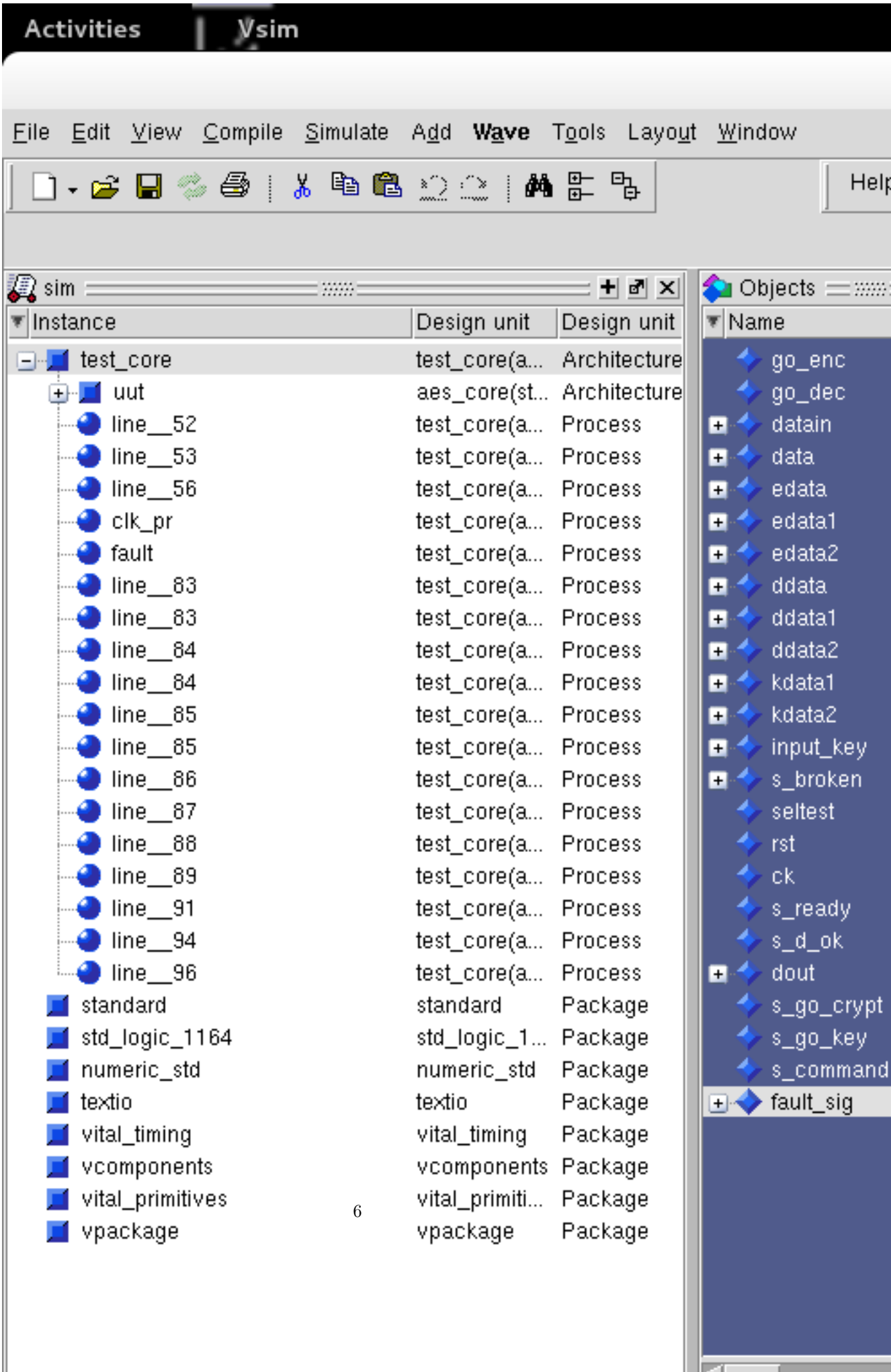
La SBox (Substitution Box) est le composant non linéaire principal de l'AES. Il s'agit d'une substitution d'octets, cette opération est effectuée juste après l'ajout de la clé à la donnée que l'on veut chiffrer. Il est donc possible, à partir du résultat de la première itération de l'AES qui ne dépendent que des données d'entrées et de la clé, de retrouver la clé.

La DPA est une attaque par canaux cachés qui utilisent la consommation électrique du circuit pour valider ou invalider des hyptohèses sur la clé. La consommation d'un circuit logique dépend grandement des données traitées. Dans notre cas, il n'y a pas de contre-mesure spécifique utilisée pour équilibrer la consommation. Ainsi, nous devrions être capables de réaliser une attaque DPA sur la SBox de l'AES. Néanmoins, nous n'attaquerons pas la totalité de la clé mais un seul octet, pour des questions de temps de calcul.

3.3 Travail réalisé

Tout d'abord, nous devons transformer les fichiers VHDL qui nous ont été donnés en une description au niveau des transistors, pour pouvoir en extraire une consommation électrique simulée. Cette étape est réalisée à l'aide de l'environnement Cadence.


Activities
Vsim
File Edit View Compile Simulate Add Wave Tools Layout Window
Help
sim
Instance
Design unit
Design u
test_core
test_core(a...
Architect
uut
aes_core(ar...
Architect
du
dataunit_dd...
Architect
col0
column(a_c...
Architect
lin_0
linear(a_lin...
Architect
lin_1
linear(a_lin...
Architect
ddr_layer_1
ddr_dual(a_...
Architect
lin_2
linear(a_lin...
Architect
lin_3
linear(a_lin...
Architect
ddr_layer_2
ddr_dual(a_...
Architect
i_sbox
sbox(a_sbox)
Architect
line__93
column(a_c...
Process
line__94
column(a_c...
Process
line__95
column(a_c...
Process
line__96
column(a_c...
Process
line__133
column(a_c...
Process
line__134
column(a_c...
Process
line__170
column(a_c...
Process
line__173
column(a_c...
Process
line__175
column(a_c...
Process
line__179
column(a_c...
Process
line__181
column(a_c...
Process
line__182
column(a_c...
Process
line__185
column(a_c...
Process
col1
column(a_c...
Architect
col2
column(a_c...
Architect
col3
column(a_c...
Architect
barrel_shifter
l_barrel(a_l...
Architect
sbox_regs_layer__0
dataunit_dd...
ForGener
sbox_regs_layer__1
dataunit_dd...
ForGener
sbox_regs_layer__2
dataunit_dd...
ForGener
sbox_regs_layer__3
dataunit_dd...
ForGener
Objects
Name
check_dual
clock
reset
broken
side_out
aux_sbox_o
b_out
fault_injectio
lin_0_in
lin_1_in
lin_2_in
lin_3_in
lin_0_out
lin_1_out
lin_2_out
lin_3_out
out_2_mc_0
out_2_mc_1
out_2_mc_2
out_2_mc_3
layer_2_out
sbox_in
sbox_out
t_ctrl_dec
cell_0
cell_1
cell_2
cell_3
cell_faulty
in_hi_sig
fault_sig


Activities
Vsim
File Edit View Compile Simulate Add Wave Tools Layout Window
Help
sim
Instance
Design unit
Design unit
test_core
test_core(a...
Architecture
uut
aes_core(st...
Architecture
line__52
line__53
line__56
clk_pr
fault
line__83
line__83
line__84
line__84
line__85
line__85
line__86
line__87
line__88
line__89
line__91
line__94
line__96
test_core(a...
Process
standard
std_logic_1164
numeric_std
textio
vital_timing
vcomponents
vital_primitives
vpackage
Package
Objects
Name
go_enc
go_dec
datain
data
edata
edata1
edata2
ddata
ddata1
ddata2
kdata1
kdata2
input_key
s_broken
seltest
rst
ck
s_ready
s_d_ok
dout
s_go_crypt
s_go_key
s_command
fault_sig

Une fois cette description générée, nous utilisons le simulateur de consommation électrique Synopsis Nanosim pour obtenir les profils de courants selon certains stimulis.

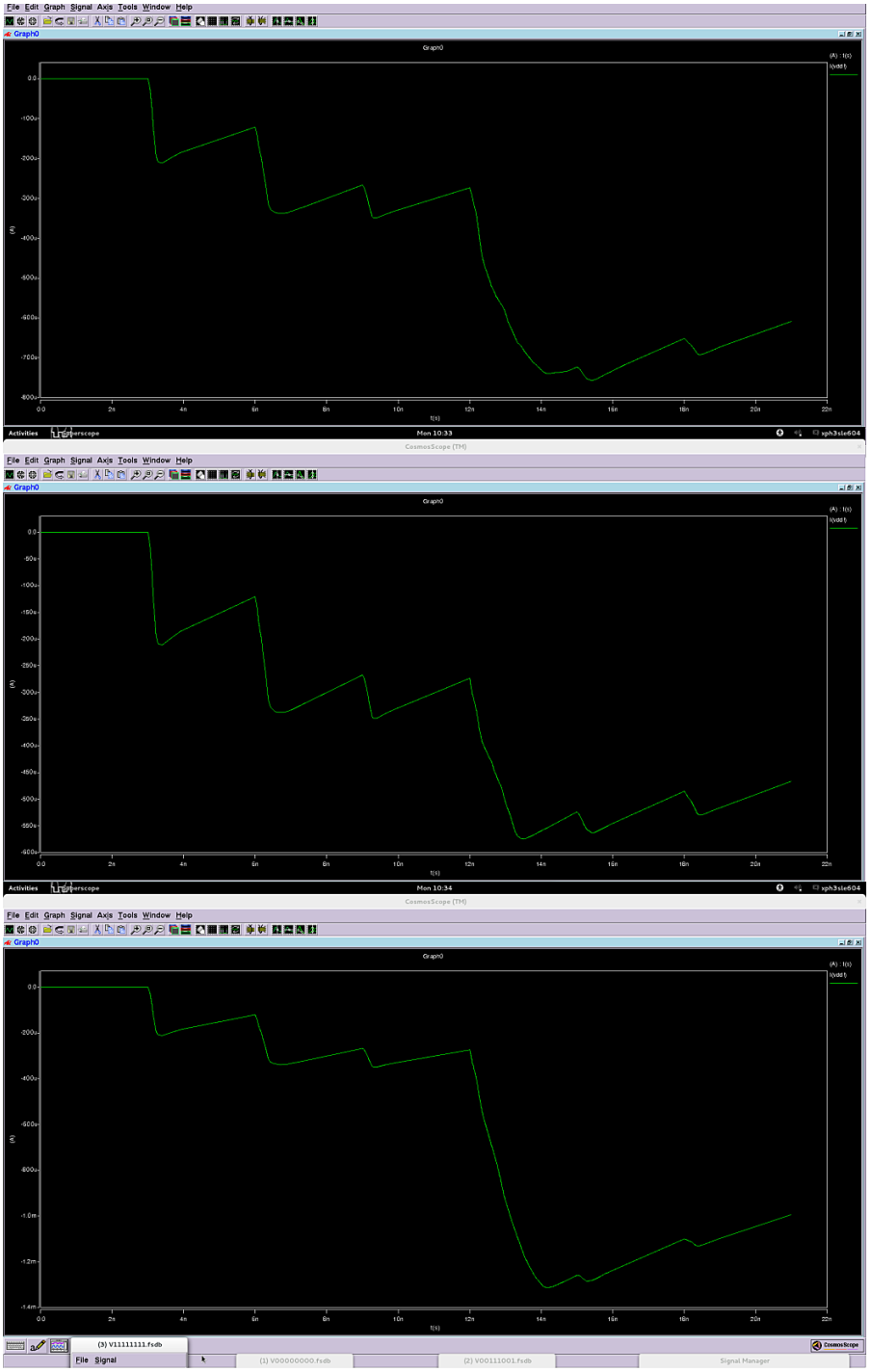


FIGURE 3 – Traces de courants pour différents stimuli

La figure 3 page 7 contient les simulations de courants pour différents stimuli (la pre-

mière image pour 0, la deuxième 57 et la dernière 255) pour une certaine clé de chiffrement AES. On remarque de légères différences, qui seront exploitées par la DPA.

Le logiciel d'analyse DPA fourni nous permet de retrouver les clés utilisées. Il accepte deux paramètres : l'index du bit attaqué et le chemin vers le dossier contenant les fichiers de simulation. L'outil lit un fichier de configuration contenant une liste des vecteurs qui seront utilisés pour la DPA. Ensuite, les données simulées sont collectées à partir des fichiers de simulation. Pour chaque hypothèse de clé, il partitionne les traces de simulation de chaque message selon la valeur du bit attaqué. Enfin, il évalue les différences entre les moyennes de chaque partition et sélectionne la valeur la plus haute.

3.4 Résultats

Suite à nos expérimentations nous obtenons les résultats suivants visibles sur la figure 4 page 9.

Pour obtenir la clé qui a le plus de chances d'être la clé réelle, il suffit de prendre celle qui apparait le plus souvent sur les différents lancements du programme avec des bits différents.

Par exemple dans le cas de la clé 4A, celle qui est apparait le plus souvent est la 58 (0b111010).

3.5 Conclusion

Dans ce TP, nous avons vu comment réaliser une attaque par canaux cachés DPA, et obtenir la clé de chiffrement. Pour se protéger de telles attaques, il faut rendre la consommation de de courant insensibles aux données d'entrée afin d'équilibrer la consommation de courant, et rendre inutile la DPA qui se base sur les différences. Néanmoins, en pratique cela reste très compliqué, car la plus légère différence peut être une faiblesse exploitable.

4 Conclusion Générale

Durant les séances de travaux pratiques, nous avons mis en évidence deux attaques classiques sur des circuits électroniques implémentant l'algorithme AES : l'attaque par faute et la DPA.

Cet algorithme, pourtant réputé incassable, devient très vulnérable dès qu'il est implementé dans un circuit électronique. Il est donc essentiel de prendre en compte la sécurité d'un circuit dès le début de sa conception, et de choisir les bonnes contre-mesures qui préservent la sécurité du chiffrement tout en ne négligeant pas les performances.

```
% bash unknown_rapid_dpa.sh
UnknownKey4A/:
bit 0: Maximum observed for Key = 58.
bit 1: Maximum observed for Key = 58.
bit 2: Maximum observed for Key = 58.
bit 3: Maximum observed for Key = 58.
bit 4: Maximum observed for Key = 58.
bit 5: Maximum observed for Key = 215.
bit 6: Maximum observed for Key = 58.
bit 7: Maximum observed for Key = 58.
UnknownKey4B/:
bit 0: Maximum observed for Key = 110.
bit 1: Maximum observed for Key = 110.
bit 2: Maximum observed for Key = 110.
bit 3: Maximum observed for Key = 110.
bit 4: Maximum observed for Key = 110.
bit 5: Maximum observed for Key = 131.
bit 6: Maximum observed for Key = 110.
bit 7: Maximum observed for Key = 110.
UnknownKey4C/:
bit 0: Maximum observed for Key = 155.
bit 1: Maximum observed for Key = 155.
bit 2: Maximum observed for Key = 155.
bit 3: Maximum observed for Key = 155.
bit 4: Maximum observed for Key = 155.
bit 5: Maximum observed for Key = 118.
bit 6: Maximum observed for Key = 155.
bit 7: Maximum observed for Key = 155.
UnknownKey4D/:
bit 0: Maximum observed for Key = 184.
bit 1: Maximum observed for Key = 143.
bit 2: Maximum observed for Key = 184.
bit 3: Maximum observed for Key = 184.
bit 4: Maximum observed for Key = 184.
bit 5: Maximum observed for Key = 85.
bit 6: Maximum observed for Key = 184.
bit 7: Maximum observed for Key = 184.
UnknownKey4E/:
bit 0: Maximum observed for Key = 239.
bit 1: Maximum observed for Key = 239.
bit 2: Maximum observed for Key = 239.
bit 3: Maximum observed for Key = 239.
bit 4: Maximum observed for Key = 239.
bit 5: Maximum observed for Key = 2.
bit 6: Maximum observed for Key = 239.
bit 7: Maximum observed for Key = 239.
```

FIGURE 4 – Résultats de l'attaque DPA sur l'ensemble de données du groupe 4