# Ne sortez pas sans vos masques ! Description d'une contre-mesure contre les attaques par canaux auxilliaires

Nicolas Bordes
nicolas.bordes@univ-grenoble-alpes.fr

Université Grenoble Alpes

**Résumé.** Les attaques par canaux auxiliaires sont des attaques particulièrement adaptée aux systèmes embarqués. Celles-ci, grâce à l'observation de grandeurs physiques lors de l'exécution d'un programme, peuvent permettre d'extraire des données sensibles de l'appareil ciblé. Lorsqu'exécutées sur un appareil non protégé, de telles attaques peuvent aboutir à l'affaiblissement, voire au contournement, des moyens cryptographiques mis en œuvre. Cet article propose une description générale des principes et enjeux d'une contremesure générique à ces attaques, notamment dans le contexte de la cryptographie symétrique : le masquage.

## 1 Introduction

Aussi bien conçu que puisse-t-être un algorithme cryptographique, par exemple une fonction de chiffrement par bloc comme l'AES, son implémentation et son exécution peuvent être la source de plusieurs vulnérabilités inhérentes au contexte de mise en œuvre. Ainsi, bien qu'il soit indispensable d'établir des modèles de cryptanalyses de haut niveau permettant d'évaluer la sécurité d'un algorithme, il ne faut pas négliger les problèmes de sécurité qui peuvent se poser lorsque l'algorithme devient programme, puis processus. Les attaques actives comme l'exploitation de vulnérabilités logicielles via les différents points d'entrée de données ou la manipulation matérielle (chevaux de Troie matériels, attaques par injection de fautes, ...) peuvent permettre d'altérer le comportement légitime, fragilisant ou contournant les moyens cryptographiques employés. Mais quand bien même le programme s'exécuterait dans des conditions où il n'est pas possible d'agir directement sur l'exécution de celui-ci, un certain nombre d'attaques physiques passives peuvent être déployées. Les attaques par canaux auxiliaires en font parties.

Elles constituent une classe d'attaques ayant pour objectif d'exploiter les variations de grandeurs physiques lors de l'exécution du programme. Celles-ci peuvent être, par exemple, la durée d'exécution, la consommation

électrique, le rayonnement électromagnétique [26, 27] et même les ondes sonores émises par les vibrations des composants électroniques [17]. Le but de l'attaquant est d'extraire, grâce aux observations de ces émanations physiques, des informations qui ne lui sont pas classiquement accessibles dans un modèle où la fonction cryptographique est considérée «en boîte noire», c'est-à-dire ne lui donnant accès qu'aux valeurs d'entrée et de sortie. La découverte même partielle, non seulement de clés secrètes mais plus généralement des valeurs intermédiaires d'un tel programme, peut avoir pour conséquence de réduire considérablement ou totalement les propriétés de sécurité attendues de celui-ci, mettant à mal le rôle de la cryptographie dans la protection de l'appareil et de ses données. Un exemple récent se retrouve dans l'attaque menée par Lomné et Roche contre les clés de sécurité Titan [29]. Dévoilée en janvier 2021, cette attaque exploite l'observation des rayonnements électromagnétiques émis lors du calcul de signatures ECDSA légitimes. Elle aboutit à la récupération de la clé secrète permettant de créer de nouvelles signatures ECDSA valides, et donc de cloner la clé de sécurité.

Ce genre d'attaques, sauf pour le cas de celles visant spécifiquement la durée d'exécution (qui peut être, dans certains cas, mesurée à distance), obligent l'attaquant à avoir un accès physique au matériel sur lequel s'exécute le programme attaqué. Ainsi, les appareils électroniques embarqués sont les plus susceptibles d'être visés par de telles attaques. Ces appareils sont par exemple des cartes à puce, des portefeuilles matériels de cryptomonnaies, des objets connectés en tout genre ou bien encore des composants électroniques présents dans des véhicules. En effet, ces appareils peuvent être facilement transportés (donc potentiellement volés/substitués), sont souvent accessibles (au moins en partie) depuis un environnement non contrôlé ou bien sont amenés à s'y déplacer. De plus, ces appareils sont généralement spécialisés, en ce sens qu'ils sont dédiés à l'exécution d'un nombre réduit de tâches et qu'ils sont rarement munis de plusieurs processeurs calculant en parallèle. Cette considération, combinée avec la capacité qu'un attaquant a à se trouver au plus proche de l'unité de calcul visée, permet à celui-ci d'obtenir des mesures fiables et précises grâce à des niveaux de perturbations relativement faibles.

### 1.1 Contremesures

Il ne faut pas perdre de vue que toutes les contremesures dont nous allons parler doivent être considérées dans l'optique de les appliquer notamment à des systèmes embarqués. Les contextes de production et d'utilisation de ceux-ci peuvent imposer des contraintes plus fortes encore

que pour les applications cryptographiques ayant vocation à s'exécuter sur des systèmes classiques. Ces contraintes s'expriment en terme de capacité de calcul, d'espace mémoire disponible, de taille physique ou de prix de fabrication. Ainsi, le critère du coût au sens large des contremesures est crucial. Les contremesures les plus communes ont pour objectif principal de limiter au maximum l'exploitabilité du signal mesuré par un attaquant et de faire augmenter le coût financier, technique, en temps ou en opportunité des attaques par canaux auxiliaires. Les trois approches majeures employées sont les suivantes :

**Réduction du signal utile.** Même si cette première approche peut sembler la plus évidente, vouloir réduire les émanations provenant du composant se révèle être très compliqué en pratique. Bien qu'il soit possible de mettre en place, par exemple, un blindage électromagnétique empêchant le rayonnement du composant d'être exploité par un attaquant, plusieurs problèmes surviennent. Tout d'abord, cette contremesure nécessite une intervention au niveau de la chaîne de production du composant et ne s'adapte pas à n'importe quel composant déjà sur le marché. De plus, cette stratégie peut être sensible à l'intervention directe d'un attaquant qui endommagerait volontairement le blindage afin de limiter son efficacité. Mais surtout, cette contremesure est spécifique au type d'attaque par canal auxiliaire (ici, les émanations électromagnétiques). Il y a donc un surcoût potentiel pour se protéger contre plusieurs d'entre elles.

**Prévention de la reproductibilité de la mesure.** La mesure de l'évolution sur un intervalle de temps de la grandeur physique considérée est classiquement appelée *trace*. Dans la pratique, il est rare qu'une seule trace suffise à l'attaquant et il doit souvent, afin d'améliorer la précision et réduire le bruit, répéter les mesures. Mais pour que ces multiples traces permettent réellement d'améliorer les chances de succès de l'attaque, l'attaquant doit réussir à les synchroniser. Cette synchronisation est utile pour faire en sorte que chaque point de mesure corresponde pour chaque trace à la fuite de la même donnée et donc que l'accumulation de traces augmente effectivement le rapport signal sur bruit. C'est de ce constat que l'idée des contremesures de désynchronisation est apparu. Le but n'est plus d'empêcher les fuites lors de l'exécution mais de décaler de manière non-déterministe le moment où les données sensibles sont manipulées, rendant l'analyse des traces beaucoup plus difficiles. En 2009 et 2010, Coron et Kizhvatov [13, 14] ont présenté une contremesure visant à ajouter du délai aléatoirement dans le flot d'exécution. Néanmoins, cette approche a

plusieurs limitations. Si une seule trace est suffisante, comme cela peut être le cas selon le contexte [24], cette contremesure n'est plus suffisante. Aussi, même si plusieurs traces sont nécessaires en pratique, le signal capturé contient toujours les informations utiles à l'attaquant. L'utilisation de techniques d'analyse de signal et de reconnaissance de motif peuvent aider à contourner la désynchronisation des traces, permettant de mener à bien l'attaque [10, 16].

**Augmentation du bruit de mesure.** La dernière contremesure envisageable consiste à créer ou amplifier le bruit présent lors d'une mesure. En faisant ainsi, l'objectif est de faire en sorte que le nombre de traces nécessaire pour qu'une attaque réussisse devienne trop grand, donc trop coûteux pour être réalisable en pratique. Il est possible d'imaginer de tels dispositifs amplificateurs de bruit pour chaque type de grandeur physique (consommation électrique, émissions électromagnétiques. . . ) mais il existe une approche générique pour atteindre ce but : le *masquage*. Dans un premier temps, nous allons décrire les grands principes du masquage et discuter de la pertinence des modèles de sécurité associés en Section 2. Ensuite, en Section 3, nous allons voir comment mettre en œuvre cette contre-mesure. Pour finir, nous discuterons en Section 4 de son coût et de stratégies visant à le réduire.

## 2 Principes et modèle de sécurité

### 2.1 Le principe du masquage

Le principe de base du masquage est d'utiliser un algorithme de partage de secret pour diviser la donnée sensible en plusieurs parties et de faire les calculs sur ces parties plutôt que sur la donnée originale. Ces parties sont, lorsque prises séparément, statistiquement indépendantes. Cette approche a été introduite simultanément par Goubin et al. [19] et Chari et. al. [12] en 1999. Le terme masquage a été utilisé pour désigner ce principe de contre-mesure par Messerges [30] en 2001 dans lequel il le développe et l'applique aux finalistes d'AES.

Le partage de secret booléen est utilisé dans ces articles et l'est toujours dans les travaux plus récents. Il consiste à remplacer la valeur que l'on souhaite masquer $x$ par deux valeurs $x'$ et $r_x$. Le masque $r_x$ est choisi aléatoirement et uniformément tandis que la valeur de $x'$ est calculée comme étant $x' = x \oplus r_x$ (où $\oplus$ désigne le OU exclusif bit à bit). De cette manière $x'$ et $r_x$ suivent tous les deux une distribution uniforme.

La Figure 1 montre un circuit de masquage ainsi que l'opération inverse permettant de retrouver la donnée $x$ à partir de la paire $(x', r_x)$.


**Fig. 1.** Un circuit de masquage de $x$ en $\boldsymbol{x} = (x', r_x)$ et son opération inverse. \$ dénote une valeur prise uniformément aléatoirement.

Ce principe peut être étendu avec ce qu'on appelle un masquage à l'ordre $d$ : dans ce cas, $x$ est partagé en $d+1$ parties $x_i$ : dans un premier temps chaque $x_i$ sauf $x_d$ est tiré aléatoirement et uniformément ; la valeur de $x_d$ est telle que $x = \bigoplus_{i=0}^{d} x_i$. Chaque $x_i$, y compris $x_d$, suit ainsi une distribution uniforme. En réalité, chaque sous-ensemble de moins de $d$ parties $x_i$ suit aussi une loi uniforme. Ainsi, la connaissance de moins de la totalité du partage de $x$ ne donne aucune information sur la valeur d'origine de $x$.

### 2.2 Le *probing model* et la *d-privacy*

L'intérêt de procéder comme décrit précédemment vient du constat suivant : un attaquant n'ayant accès, via des attaques par canaux auxiliaires, qu'à moins de $d$ valeurs parmi les $d+1$ d'un partage n'observera en réalité qu'une distribution uniforme, et il ne pourra donc pas déduire d'informations sur la valeur "réelle" de $x$. Ce modèle de sécurité, où l'on considère que l'attaquant à accès parfaitement à un certain nombre de valeurs intermédiaires simultanément s'appelle le *probing model* et a été introduit en 2003 par Ishai, Sahai et Wagner [22]. Dans ce modèle, on représente notre algorithme par un circuit composé de portes logiques reliées par des fils et on donne la possibilité à l'attaquant de mettre en place jusqu'à $d$ sondes sur les fils de ce circuit. On dit d'un circuit qu'il est *d-private* quand un attaquant capable de poser jusqu'à $d$ sondes sur le circuit n'observe que des valeurs dont la distribution ne dépend pas des données que l'on souhaite protéger par le masquage.

Par exemple, le circuit présenté en Figure 2 prend en entrée deux valeurs masquées à l'ordre $d = 1$ : $\boldsymbol{a} = (a_0, a_1)$ et $\boldsymbol{b} = (b_0, b_1)$. La sonde $p$, symbolisée en rouge, permet à un attaquant de lire la valeur en sortie de la première porte OU exclusif. Cette valeur étant $a_0 \oplus a_1 = a$, elle dépend

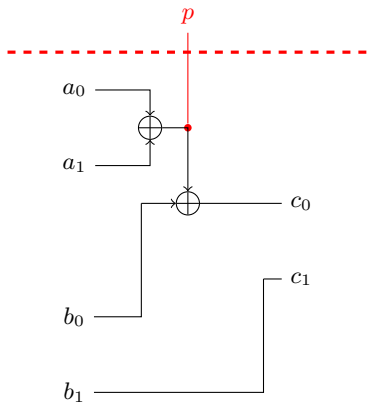


**Fig. 2.** Exemple d'un circuit **non-sécurisé** sur des données $\boldsymbol{a}$ et $\boldsymbol{b}$ masquées à l'ordre 1. $p$ est une sonde possible sur ce circuit.

directement d'une des deux données. Ainsi ce circuit n'est pas 1-private. Nous verrons plus tard plus tard des circuits qui le sont.

## 2.3 La pertinence du *probing model* et d'un ordre de masquage élevé

Le *probing model*, qui suppose que l'attaquant a accès parfaitement à un nombre limité de sondes parfaites sur le circuit, permet de faciliter les analyses de sécurité formelles. Néanmoins, l'attaquant n'a accès en réalité qu'à des observations bruitées mais peut réaliser plusieurs observations et ces mesures concernent la totalité du circuit simultanément. Même si le *probing model* peut sembler trop éloigné de la réalité, en 2014 Duc et al. [15] ont unifié deux modèles : le *probing model* vu précédemment et le *noisy leakage model*, un modèle beaucoup plus proche de la réalité physique.

Ainsi, sous les bonnes hypothèses, le nombre de mesures nécessaires à la réussite d'une attaque par canal auxiliaire grandit exponentiellement en l'ordre $d$ du masquage défini dans le *probing model*. Sachant que le nombre de traces nécessaire est un bon estimateur du coût d'une attaque, il apparaît donc que l'ordre de masquage joue un rôle important dans la résistance d'une implémentation aux attaques par canaux auxiliaires.

## 3 Mise en œuvre du masquage

### 3.1 Construction d'une variante masquée d'un algorithme

Pour protéger un circuit, comme par exemple le circuit de la Sbox de KECCAK-f [8] en Figure 3, il suffit donc de "traduire" celui-ci pour qu'il n'agisse plus sur les données directement mais sur un masquage de celles-ci, produisant des sorties elles aussi masquées. On appellera *schéma de masquage*, ou plus simplement *schéma*, la description formelle d'un circuit de masquage et *gadget* l'instanciation d'un schéma pour un ordre fixé.

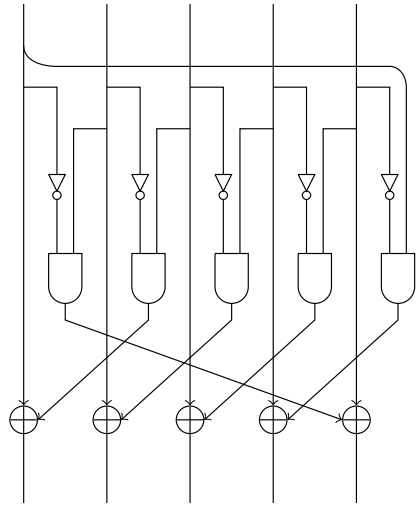

**Fig. 3.** Circuit de la Sbox de KECCAK-f [8].

Une approche fréquemment utilisée consiste à concevoir des versions masquées d'opérations élémentaires de telle manière qu'elles sont individuellement correctes et sécurisées puis de faire en sorte que leur composition le soit aussi. Pour l'exemple de la Figure 3, les opérations élémentaires nécessaires sont le NON, le OU exclusif ainsi que le ET. Avec un masquage booléen tel que décrit précédemment, certaines de ces opérations élémentaires sont plus "faciles" et moins coûteuses que d'autres à masquer, comme nous allons le voir par la suite.

Le schéma de masquage pour une fonction $f$ (par exemple, une fonction de chiffrement par bloc) doit satisfaire deux propriétés : 1) il doit être correct, c'est-à-dire qu'il calcule bien un partage du résultat de la fonction

$f$ sur les entrées ; 2) il doit être sécurisé, c'est-à-dire qu'il n'existe pas d'attaques à un ordre plus petit qu'un ordre $t$ fixé par le concepteur (souvent $t$ est égal à $d$, l'ordre du masquage).

Par soucis de lisibilité, et lorsqu'ils ne sont pas explicitement définis, les indices des parties d'un masquage seront omises et prennent leurs valeurs entre 0 et $d$, l'ordre du masquage, inclus.

**Masquage d'une porte NON.** La porte NON est certainement la plus simple et la moins coûteuse à masquer : à partir de $\boldsymbol{a} = (a_0, \cdots, a_d)$ la donnée masquée, $\boldsymbol{a'} = (\neg a_0, a_1, \cdots, a_d)$ est un schéma trivialement correct et sécurisé pour l'opération NON.

**Masquage d'une porte OU exclusif.** Étant donné le partage de $a$ et de $b$, respectivement $(a_i)$ et $(b_i)$, on peut calculer un partage $(c_i)$ de $c := a \oplus b$ (où $\oplus$ désigne le OU exclusif) en calculant $c_i = a_i \oplus b_i$, pour chaque $c_i$. Ce schéma est correct :

$$c = \bigoplus_{i=0}^{d} c_i = \bigoplus_{i=0}^{d} a_i \oplus \bigoplus_{i=0}^{d} b_i = a \oplus b$$

Il est aussi *d-private* car aucun ensemble de $d$ valeurs intermédiaires dans ce circuit (composé de seulement $d + 1$ portes OU exclusif) ne permet d'obtenir autre chose qu'une distribution uniforme.

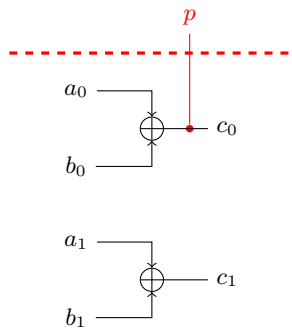


**Fig. 4.** Exemple d'un circuit calculant le OU exclusif sur des données $\boldsymbol{a}$ et $\boldsymbol{b}$ masquées à l'ordre 1. $p$ est une sonde possible sur ce circuit.

La Figure 4 présente un circuit de masquage à l'ordre 1 prenant en entrée deux données masquées $\boldsymbol{a} = (a_0, a_1)$ et $\boldsymbol{b} = (b_0, b_1)$, tel que

$a = a_0 \oplus a_1$ et $b = b_0 \oplus b_1$. Ce circuit permet de calculer $\boldsymbol{c} = (c_0, c_1)$, le résultat masqué de l'opération $a \oplus b$. La sonde $p$ présente sur ce circuit donne à l'attaquant l'information sur $a_0 \oplus b_0$, ce qui ne permet pas à celui-ci d'obtenir une quelconque information sur $a$, sur $b$ ou même sur le résultat $c$. La donnée d'au minimum une autre sonde est nécessaire pour mener à une attaque, le circuit est donc 1-private mais n'est pas 2-private.

**Masquage d'une porte ET.** Le calcul du ET logique,

$$c := ab = \bigoplus_{i=0}^{d} \bigoplus_{j=0}^{d} (a_i b_j)$$

présente une difficulté supplémentaire. Pour montrer comment survient ce problème, essayons de définir naïvement et de manière analogue au OU exclusif masqué, un schéma pour le ET :

$$\text{pour chaque } c_j,\ c_j = \bigoplus_{i=0}^{d} (a_i b_j)$$

Ce schéma est bien correct ($\bigoplus c_j = c$). Par contre, chaque $c_j$ peut être réécrit en

$$b_j \bigoplus_{i=0}^{d} a_i = ab_j$$

Chaque $c_j$ est donc biaisé et révèle de l'information sur la distribution de $b$.

Par exemple, sur la Figure 5 l'attaquant observe grâce à la sonde $p$ la valeur $c_0 = a_0 b_0 \oplus a_1 b_0 = ab_0$. Si lors d'une exécution la valeur lue est 1, l'attaquant peut en déduire que $a$ est égal à 1. Ainsi, ce schéma n'est pas 1-*private* car il existe une attaque d'ordre 1, c'est-à-dire une seule sonde permettant de retrouver de l'information sur une entrée.

Pour empêcher de telles attaques, tous les schémas sécurisés connus pour le ET logique ont recours à des "masques de rafraîchissement" au cours du calcul. Ces masques sont des valeurs générées aléatoirement et uniformément à chaque exécution et sont nécessaires à la sécurité du schéma. Le nombre de ces masques de rafraîchissement dépend du schéma et, comme nous le verrons plus en détails par la suite, va grandement influer sur les performances de celui-ci.

Par exemple, dans le même article que celui formalisant le *probing model*, Ishai, Sahai et Wagner [22] proposent un schéma de masquage (couramment nommé ISW) pour le ET logique a un ordre quelconque

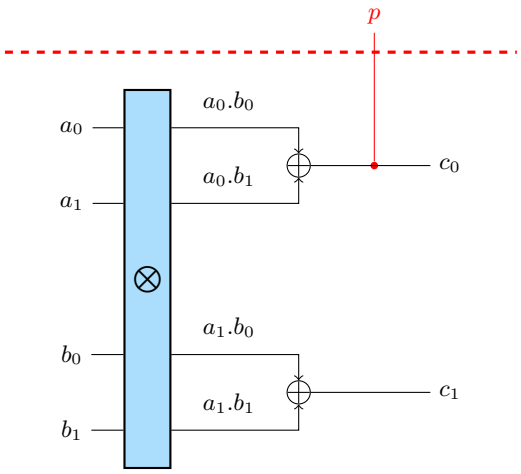


**Fig. 5.** Exemple d'un circuit calculant de manière **non-sécurisée** le ET logique sur des données $\boldsymbol{a}$ et $\boldsymbol{b}$ masquées à l'ordre 1. $p$ est une sonde sur ce circuit. Par soucis de simplicité, le bloc $\otimes$ est le circuit calculant tous les $a_i b_j$ (une porte ET par élément en sortie).

$d$. ISW à l'ordre $d$ nécessite la génération de $d(d+1)/2$ masques de rafraîchissement. Au fil du temps, des nouveaux schémas [3, 5] ont été proposés permettant de réduire ce coût.

La Figure 6 et la Figure 7 montrent des exemples de gadgets sécurisé à l'ordre 1, 2 et 3. À partir de deux entrées $\boldsymbol{a}$ et $\boldsymbol{b}$ toutes les deux masquées à l'ordre 3, il permet d'obtenir le résultat masqué $\boldsymbol{c}$ de leur ET logique. La première étape, qui consiste en le calcul de tous les $a_i b_j$ (une porte ET par $a_i b_j$), est dénotée par le bloc $\otimes$ par soucis de lisibilité du circuit. Ces gadgets utilisent 1, 2, et 4 masques de rafraîchissement $r_k$ respectivement contre 1, 3 et 6 pour ISW. Ces gadgets sont prouvés comme étant $d$-private, c'est-à-dire qu'il n'existe pas d'ensemble de $d$ sondes ou moins permettant d'obtenir des informations sur les données.

**Composition des gadgets.** En composant de manière adaptée ces deux portes logiques masquées, il est possible d'implémenter n'importe quel circuit, par exemple un circuit permettant de calculer un chiffrement par l'AES. Le problème de la composition de gadgets est cependant loin d'être trivial et nous ne rentrerons pas dans ses détails ici. Il est néanmoins important de retenir que si deux gadgets sont *d-private*, cela ne suffit pas à déduire que leur composition l'est aussi.

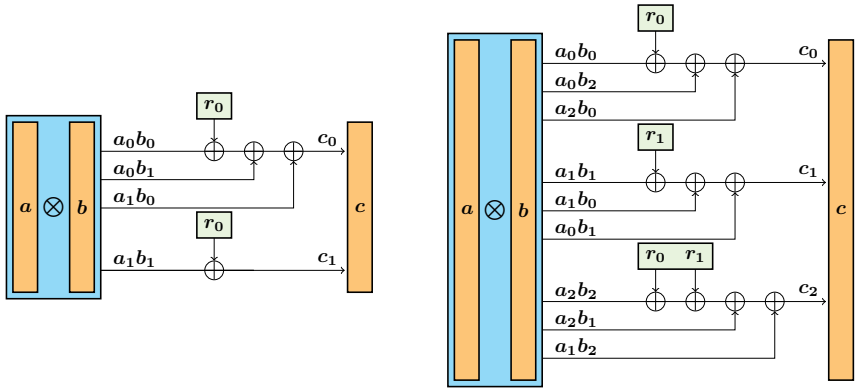


**Fig. 6.** Exemple de gadget à l'ordre $d = 1$ et $d = 2$ pour le ET logique (schéma à l'ordre 2 de Belaïd et al. [4]). Les $r_k$ désignent les masques de rafraîchissement.

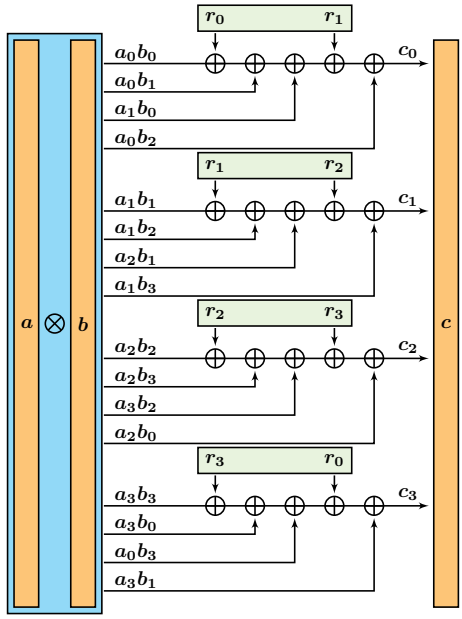


**Fig. 7.** Exemple de gadget à l'ordre $d = 3$ pour le ET logique où les $r_k$, $0 \leq k \leq 3$ désignent les masques de rafraîchissement (schéma de Barthe et al. [3], illustration par Pierre Karpman)

Pour garantir la composabilité, des conditions supplémentaires sont nécessaires. Par exemple, deux propriétés nommées *non-interférence* et *non-interférence forte* à l'ordre $d$ (respectivement abrégées en *d-NI* et *d-SNI*) permettent d'obtenir des compositions sécurisées dans le *probing model* [2]. En effet, s'il n'est pas possible de prouver que la composition de deux gadgets $d$-private l'est aussi, la composition d'un gadget $d$-SNI avec un gadget $d$-NI est lui-même $d$-SNI. Sachant qu'un gadget $d$-SNI est aussi $d$-private (mais que la réciproque n'est pas vraie), cela permet de composer des gadgets en s'assurant que le résultat est $d$-private.

Les travaux de Cassiers et Standaert [11] proposent encore une autre approche pour la composition de gadgets, appelée *Probe Isolating Non-Interference* (PINI).

**Étape finale : masquage du circuit** Une fois que les gadgets de toutes les opérations élémentaires nécessaires ont été conçus, il suffit de remplacer chaque porte (c'est-à-dire pour l'exemple de la Figure 3, 5 portes ET, 5 portes OU exclusif et 5 portes NON) par le gadget masqué correspondant en ayant par ailleurs vérifié que leur composition est sécurisée.

### 3.2 Analyse de la correction et de la sécurité d'un schéma

La propriété de correction est souvent la plus simple à vérifier parce qu'il suffit, dans le cas d'un masquage booléen comme décrit précédemment, de calculer le OU exclusif formel du partage obtenu et de vérifier qu'il correspond bien au résultat attendu de la fonction $f$ sur les entrées considérées.

La propriété de sécurité est quant à elle, plus compliquée à vérifier. Elle dépend tout d'abord du modèle de sécurité choisi. Ici, nous allons voir le cas du *probing model*. Une manière générique de prouver la sécurité de n'importe quel schéma est de vérifier que chaque ensemble de $d$ sondes ou moins ne constitue pas une attaque contre la propriété voulue (*d-privacy*, *d-NI*, *d-SNI*. . . ). Néanmoins, le nombre de ces ensembles de sondes grandit très rapidement en fonction de l'ordre $d$ de masquage. En effet, l'augmentation est factorielle en le nombre de valeurs intermédiaires dans le circuit et, comme nous le verrons par la suite, le nombre d'opérations (et donc le nombre de valeurs intermédiaires) d'un gadget pour le ET logique augmente au moins quadratiquement en l'ordre $d$ du schéma étudié. Par exemple, pour les meilleurs schémas de masquage du ET logique défini par Barthe et al. [3] et dans nos propres travaux [9], le nombre total d'ensembles différents de $d$ sondes ou moins est de $2^{14}$, $2^{40}$ et $2^{62}$ à l'ordre

respectivement $d = 3$, $d = 7$ et $d = 10$. La vérification générique d'un gadget issu d'un schéma de masquage instancié à ordre élevé $d$ devient alors très vite une difficulté pratique : celle d'énumérer tous les ensembles de $d$ sondes ou moins.

En 2019, Barthe et al. [1] présentent maskVerif, un outil qui permet de tester la sécurité $d$-NI et $d$-SNI de schémas.

**Un nouvel outil de vérification de gadgets** Dans des travaux récents [9], Pierre Karpman et moi-même présentons une approche permettant la vérification exhaustive de la sécurité (*d-NI* ou *d-SNI*) de gadgets. Cette vérification se fait plus efficacement et à des ordres plus élevés que ne peut le faire maskVerif. Conformément à ce qui a été expliqué précédemment, elle a pour vocation à être appliquée à des gadgets d'opérations élémentaires pouvant par la suite être utilisés par composition dans de plus gros circuit. Nous allons décrire succinctement son fonctionnement ici.

Dans un premier temps, l'outil implémentant cette approche reçoit en entrée une description du gadget à analyser. Par exemple, le gadget en Figure 7 se décrit via le fichier suivant :

```
ORDER = 3
MASKS = [r0, r1, r2, r3]
s00 r0 s01 s10 r1 s02
s11 r1 s12 s21 r2 s13
s22 r2 s23 s32 r3 s20
s33 r3 s30 s03 r0 s31
```

Les deux premières lignes donnent l'ordre du masquage et les masques de rafraîchissement utilisés par le circuit. Ensuite, chaque ligne correspond à une sortie $c_i$ et les `sij` correspondent aux $a_i b_j$. Chaque espace dénote la présence d'un OU exclusif. L'ordre des opérations est importante et celles-ci se font, en l'absence de parenthésage, de gauche à droite.

Une premier traitement très rapide transforme cette description du gadget en une paire de fichiers `C` décrivant notamment les sondes à analyser sur ce circuit (voir exemple en Annexe A).

Finalement le programme `binverif`, compilé en utilisant la paire de fichiers générés précédemment, prend en option le type de vérification ($d$-NI ou $d$-SNI) ainsi que le nombre de threads à utiliser. Ce programme va vérifier tous les ensembles de sondes à la recherche d'une attaque. Sa durée d'exécution est environ 3 ordres de grandeur plus rapide que maskVerif, aboutissant à la vérification de gadgets à des ordres plus élevés (jusqu'à l'ordre 11) par rapport à ce qui a pu être fait par ailleurs.

Par exemple, sur le même processeur et avec le même nombre de threads (ici 4), la vérification d'un même schéma $d$-SNI à l'ordre $d = 8$ dure 14 minutes avec notre outil contre 13 jours pour maskVerif.

L'amélioration des performances de vérification apportée par cet outil repose sur cinq piliers :

— une condition nécessaire et suffisante facilement vérifiable pour qu'un ensemble de sondes constitue une attaque contre la sécurité du schéma qui étend le modèle proposé par Belaïd et al à CRYPTO 2017 [5] ;
— un filtrage en amont des sondes, réduisant le nombre d'ensemble de sondes à explorer sans perdre en exhaustivité ;
— une énumération efficace des sondes via des codes de Gray combinatoriaux ;
— une implémentation optimisée utilisant des instructions vectorielles (plus particulièrement les extensions Intel AVX) permettant de vérifier environ $2^{27}$ ensemble de sondes par seconde ;
— et le caractère parallélisable de cette implémentation.

Cet outil est disponible publiquement [1] et sa description détaillée est en cours de publication mais déjà disponible sur une archive ouverte.[2]

Des outils de vérification adoptant d'autres approches et d'autres modèles existent. Par exemple, Knichel et al. [25] proposent un outil nommé SILVER permettant de vérifier de manière globale la sécurité d'un circuit dans un modèle prenant en compte certaines interactions dues à des considérations physiques ou micro-architecturales qui peuvent être favorables à l'attaquant et qui sont particulièrement pertinentes pour les implémentations matérielles. Mais cela impacte les performances de SILVER et il n'a pas été utilisé par les auteurs pour vérifier des schémas a un ordre supérieur à 3. D'autres outils encore [6, 7] s'intéressent plus particulièrement à la vérification de la sécurité d'un enchaînement de gadgets dont la sécurité doit être prouvée par ailleurs.

## 4 Coût du masquage et son optimisation

### 4.1 Le coût du masquage

Comme nous l'avons vu, le masquage est une contre-mesure générique contre les attaques par canaux auxiliaires, mais cela a un coût. Pour réaliser un OU exclusif masqué à l'ordre $d$, il faudra réaliser $d + 1$ OU

1. https://github.com/NicsTr/binary_masking/
2. https://eprint.iacr.org/2019/1165

exclusif donc un surcoût de $d$ opérations. L'impact est encore plus grand pour le calcul du ET logique ($c = ab$). Les meilleurs schémas de l'état de l'art ont besoin de $(d+1)^2$ ET logique pour calculer dans un premier temps chacun des termes $a_i b_j$. Ensuite il y a un surcoût en OU exclusif, utilisés pour "compresser" ces $(d+1)^2$ $a_i b_j$ en les $d+1$ $c_i$. Ce surcoût est donc d'au moins $d(d+1)$ et il va encore augmenter d'au moins 2 pour chaque masque de rafraîchissement introduit : en effet, pour pouvoir maintenir la propriété de correction de notre schéma ($\bigoplus c_i = c$) chacun de ces masques doit apparaître un nombre pair de fois dans le calcul des $c_i$. Par exemple, dans le schéma décrit en Figure 7, il y a 4 masques de rafraîchissement ce qui donne un coût total de 16 ET logique et 20 OU exclusif.

Ainsi, le nombre de masques de rafraîchissement introduits est un facteur déterminant dans le coût d'un schéma. D'autant plus que leur génération elle-même peut coûter cher. En effet, il n'est pas possible de faire appel directement à un générateur de nombres pseudo-aléatoires efficace classique car les opérations internes de celui-ci doivent être elles-mêmes protégées contre les potentielles attaques par canaux auxiliaires. Deux solutions sont possibles : 1) créer un circuit masqué protégeant le générateur pseudo-aléatoire ; 2) utiliser un générateur de nombres aléatoires matériel, aussi appelé *True Random Number Generator*. En plus du surcoût inhérent à l'intégration de celui-ci à la plateforme, chaque appel à de tels générateurs peut coûter plusieurs dizaines de cycles. Par exemple, lors de leurs expériences Journault et Standaert [23] utilisent un générateur qui peut fournir un mot de 32 bits tous les 80 cycles. Pour leur implémentation de l'AES masqué à l'ordre $d = 31$, ils rapportent qu'environ 90% du temps de calcul est passé dans la génération d'aléas.

### 4.2 Réduire le coût du masquage

Plusieurs pistes sont alors envisageables pour réduire le coût des implémentations masquées. La première est, considérant que les opérations les plus chères à masquer sont les ET logiques, de concevoir et d'utiliser des algorithmes cryptographiques dont la complexité en ces opérations est la plus faible possible. C'est notamment dans ce but que PICARO [31], `Zorro` [18], Fantomas/Robin [21], (i)SCREAM et plus récemment `Pyjamask` [20] ont été conçus. Néanmoins, cette approche n'est pas générique et l'effort mis en œuvre pour obtenir une fonction de chiffrement symétrique efficace ne peut pas forcément être réinvesti pour d'autres primitives cryptographiques. De plus, limiter radicalement la complexité multiplicative dans le but de réduire le nombre de ET logiques nécessaires fait émerger des

attaques exploitant la faible quantité d'éléments non-linéaires. Un exemple d'attaque contre iSCREAM, Robin et Zorro a été présenté par Leander, Minaud et Rønjom [28] à EUROCRYPT 2015.

Il est donc intéressant de chercher en parallèle à améliorer les performances du masquage en lui-même. Cela passe notamment par la conception de schémas de masquage de portes élémentaires moins gourmands en opérations et en masques de rafraîchissement, mais aussi par la recherche de modèles plus précis pour la sécurité de la composition de gadgets. Le développement d'outils d'optimisation mais aussi de vérification de la sécurité des gadgets et des compositions de gadgets facilite la conception de ceux-ci. À terme, ces efforts ont pour but de permettre de protéger génériquement et à des coûts abordables des implémentations entières.

## Remerciements

L'auteur remercie Pierre Karpman pour son aide, sa relecture rapide et attentive ainsi que pour ses commentaires éclairants.

## Références

1. Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. maskverif : Automated verification of higher-order masking in presence of physical defaults. In Kazue Sako, Steve A. Schneider, and Peter Y. A. Ryan, editors, *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part I*, volume 11735 of *Lecture Notes in Computer Science*, pages 300–318. Springer, 2019.
2. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129. ACM, 2016.
3. Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. *IACR Cryptol. ePrint Arch.*, 2016 :912, 2016.
4. Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 616–648. Springer, 2016.

5. Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Private multiplication over finite fields. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 397–426. Springer, 2017.
6. Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, and Raphaël Wintersdorff. Tornado : Automatic generation of probing-secure masked bitsliced implementations. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 311–341. Springer, 2020.
7. Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight private circuits : Achieving probing security with the least refreshing. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 343–372. Springer, 2018.
8. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference. `https://keccak.team/files/Keccak-reference-3.0.pdf`, 2011.
9. Nicolas Bordes and Pierre Karpman. Fast verification of masking schemes in characteristic two. In Anne Canteaut and Francois-Xavier Standaert, editors, *To appear in : Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings*.
10. Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 45–68. Springer, 2017.
11. Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15 :2542–2555, 2020.
12. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
13. Jean-Sébastien Coron and Ilya Kizhvatov. An efficient method for random delay generation in embedded software. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 2009.
14. Jean-Sébastien Coron and Ilya Kizhvatov. Analysis and improvement of the random delay countermeasure of CHES 2009. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES*

*2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2010.

15. Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models : From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440. Springer, 2014.
16. François Durvaux, Mathieu Renauld, François-Xavier Standaert, Loïc van Oldeneel tot Oldenzeel, and Nicolas Veyrat-Charvillon. Efficient removal of random delays from embedded software implementations using hidden markov models. In Stefan Mangard, editor, *Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers*, volume 7771 of *Lecture Notes in Computer Science*, pages 123–140. Springer, 2012.
17. Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 444–461. Springer, 2014.
18. Benoît Gérard, Vincent Grosso, María Naya-Plasencia, and François-Xavier Standaert. Block ciphers that are easier to mask : How far can we go ? In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, volume 8086 of *Lecture Notes in Computer Science*, pages 383–399. Springer, 2013.
19. Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
20. Dahmun Goudarzi, Jérémy Jean, Stefan Kölbl, Thomas Peyrin, Matthieu Rivain, Yu Sasaki, and Siang Meng Sim. Pyjamask : Block cipher and authenticated encryption with highly efficient masked implementation. *IACR Trans. Symmetric Cryptol.*, 2020(S1) :31–59, 2020.
21. Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. Ls-designs : Bitslice encryption for efficient masked software implementations. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 18–37. Springer, 2014.
22. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits : Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 463–481, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
23. Anthony Journault and François-Xavier Standaert. Very high order masking : Efficient implementation and security evaluation. In Wieland Fischer and Naofumi

Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 623–643. Springer, 2017.

24. Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on keccak. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3) :243–268, 2020.
25. David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - statistical independence and leakage verification. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816. Springer, 2020.
26. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
27. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
28. Gregor Leander, Brice Minaud, and Sondre Rønjom. A generic approach to invariant subspace attacks : Cryptanalysis of robin, iscream and zorro. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 254–283. Springer, 2015.
29. Victor Lomne and Thomas Roche. A Side Journey to Titan. `https://ninjalab.io/wp-content/uploads/2021/01/a_side_journey_to_titan.pdf`, 2021.
30. Thomas S. Messerges. Securing the aes finalists against power analysis attacks. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, and Bruce Schneier, editors, *Fast Software Encryption*, pages 150–164, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
31. Gilles Piret, Thomas Roche, and Claude Carlet. PICARO - A block cipher allowing efficient higher-order side-channel resistance. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, *Applied Cryptography and Network Security - 10th International Conference, ACNS 2012, Singapore, June 26-29, 2012. Proceedings*, volume 7341 of *Lecture Notes in Computer Science*, pages 311–328. Springer, 2012.

# A Description des sondes utilisée lors de la compilation de `binverif`

```
#include <stdint.h>

/* Probe description for /tmp/gadget */
```

```
char *filename = "/tmp/gadget";

uint64_t probes_r[16] =  { 0x9, 0x1, 0x1, 0x3, 0x2, 0x2, 0x4, 0x6, 0
    x4, 0x8, 0xc, 0x8, 0xc, 0x6, 0x3, 0x9 };

uint16_t probes_sh_a[16][16] =  { { 0x1, 0x0, 0x0, 0x9 }, /*s33 r3
    s30 s03 r0 */
{ 0x0, 0x0, 0x0, 0x1 }, /*s33 r3 */
{ 0x1, 0x0, 0x0, 0x9 }, /*s33 r3 s30 s03 */
{ 0x0, 0x0, 0x3, 0x2 }, /*s22 r2 s23 s32 r3 */
{ 0x0, 0x0, 0x3, 0x2 }, /*s22 r2 s23 s32 */
{ 0x0, 0x0, 0x2, 0x0 }, /*s22 r2 */
{ 0x0, 0x6, 0x4, 0x0 }, /*s11 r1 s12 s21 */
{ 0x0, 0x6, 0x4, 0x0 }, /*s11 r1 s12 s21 r2 */
{ 0x0, 0x4, 0x0, 0x0 }, /*s11 r1 */
{ 0xc, 0x8, 0x0, 0x0 }, /*s00 r0 s01 s10 */
{ 0xc, 0x8, 0x0, 0x0 }, /*s00 r0 s01 s10 r1 */
{ 0x8, 0x0, 0x0, 0x0 }, /*s00 r0 */
{ 0xe, 0x8, 0x0, 0x0 }, /*s00 r0 s01 s10 r1 s02 */
{ 0x0, 0x7, 0x4, 0x0 }, /*s11 r1 s12 s21 r2 s13 */
{ 0x0, 0x0, 0xb, 0x2 }, /*s22 r2 s23 s32 r3 s20 */
{ 0x1, 0x0, 0x0, 0xd }, /*s33 r3 s30 s03 r0 s31 */
};

uint16_t probes_sh_b[16][16] =  { { 0x1, 0x0, 0x0, 0x9 }, /*s33 r3
    s30 s03 r0 */
{ 0x0, 0x0, 0x0, 0x1 }, /*s33 r3 */
{ 0x1, 0x0, 0x0, 0x9 }, /*s33 r3 s30 s03 */
{ 0x0, 0x0, 0x3, 0x2 }, /*s22 r2 s23 s32 r3 */
{ 0x0, 0x0, 0x3, 0x2 }, /*s22 r2 s23 s32 */
{ 0x0, 0x0, 0x2, 0x0 }, /*s22 r2 */
{ 0x0, 0x6, 0x4, 0x0 }, /*s11 r1 s12 s21 */
{ 0x0, 0x6, 0x4, 0x0 }, /*s11 r1 s12 s21 r2 */
{ 0x0, 0x4, 0x0, 0x0 }, /*s11 r1 */
{ 0xc, 0x8, 0x0, 0x0 }, /*s00 r0 s01 s10 */
{ 0xc, 0x8, 0x0, 0x0 }, /*s00 r0 s01 s10 r1 */
{ 0x8, 0x0, 0x0, 0x0 }, /*s00 r0 */
{ 0xc, 0x8, 0x8, 0x0 }, /*s00 r0 s01 s10 r1 s02 */
{ 0x0, 0x6, 0x4, 0x4 }, /*s11 r1 s12 s21 r2 s13 */
{ 0x2, 0x0, 0x3, 0x2 }, /*s22 r2 s23 s32 r3 s20 */
{ 0x1, 0x1, 0x0, 0x9 }, /*s33 r3 s30 s03 r0 s31 */
};

uint8_t radices[16] =  { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1 };
```

```
#ifndef PROBES_DESC_H
#define PROBES_DESC_H

#define NB_SH 4
#define NB_PR 16
#define NB_R 4
#define D 3
#define NB_INT 12
#define VECT
/* Probe description for /tmp/gadget */
```

```
extern char *filename;
extern uint64_t probes_r[16];
extern uint16_t probes_sh_a[16][16];
extern uint16_t probes_sh_b[16][16];
extern uint8_t radices[16];

#endif /* PROBES_DESC_H */
```