VERISICC
Livrable 1.2
Définition des besoins

APPEL A PROJET: FUI25

ACRONYME DU PROJET: VERISICC

NOM DU PROJET: VERISICC

Responsable de ce livrable

- Partenaire : Idemia
- Nom du contact : Aurélien Greuet
- Adresse du contact : aurelien.greuet@idemia.com

Leader du projet

- Entité : CryptoExperts
- Nom du contact : Sonia Belaïd
- Coordonnées (mail + téléphone) : sonia.belaid@cryptoexperts.com, 06 68 75 30 66

Partenaires

- PMEs : CryptoExperts et NinjaLab
- Grande entreprise : IDEMIA
- Etablissements publics : INRIA, ANSSI et Université du Luxembourg

Version : juillet 2019

Sommaire

1. Contexte

Dans un monde de plus en plus connecté, aussi bien entre individus qu'entre objets, les problématiques de confidentialité, d'intégrité, d'identification et d'authentification sont critiques. La carte à puce et les éléments sécurisés des systèmes embarqués offrent des réponses à ces problématiques dans plusieurs domaines clés :

- Les transactions bancaires sont sécurisées grâce aux cartes de paiement ou, dans le cas du paiement mobile sans contact, grâce aux éléments sécurisés des téléphones. Les données de l'utilisateur sont protégées par le chiffrement. La validité de la transaction est quant à elle assurée grâce à une signature numérique ainsi que par l'authentification de la carte ou de l'élément sécurisé.
- Les communications téléphoniques mobiles sont protégées à plusieurs niveaux. Depuis le protocole 3G, une authentification mutuelle permet d'assurer à la fois la légitimité de l'utilisateur sur le réseau, mais aussi la légitimité de l'antenne à laquelle le téléphone se connecte. La confidentialité des communications est assurée grâce au chiffrement entre le téléphone et l'antenne. En plus des données de communication, des codes d'authentification sont transmis entre antenne et téléphone, assurant ainsi l'intégrité des échanges.
- Dans le domaine de l'identité, la biométrie est de plus en plus utilisée, aussi bien pour la sécurité qu'elle apporte que pour sa simplicité d'usage pour l'utilisateur final.

  Les données biométriques étant des données à caractère personnel et pour la plupart uniques et permanentes, une attention particulière doit être apportée pour leur protection.

  Dans le cas de passeports ou de cartes d'identité électroniques, les données biométriques sont stockées chiffrées. Pour vérifier l'identité du porteur, le passeport ou la carte ne peut transmettre les données biométriques qu'aux seuls terminaux légitimes (*e.g.* aéroports, gouvernements), ces terminaux prouvant leur légitimité par un protocole d'authentification et le canal de transmission étant chiffré.

La cryptographie est au cœur de la sécurité des exemples précédents. Les algorithmes cryptographiques utilisés dans les cartes à puces et les éléments sécurisés sont des primitives standardisées et sont sûres (ou présumées sûres) contre les attaques « en boîte noire » : un attaquant ayant accès à un nombre limité d'entrées et de sorties ne sera pas en mesure de retrouver le secret, clé cryptographique ou message en clair, en un temps raisonnable.

Néanmoins, s'ils sont implémentés sans précaution sur des composants, ces algorithmes peuvent être vulnérables aux attaques par canaux auxiliaires. Ces attaques extrêmement puissantes tirent parti du comportement physique du composant exécutant l'algorithme : le temps d'exécution, la consommation de courant ou encore les émanations électromagnétiques peuvent apporter à l'attaquant de l'information sur les secrets manipulés, comme des clés de chiffrement ou les messages en clair.

En pratique, les développeurs de produits embarquant de la cryptographie ajoutent et évaluent les contremesures aux attaques physiques manuellement. Concrètement, ils mettent en place des mesures de protection dans leurs implémentations qui sont ensuite relues en interne et évaluées en pratique sur banc d'attaque.

Pour garantir aux utilisateurs un bon niveau de sécurité, les produits embarquant de la cryptographie sont ensuite généralement évalués par les Centres d'Évaluation de la Sécurité des Technologies de l'Information (CESTI), afin d'obtenir une certification de l'Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI). Les évaluations d'implémentations par les CESTI passent par de la relecture de code pour détecter d'éventuelles vulnérabilités, mais aussi par la mise en place d'attaques physiques.

Dans les deux cas, ce processus d'évaluation est peu automatisé. Il est donc coûteux en terme de temps et peut potentiellement souffrir d'une erreur humaine. En effet, on privilégie en général l'aspect sécuritaire en pratique : on vérifie que l'implémentation est résistante aux attaques physiques de l'état de l'art, mais les contraintes de l'industrie ne permettent pas d'allouer du temps pour la production de preuves formelles de sécurité.

2. Cryptographie en environnement embarqué

2.1. Spécificités

2.1.1. Architecture et contraintes de performances

En environnement embarqué sécurisé, les calculs cryptographiques sont effectués par des composants dont l'architecture est composée :

- de mémoire flash non-volatile, dans laquelle sont stockés les données (clés cryptographiques, biométrie) et le code à exécuter,
- de mémoire vive, dans laquelle sont stockées les informations manipulées par le microprocesseur et les éventuels coprocesseurs,
- d'un microprocesseur à usage général, 16 ou 32 bits, capable de lire et écrire en mémoire vive et d'effectuer des opérations logiques et arithmétiques sur des mots machines,
- d'un générateur de nombres aléatoires,
- éventuellement d'unités de calcul destinées à un usage très spécifique, par exemple du contrôle de redondance cyclique (CRC), du chiffrement symétrique ou encore du calcul modulaire.

Ces composants, aussi bien pour un usage dans l'Internet des objets (IoT) que pour les cartes à puces, sont soumis à de nombreuses contraintes. Ils doivent généralement être petits et consommer peu d'énergie pour fonctionner. De plus, pour les productions à grande échelle, les fabricants doivent minimiser au maximum les coûts pour être compétitifs par rapport à leurs concurrents.

De par ces contraintes, les puces embarquées sont souvent, comparées à un ordinateur ou un smartphone, très peu puissantes, contiennent peu de mémoire vive et peu de mémoire non-volatile. À titre d'exemple, les composants les plus hauts de gamme utilisés actuellement dans les cartes à puces destinées à un usage sensible comme les cartes bancaires, passeports ou encore cartes SIM, disposent généralement :

- de quelques dizaines de kilooctets de mémoire vive,
- moins de 2 mégaoctets de mémoire non-volatile,
- d'un processeur simple cœur cadencé à moins de 100MHz.

Un smartphone bas de gamme de moins de 100€ en 2019 va contenir

- au moins 1Go de mémoire vive (20000 fois plus),
- au moins 8Go de mémoire non-volatile (4000 fois plus),
- un processeur de 4 cœurs, chacun des cœurs étant cadencé à au moins 1GHz (10 fois plus rapide).

Malgré la puissance limitée des composants embarqués sécurisés, les contraintes de performances imposées par les clients des fabricants de cartes ou de systèmes embarqués peuvent être très fortes. Par exemple, une transaction bancaire sans contact doit pouvoir être effectuée en moins de 300ms. De plus, il est souvent préférable de minimiser la taille de code afin de laisser plus de mémoire non-volatile pour le stockage de données, par exemple biométriques, ou pour laisser au client la possibilité d'ajouter ses propres applications. Les algorithmes cryptographiques sont donc souvent développés en langage bas niveau comme le C, voire en assembleur pour les parties les plus critiques en terme de performances.

2.1.2. Vulnérabilités

Les systèmes embarqués sont sujets à des vulnérabilités spécifiques. Ils sont en effet généralement très faciles d'accès : les systèmes embarqués pour l'Internet des objets peuvent être achetés par un attaquant pour ensuite être analysés comme bon lui semble, sans aucun contrôle possible par le fabricant. Il lui est donc possible d'analyser les communications, d'étudier le système en boîte noire, mais aussi de le démonter et l'altérer. De même, il suffit d'ouvrir un compte dans une banque pour disposer d'une carte bancaire ou de s'abonner chez un opérateur téléphonique pour obtenir une carte SIM, qu'un attaquant pourra analyser, avec des méthodes invasives qui peuvent détériorer le composant, sans que le fabricant ne puisse le détecter.

De par cette facilité d'accès, la recherche de vulnérabilités est aisée et les attaques par canaux auxiliaires sont souvent privilégiées. En effet, ces attaques sont extrêmement puissantes et, dans le cas des systèmes embarquées, assez faciles à mettre en place. Puisqu'un attaquant peut disposer d'un accès physique à la cible sans que le fabricant ne puisse le détecter, il peut utiliser du matériel conséquent (oscilloscope, sondes électromagnétiques, lecteur de carte à puce modifié pour l'acquisition de traces de consommation, etc.) et n'est pas limité dans le temps pour mener ses attaques.

De plus, la faible puissance des composants dans les systèmes embarqués en fait une cible de choix pour les attaques par canaux auxiliaires : contrairement à un ordinateur ou un smartphone, sur lesquels plusieurs dizaines d'applications sont actives en parallèle, un système embarqué est très souvent monotâche. L'exécution d'algorithmes cryptographiques étant donc généralement isolée, l'information pouvant fuir d'une telle exécution n'est pas noyée dans le bruit généré par d'autres applications s'exécutant en parallèle.

Enfin, les produits embarqués, et particulièrement les cartes à puces, ont une durée de vie relativement longue alors qu'une mise à jour n'est pas toujours possible. Par exemple, une carte bancaire est valable deux ou trois ans et un passeport dix ans. Il est très fréquent que le code cryptographique embarqué dans ces types de puces ne soit pas modifiable. En cas de découverte d'une vulnérabilité critique, il peut alors être nécessaire de rappeler les produits concernés. Quand bien même une mise à jour serait possible, elle pourrait nécessiter une action active de la part de l'utilisateur, contrairement à une mise à jour logicielle automatique sur un ordinateur ou un smartphone. Il est donc probable que la mise à jour ne soit pas rapidement ni correctement déployée.

Il est donc nécessaire, aussi bien pour les développeurs que pour les évaluateurs des CESTIs, d'anticiper au maximum les évolutions possibles des attaques à moyen terme, afin de proposer des contre-mesures efficaces sur la durée. Les développeurs doivent donc avoir un contrôle très fin sur l'exécution du code et sur la mise en place des contre-mesures pour les parties critiques, au niveau sécuritaire, des algorithmes cryptographiques. Pour cela, les parties sensibles sont écrites en code assembleur. En effet, le passage par un langage de plus haut niveau impliquerait la compilation du code, qui pourrait altérer les contre-mesures, par exemple par des optimisations qui ne perturbent pas le comportement fonctionnel de l'algorithme.

2.2. Algorithmes cryptographiques

2.2.1. Panorama des algorithmes embarqués dans les produits sécurisés

Afin de répondre aux problématiques d'authentification, de confidentialité et d'intégrité des données sensibles, les systèmes embarqués sécurisés disposent d'un large éventail d'algorithmes cryptographiques.

Les produits embarquent généralement des algorithmes dont la conception et le fonctionnement sont publics, standardisés et recommandés par des organismes tels que l'Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI) en France ou le National Institute of Standards and Technology (NIST) aux États-Unis. Par exemple, les algorithmes suivants, standardisés et satisfaisant les recommandations de l'ANSSI et du NIST, se retrouvent dans une large part des produits sécurisés :

- l'AES, qui permet de chiffrer des données et d'authentifier et vérifier leur intégrité,
- le TDES, ancien standard américain remplacé par l'AES, qui a les mêmes fonctionnalités que ce dernier. Il reste présent pour des raisons de rétrocompatibilité.
- le RSA, algorithme de chiffrement asymétrique, utilisé pour le calcul de signatures numériques,

- l'ECDH, échange de clé Diffie-Hellman basé sur les courbes elliptiques,
- l'ECDSA, algorithme de signature utilisant les courbes elliptiques,
- les fonctions de hachage SHA-2 et SHA-3, utilisées pour vérifier l'intégrité des échanges et dans la construction de signatures numériques,
- les algorithmes de génération d'aléa et de clés cryptographiques.

D'autres algorithmes moins classiques peuvent aussi être présents en fonction des demandes des clients. Par exemple, le standard de chiffrement symétrique coréen SEED est souvent demandé sur les puces destinées à la Corée du Sud. De même, le standard russe GOST est souvent présent dans les produits utilisés en Russie. Ces deux algorithmes ont globalement la même structure que les standards plus classiques. Cependant, certaines briques de base sont assez différentes, avec par exemple de l'arithmétique modulaire dans les boîtes S ou une description des boîtes S uniquement donnée par une table. Ces différences peuvent induire des méthodes d'implémentation et des contre-mesures spécifiques.

En plus de ces algorithmes, les standards coréens, russes et chinois définissent des algorithmes de signature numérique et des fonctions de hachage, mais ils sont très similaires aux standards plus classiques, même au niveau des briques de base.

2.2.2. Accélérateurs matériels

Comme indiqué en Section 2.1.1, les périphériques embarqués sont assez limités en terme de puissance de calcul et de quantité de mémoire alors même qu'ils doivent réaliser des opérations cryptographiques complexes. Pour concilier ces deux problématiques, les fondeurs proposent des puces qui incluent des accélérateurs matériels. Ce sont des coprocesseurs destinés à exécuter une tâche très spécifique et qui peuvent généralement réaliser leurs opérations en parallèle du microprocesseur.

Les accélérateurs matériels pour la cryptographie symétrique permettent d'effectuer le chiffrement complet d'un bloc de message en quelques centaines de cycles, là où une implémentation logicielle équivalente prendrait au moins 50 fois plus de temps. Le développement est aussi bien plus rapide avec un accélérateur matériel : il suffit généralement de copier la clé et le bloc de message dans le coprocesseur, d'attendre le signal indiquant que le chiffrement est terminé, puis de récupérer le résultat du coprocesseur vers la mémoire principale.

En revanche, contrairement à une implémentation logicielle, on a très peu de contrôle sur le niveau de sécurité et puisque le chiffrement complet d'un bloc est effectué, il n'est pas possible de se protéger contre les attaques par canaux auxiliaires avec du masquage. Souvent, les coprocesseurs pour la cryptographie symétrique proposent plusieurs niveaux de sécurité, son augmentation altérant légèrement les performances. Mais il n'est en général pas possible d'obtenir des détails sur les contre-mesures matérielles choisies par les fondeurs, et les développeurs doivent souvent ajouter des contre-mesures logicielles.

Les accélérateurs matériels pour la cryptographie asymétrique permettent quant à eux d'effectuer rapidement certaines opérations de base sur des grands nombres. Comme pour les coprocesseurs symétriques, le temps d'exécution et le temps de développement sont bien plus courts qu'une implémentation logicielle. Les fonctionnalités dépendent des composants, mais parmi les opérations disponibles, on peut citer :

- Addition, soustraction et multiplication d'entiers et/ou d'entiers modulaires, le module pouvant généralement être n'importe quel entier impair
- Addition et multiplication sur un corps de caractéristique 2
- Opérations logiques : ou exclusif (xor), décalages à gauche et à droite
- Multiplication et division modulaires par 2
- Réduction modulaire

Ces calculs s'effectuent, suivant les composants, sur des nombres dont la taille peut aller jusqu'à 2048 ou 4096 bits. On peut donc utiliser les opérations précédentes comme briques de base pour les algorithmes basés sur des corps finis ou des anneaux modulaires comme RSA ou les algorithmes basés sur les courbes elliptiques.

Du point de vue de la sécurité, les accélérateurs matériels font généralement leurs calculs en temps constant : le temps d'exécution d'une opération sera identique pour une taille d'opérandes donnée, quelles que soient les valeurs des opérandes. Ainsi, le temps de calcul ne dépendra pas des valeurs manipulées, ce qui autorise la manipulation de secrets.

2.3. Attaques par canaux auxiliaires

Les attaques par canaux auxiliaires exploitent le fait que le temps d'exécution, la température, la consommation de courant ou encore les émanations électromagnétiques d'un composant dépendent des données manipulées. En particulier, lorsqu'un secret est manipulé, de l'information liée à ce secret peut être révélée.

2.3.1. Attaques simples de type SPA (Simple Power Analysis)

Description. Une attaque simple est une attaque qui permet de retrouver le secret ou une partie du secret avec une seule trace d'exécution (consommation de courant ou émanations électromagnétiques par exemple). Elles sont particulièrement efficaces lorsque les calculs effectués dépendent directement du secret.

Pour l'illustrer, on considère la multiplication scalaire d'un point sur une courbe elliptique, où le scalaire est secret, opération à la base des algorithmes basés sur les courbes elliptiques. Pour l'implémenter efficacement, on utilise un algorithme dit binaire : on parcourt les bits du scalaire et les opérations sur les points de la courbe dépendent de la valeur du bit courant. Si le bit est à 0, on fait un doublement du point courant alors que s'il vaut 1, on fait un doublement et une addition de points. Les formules de doublement et d'addition sont des enchaînements d'opérations arithmétiques sur le corps de base. Pour les plus efficaces, les formules d'addition et de doublement utilisent des opérations différentes. Ces différences peuvent être facilement observées sur une trace d'exécution. Un attaquant peut donc facilement distinguer un doublement et un doublement suivi d'une addition. Comme cette distinction dépend directement de la valeur du secret, il est alors en mesure de retrouver les bits du scalaire, et donc le secret.

Contremesures. Pour se protéger d'une telle attaque, il faut rendre l'algorithme régulier : les opérations ne doivent pas dépendre de la valeur du secret. On peut par exemple effectuer un doublement suivi d'une multiplication, quelle que soit la valeur du bit courant de scalaire. Dans le cas où le bit vaut 0, le résultat de la multiplication ne sera pas conservé.

Une autre solution est d'utiliser des formules de doublement et d'addition qui génèrent la même trace d'exécution. Il faut alors que les suites d'opérations sur le corps de base soient les mêmes. On parle alors de formules atomiques [CCJ04, Lon07, GV10, Ron14].

2.3.2. Attaques par corrélation de type CPA (Correlation Power Analysis)

Description. Les attaques par corrélation [BCO04], extensions des attaques différentielles introduites dans [KJJ99], nécessitent plusieurs traces d'exécution. Un traitement statistique sur ces traces d'exécution permettra d'obtenir de l'information sur les secrets. On présente ici le principe général d'une attaque par corrélation.

Le but pour l'attaquant est d'abord de modéliser la fuite du composant. Si cette fuite dépend de la donnée manipulée x, on observera une consommation ou une émanation électromagnétique de la forme $f(x)+\mathcal{B}$, où $\mathcal{B}$ est du bruit aléatoire indépendant de x. Il s'agit alors d'obtenir une estimation de f afin de prédire la fuite, ou du moins être capable de calculer des valeurs fortement corrélées à cette fuite. Par exemple, si une puce génère des fuites dépendant du poids de Hamming de la donnée manipulée, la fonction de fuite f sera de la forme $f(x) = a \cdot HW(x)$, où a est une constante non nulle et HW la fonction poids de Hamming (*i.e.* $HW(x)$ est égale au nombre de bits non nuls de x). L'attaquant peut alors modéliser la fuite avec la fonction HW. La fonction choisie pour la modélisation s'appelle la *fonction de prédiction*. À cause de la constante a et du bruit généré, cette modélisation ne sera jamais parfaitement égale à la consommation observée. Néanmoins, les deux valeurs seront fortement corrélées.

Grâce à une telle modélisation, l'attaquant peut être en mesure de déduire de l'information sur les secrets manipulés. Il cible un instant où, durant l'exécution de l'algorithme, la « combinaison » d'une petite valeur $\hat{k}$ dépendant du secret et d'une petite valeur m, connue et pouvant varier, est manipulée. Cette combinaison est généralement une opération booléenne comme le « ou exclusif » ou une opération arithmétique, comme une multiplication.

Ensuite, pour toutes les valeurs possibles de $\hat{k}$ et pour des messages m_i, $1 \leq i \leq n$, il évalue la fonction de prédiction en chaque combinaison de $\hat{k}$ et de m_i. Il dispose donc, pour chaque valeur possible de $\hat{k}$, d'un vecteur $p_{\hat{k}}$ de n coordonnées, qui contient des valeurs qui doivent être corrélées à la consommation observée à l'exécution de l'algorithme avec le secret $\hat{k}$ et les valeurs m_i.

Il mesure alors la consommation réelle du composant à l'instant ciblé, pour tous les messages choisis précédemment. Il obtient un vecteur c_k de n coordonnées, correspondant aux consommations observées avec la véritable valeur secrète k et les valeurs m_i.

Il lui suffit ensuite de calculer, pour chaque valeur possible de $\hat{k}$, un coefficient de corrélation pour le couple de vecteurs $(c_k, p_{\hat{k}})$. Le $\hat{k}$ pour lequel le coefficient de corrélation est le plus élevé a alors de bonne chances d'être égal au vrai secret k. En effet, si le modèle de fuite choisi correspond bien au comportement de la puce, le vecteur p_k correspondant au secret sera formé de valeurs toutes fortement corrélées à celles du vecteur c_k. À l'inverse, pour $\hat{k} \neq k$, la combinaison de $\hat{k}$ et du message m_i a de bonnes chances d'être différente de la combinaison de k et du même message m_i. Ainsi, il est très probable que la fonction de prédiction donne un résultat qui n'est pas corrélé à la consommation observée. Comme c'est le cas pour chaque m_i, le coefficient de corrélation $(c_k, p_{\hat{k}})$ sera proche de 0.

Pour que les étapes de construction des vecteurs $p_{\hat{k}}$ et des calculs de coefficients de corrélation correspondants soient praticables, l'attaquant cible des instants où la partie de secret manipulée est de l'ordre d'un octet.

Contremesures. Le succès d'une attaque différentielle dépend de la corrélation entre la consommation de courant ou les émanations électromagnétiques et la valeur des variables sensibles manipulées. L'utilisation d'aléa permet de supprimer cette corrélation [GP99, CJRR99]. Par exemple, dans le cas d'un algorithme utilisant des opérations booléennes comme le TDES ou l'AES, au lieu de manipuler directement une variable sensible x, on manipule indépendamment $x \oplus m$ et m, où m est un masque aléatoire de la même taille que x. L'algorithme doit donc être modifié pour que le masque se propage correctement.

Lorsqu'un chiffrement symétrique est effectué par un accélérateur, son implémentation ne peut pas être modifiée et on ne peut donc pas se protéger avec du masquage. Pour éviter qu'une attaque puisse être menée, on peut augmenter le bruit. En théorie, l'attaque sera toujours possible, mais en pratique, l'attaquant aurait besoin d'un nombre non praticable de traces pour éliminer le bruit et retrouver l'information.

La principale contre-mesure utilisée avec un coprocesseur symétrique est le « 1 parmi N » : on noie le « vrai » chiffrement d'un bloc de message dans $N-1$ « faux » chiffrements, effectués avec des messages aléatoires. La position du vrai chiffrement parmi les faux est choisie aléatoirement, de sorte qu'un attaquant ne pourra pas la détecter. Le choix du N dépend du composant et de sa manière de fuir l'information.

Attaques d'ordre supérieur. L'exemple de CPA précédent est une attaque dite « d'ordre 1 ». Sur chacune des traces d'exécution, un seul instant est ciblé. L'utilisation d'un masque aléatoire permet donc de décorréler la consommation du secret. On parle de masquage d'ordre 1.

Dans les attaques d'ordre supérieur, on cible plusieurs instants à la fois. Avec une attaque d'ordre 2, on peut casser une implémentation masquée à l'ordre 1 : on cible à la fois un instant où le secret masqué est manipulé et un instant où on manipule le masque. La combinaison statistique de ces deux informations permet de révéler le secret démasqué [Mes00, PRB09].

Contremesures d'ordre supérieur. De la même manière qu'on se protège d'attaques d'ordre 1 en masquant les données sensibles pour décorréler la consommation, on partage le secret en $n+1$ parties pour se protéger d'attaques d'ordre n. Étant donnée une variable sensible x, on manipulera indépendamment $x_0, x_1, \ldots, x_n$, où les x_i sont uniformément distribués, indépendants et tels que $x = x_0 \oplus \cdots \oplus x_n$, plutôt que x.

Pour fonctionner avec un tel partage, l'algorithme doit être modifié en conséquence. De plus, pour qu'un partage en $n+1$ parties protège contre une attaque d'ordre n, certaines précautions supplémentaires doivent être prises pour assurer la sécurité globale de l'algorithme [CPR07, CPRR14].

Comme pour l'ordre 1, une telle contre-mesure ne peut pas être mise en place avec un coprocesseur symétrique. Il faut alors estimer la complexité de l'attaque et adapter le choix du N dans la contre-mesure « 1 parmi N » en fonction de la manière de fuir du composant.

2.3.3. Attaques horizontales

Les attaques CPA du paragraphe précédent exploitent plusieurs mesures de consommation, effectuées à un même instant sur plusieurs exécutions. Les attaques horizontales vont tirer parti de plusieurs mesures de consommation d'une seule exécution à des instants différents. Elles permettent d'attaquer des implémentations protégées contre les attaques CPA en cryptographie symétrique [BCPZ16] et des implémentations de cryptographie asymétrique protégées contre les attaques SPA [Wal01, CFG$^+$10, BJPW13, BJP$^+$15].

Description. Pour illustrer le fonctionnement de ce type d'attaques, considérons une implémentation atomique de la multiplication scalaire sur une courbe elliptique [CCJ04, Lon07, GV10, Ron14]. Une telle multiplication scalaire est au cœur des algorithmes basés sur les courbes elliptiques, aussi bien pour l'échange de clé Diffie-Hellman que pour les signatures de type ECDSA.

Comme indiqué au paragraphe 2.3.1, le calcul d'une multiplication scalaire est formé d'un enchaînement d'additions et de doublements de points de la courbe elliptique. L'ordre de ces additions et doublements étant directement dépendant du secret, il est nécessaire qu'un attaquant ne puisse pas différencier les deux opérations. L'utilisation de formules atomiques répond à ce problème : l'addition et le doublement sont calculés en utilisant les mêmes blocs d'opérations sur le corps de base, seuls les opérandes sont différents. Ainsi, même si l'attaquant est capable d'identifier les différentes opérations arithmétiques sur le corps de base, il n'en observera que des blocs identiques et ne sera donc pas en mesure de différencier les additions et doublements de points.

De plus, parmi les contre-mesures aux attaques SPA sur la multiplication scalaire, l'utilisation de formules atomiques est généralement la plus efficace en terme de performances.

Les attaques horizontales sur des implémentations atomiques vont tirer parti du fait qu'il y a, avec la plupart des accélérateurs matériels, une corrélation entre la consommation de deux opérations ayant un opérande en commun. Par exemple, dans les deux formules suivantes, les opérations arithmétiques sont identiques :

1. • $A \times B$
   • $C \times D$
2. • $A \times B$
   • $C \times B$

Néanmoins, dans la seconde, le deuxième opérande B est commun aux deux multiplications. Il risque alors d'y avoir une corrélation entre la consommation mesurée à l'instant où $A \times B$ est calculé et la consommation au moment du calcul de $C \times B$. En revanche, si $B \neq D$, il n'y aura pas corrélation entre les consommations des calculs de $A \times B$ et de $C \times D$ de la première formule.

L'attaquant peut alors identifier les deux formules s'il dispose d'une trace de consommation : il mesure les corrélations entre les consommations de deux multiplications successives. Si la corrélation est faible, il s'agit de la première formule alors que si elle est forte, il s'agit de la seconde.

Contremesures. Lorsque ce type d'attaque cible une implémentation reposant sur un accélérateur matériel, la consommation permettant de détecter une corrélation est générée par l'accélérateur. Il n'est alors pas possible de modifier la manière de calculer pour y introduire de l'aléa comme on peut le faire contre les attaques CPA. On peut alors, lorsque c'est possible, modifier les formules atomiques pour qu'il n'y ait plus d'opérations arithmétiques avec opérandes communs dans seulement une des formules.

Lorsque ce n'est pas possible ou trop coûteux, on peut remplacer les opérandes communs par des valeurs distinctes, pour éviter la corrélation, mais qui ne modifient pas le résultat final.

Par exemple, si le corps de base est de caractéristique p, les opérations arithmétiques se font modulo p. Il est alors possible de travailler modulo $k \cdot p$, où k est un petit nombre. On peut alors remplacer les formules du paragraphe précédent par les suivantes :

1. • $A \times B$
   • $C \times (D + p)$
2. • $A \times B$
   • $C \times (B + p)$

Les valeurs B et $B + p$ étant distinctes modulo $k \cdot p$, il n'est alors plus possible de différentier les deux formules en mesurant les corrélations. Les résultats intermédiaires seront différents de la version non protégée, mais une réduction modulo p à la toute fin des calculs éliminera les différents ajouts de p et permettra de retrouver le résultat original.

3. Identification des besoins

Comme indiqué dans les sections précédentes, la sécurisation contre les attaques par canaux auxiliaires comme le masquage nécessite de prendre certaines précautions, particulièrement pour les ordres supérieurs à 2. La vérification manuelle et les preuves de sécurité sont fastidieuses et sujettes à des erreurs. On peut citer le schéma de masquage de [SP06], cassé dans [CPR07], ou encore [RP10], dans lequel une étape était manquante pour assurer la sécurité globale, comme indiqué dans [CPRR14].

Pour éviter des études de sécurité complexes et laborieuses et ainsi limiter le risque d'erreur, des outils automatiques ont été développés [BRNI13, EWS14, EWTS14]. La *vérification formelle du masquage* a été introduite dans [BBD+15] puis améliorée dans [BBD+16] et plus récemment dans [BBC+19] pour fonctionner avec des ordres de masquage plus grands. Plus récemment, [Cor17] a proposé une approche différente permettant d'obtenir sensiblement les mêmes résultats.

Une autre approche pour faciliter le développement d'algorithmes sécurisés est de générer automatiquement les contre-mesures à partir d'un code ayant le comportement fonctionnel attendu. Cette approche a été proposée dans [EW14, BBD+16, WS17].

Les outils actuels, aussi bien pour la vérification formelle que pour la génération de code sécurisé, ne répondent pas complètement aux besoins des développeurs d'algorithmes cryptographiques ni aux évaluateurs des CESTI. Néanmoins, ils constituent certainement la base d'un cœur commun à partir duquel des outils spécifiques répondant à ces besoins peuvent être développés. Ces besoins spécifiques ainsi que leurs motivations sont décrits dans les paragraphes suivants.

3.1. Besoins des développeurs

Les outils de vérification formelle et de génération de code sécurisé pourraient aider à la conception d'algorithmes cryptographiques sécurisés. Pour qu'ils soient utilisés en pratique dans l'industrie, ils doivent répondre à un certain nombre de besoins.

3.1.1. Vérification formelle

Format des implémentations. Comme expliqué précédemment aux paragraphes 2.1.1 et 2.1.2, les parties les plus sensibles des algorithmes cryptographiques, du point de vue de la sécurité et des performances, sont développées en langage assembleur. Pour qu'ils soient utilisés en pratique par les développeurs, il est donc essentiel que les outils de vérification formelle puissent travailler directement à partir de cette représentation. En effet jusqu'à présent, les vérifieurs à l'état de l'art acceptent des implémentations en C, en CommonLisp, en Verilog, mais ne se sont pas encore adaptés aux langages de développement software bas niveau.

Alors que la majorité des composants sont des architectures ARM Cortex-M ou 80251, les outils de vérification formelle pourrait dans un premier temps être étendu à l'un de ces deux langages. Une architecture modulable pourrait être identifiée afin de permettre ensuite l'extension des outils de vérification formelle à d'autres architectures, par exemple avec une description des instructions disponibles.

Prise en compte de toutes les fonctions concrètement utilisées. On explique au paragraphe 2.2.1 que certains produits embarquent des standards asiatiques ou russes, qui utilisent des opérations assez différentes de celles utilisées pour l'AES. La boite S du SEED utilise par exemple une addition modulaire. Dans le cas d'un masquage booléen, il est nécessaire d'utiliser un algorithme spécifique pour effectuer cette addition, ou de procéder à une conversion vers un masque arithmétique avant l'addition, puis reconvertir le masque du résultat vers un masque booléen. Pour l'algorithme GOST par contre, la boite S est donnée sous forme de table. Ces deux exemples, bien que moins classiques que les briques de base utilisées dans l'AES, doivent pouvoir être gérés par les outils de vérification formelle.

Les besoins de vérification ne sont pas contraints à la cryptographie symétrique. En ce qui concerne la cryptographie asymétrique, nous avons vu au paragraphe 2.3.3 un exemple d'attaque horizontale. Pour s'en protéger avec des formules atomiques d'addition et de doublement, il faut assurer que des corrélations sur les opérandes en commun n'aient pas lieu dans une seule des deux formules. Les meilleures formules comportent au moins une vingtaine d'opérations sur le corps de base. Une telle vérification serait donc fastidieuse en pratique.

En revanche, ce type de tâche peut être automatisable et les mécanismes permettant de le faire ont certainement des similarités avec les méthodes de vérification formelle. Comme pour la vérification formelle, on peut imaginer une méthode modulable, dans laquelle on pourrait introduire des règles particulières suivant les caractéristiques du composant.

Identification précise des vulnérabilités. Lorsque l'outil de vérification formelle trouve des vulnérabilités, il pourrait indiquer précisément où se trouve la vulnérabilité et détailler le problème, voire proposer une solution. De cette manière, l'ajout de contre-mesures serait accéléré pour le développeur.

Prise en compte de la caractérisation. Il arrive qu'en pratique, une implémentation qui soit prouvée sûre dans un modèle de fuite classique présente des vulnérabilités dues à des fuites particulières du composant sur lequel elle est exécutée. Par exemple, il est possible que certains registres fuient beaucoup plus d'informations sur les variables manipulées que d'autres. Dans ce cas, il est certainement plus sûr de ne pas les utiliser pour manipuler des variables sensibles.

Aussi, dans le contexte du masquage au premier ordre, même en veillant soigneusement à ne pas manipuler simultanément une valeur masquée et son masque, une fuite dépendant de la valeur démasquée peut être observée. Ce cas n'est pas très fréquent mais peut se produire lorsque la valeur masquée et son masque sont stockés dans deux registres différents.

Ces particularités du composant sont essentielles pour le développeur car il doit adapter son implémentation à ces contraintes spécifiques de sécurité. Les outils de vérification et de génération de contre-mesures pourraient l'aider dans cette tâche en acceptant en entrée un ensemble de caractéristiques du composant ou un ensemble de règles pour faciliter le développement. Pour les deux exemples précédents, on pourrait alors indiquer au logiciel de vérification formelle de signaler les utilisations d'un registre donné pour manipuler des valeurs sensibles ou de lever une alerte lorsqu'une valeur masquée et le masque se trouvent dans des registres fuyant simultanément.

Quantification de la fuite. Nous avons expliqué au paragraphe 2.3.2 que pour mener la plupart des attaques par corrélation, nous ciblons la combinaison d'une petite constante dépendant du secret et d'une petite variable connue. Dans le cas d'un algorithme de cryptographie symétrique, la variable connue est généralement une partie du message clair, ce qui permet d'attaquer les premiers tours, ou, lorsqu'on attaque les derniers tours, une valeur qu'on retrouve à partir du message chiffré. Avec ce type d'attaque CPA, il est difficile d'attaquer les tours intermédiaires. En effet, après 2 ou 3 tours, la valeur connue est mélangée avec plusieurs bits de la clé et devient donc difficilement prédictible.

Il est donc fréquent de supposer que l'attaquant ne peut attaquer facilement que les 2 ou 3 premiers ou 2 ou 3 derniers tours. Ainsi, pour gagner en performance, on peut dégrader le niveau de sécurité pour les rounds intermédiaires. Par exemple pour une implémentation masquée à l'ordre 1, on peut ne protéger que les 3 premiers et 3 derniers tours et démasquer les autres. Une implémentation ainsi obtenue ne pourra pas obtenir une preuve formelle de sécurité sur son ensemble alors qu'en pratique, elle sera très probablement aussi sûre que la même version complètement masquée.

Néanmoins, on pourrait imaginer que la dépendance des données avec les bits de clés soit quantifiée et prise en compte dans des outils de vérification. Ainsi, on pourrait obtenir une garantie spécifique, qui assurerait un niveau de sécurité *en pratique* contre un type d'attaques donné.

Modèle de sécurité spécifique au composant Nous avons vu au paragraphe 2.2.2 que pour les algorithmes de cryptographie symétriques classiques comme le TDES ou l'AES, on dispose dans la majorité des cas d'accélérateurs matériels. Au paragraphe 2.3.2, on a donné l'exemple de la contre-mesure du 1 parmi N : N chiffrements sont effectués, parmi lesquels $N-1$ avec des messages et clés aléatoires et un seul avec le message et la clé corrects. La position du chiffrement correct est aléatoire, de sorte que l'attaquant ne sache pas sur quel calcul parmi les N mener son attaque.

Le choix de la valeur de N dépend du composant et de sa manière de fuir l'information. En pratique, on choisit ce N de sorte que le nombre de traces nécessaires pour retrouver le secret ne soit pas praticable. Le nombre de traces nécessaire est estimé en fonction de la caractérisation du composant, mais la procédure n'est pas automatique et l'estimation peut potentiellement être erronée à cause d'une erreur humaine ou d'un manque d'exhaustivité dans la caractérisation.

Le besoin utilisateur dans ce contexte serait de recevoir de la part des outils formels une valeur de N optimale permettant d'atteindre un niveau de sécurité cible en fonction des caractéristiques du composant.

3.1.2. Génération de code sécurisé

Pour qu'il soit utilisé dans l'industrie, un outil de génération de code sécurisé devra répondre aux différentes contraintes déjà discutées précédemment. En particulier, il est nécessaire que le code généré soit en langage assembleur. De plus, il serait certainement indispensable, pour que le code généré soit utilisé sans modification manuelle ultérieure par les développeurs, qu'il puisse tenir compte de vulnérabilités spécifiques comme indiqué au paragraphe 3.1.1. Dans ce cas, si la caractérisation du composant révèle des fuites particulières, on pourra adapter la génération de code pour s'en protéger.

Comme pour la vérification formelle au paragraphe 3.1.1, on s'attend à pouvoir dégrader le niveau de sécurité pour les tours intermédiaires d'un algorithme de chiffrement symétrique. En effet, les contraintes de performances étant fortes, il est très fréquent qu'on ne puisse pas se permettre de masquer l'algorithme dans son ensemble, sans que cela ne se traduise par un affaiblissement de la sécurité en pratique.

Toujours pour des raisons de performances, un outil de génération de code sécurisé pour l'embarqué devra être en mesure de générer du code efficace. Il est normal que du code généré par un outil automatique soit moins performant qu'un code équivalent optimisé par un développeur, mais la différence ne doit pas être trop importante.

3.2. Besoins des CESTI

Les besoins des CESTI sont assez similaires à une partie des besoins des développeurs. En effet, là où les développeurs vont chercher des vulnérabilités pour les corriger, les évaluateurs vont tenter de les exploiter et de mener des attaques.

3.2.1. Recherche de vulnérabilité pour code assembleur

Comme expliqué au paragraphe 3.1.1, les parties critiques des algorithmes cryptographiques sont développées en assembleur. Ainsi, il est primordial que les outils de recherche de vulnérabilité puissent travailler

directement sur du code assembleur afin d'aider l'évaluateur à cibler précisément les points d'intérêt pour les attaques.

Comme pour la vérification formelle, la recherche de vulnérabilité doit pouvoir gérer les algorithmes peu classiques comme les standards coréens et russes.

De même, les vulnérabilités spécifiques à un composant devraient être prises en compte. Pour reprendre l'exemple donné au paragraphe 3.1.1, il arrive que de l'information sur une variable sensible fuie alors même qu'elle est masquée. En effet, dans des cas très particuliers, un composant peut fuir de l'information si le masque se trouve dans un autre registre, même si la variable sensible masquée et le masque ne sont pas manipulés simultanément. Si un tel comportement est détecté lors de la caractérisation du composant, il serait préférable que l'outil de recherche de vulnérabilité puisse le prendre en compte et ainsi détecter de potentielles attaques en pratique, même si le schéma est protégé dans un modèle théorique générique.

Un tel outil pourra partager un cœur commun avec un outil de vérification formelle comme décrit au paragraphe 3.1.1.

3.2.2. Recherche de vulnérabilités pour l'asymétrique

Nous avons vu au paragraphe 2.3.3 que l'utilisation de formules atomiques d'addition et de doublement était la solution privilégiée pour se protéger contre les attaques SPA pour les algorithmes basés sur les courbes elliptiques. En revanche, des corrélations sur les opérandes qui seraient en commun dans une seule des deux formules permettent de les distinguer et donc de mener une attaque horizontale. Ces formules comportant au moins une vingtaine d'opérations, la recherche de vulnérabilités à ce niveau est très laborieuse. De plus, aussi bien avec une description en pseudo-code qu'avec une implémentation en langage bas niveau, la description peut se faire généralement en terme de registres ou de variables. Ainsi on peut avoir deux variables différentes qui contiennent en fait la même valeur, ce qui complexifie la recherche de corrélations possibles.

Il a été montré au paragraphe 3.1.1 qu'une adaptation des outils de vérification formelle pour détecter des problèmes de corrélation pourrait permettre d'automatiser cette tâche.

Références bibliographiques

[BBC+19] 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 *to appear in the Proceedings of ESORICS 2019*, ESORICS '19, 2019.

[BBD+15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 457–485, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[BBD+16] 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 *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 116–129, New York, NY, USA, 2016. ACM.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, pages 16–29, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[BCPZ16] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the isw masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, pages 23–39, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[BJP+15] Aurélie Bauer, Eliane Jaulmes, Emmanuel Prouff, Jean-René Reinhard, and Justine Wild. Horizontal collision correlation attack on elliptic curves. *Cryptography and Communications*, 7(1):91–119, Mar 2015.

[BJPW13] Aurélie Bauer, Eliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal and vertical side-channel attacks against secure rsa implementations. In Ed Dawson, editor, *Topics in Cryptology – CT-RSA 2013*, pages 1–17, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[BRNI13] Ali Galip Bayrak, Francesco Regazzoni, David Novo, and Paolo Ienne. Sleuth: Automated verification of software power analysis countermeasures. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013*, pages 293–310, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[CCJ04] B. Chevallier-Mames, M. Ciet, and M. Joye. Low-cost solutions for preventing simple side-channel analysis: side-channel atomicity. *IEEE Transactions on Computers*, 53(6):760–768, June 2004.

[CFG+10] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In Miguel Soriano, Sihan Qing, and Javier López, editors, *Information and Communications Security*, pages 46–61, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 398–412, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Cor17] Jean-Sebastien Coron. Formal verification of side-channel countermeasures via elementary circuit transformations. Cryptology ePrint Archive, Report 2017/879, 2017. https://eprint.iacr.org/2017/879.

[CPR07] Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side channel cryptanalysis of a higher order masking scheme. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 28–44, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[CPRR14] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *Fast Software Encryption*, pages 410–424, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[EW14] Hassan Eldib and Chao Wang. Synthesis of masking countermeasures against side channel attacks. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification*, pages 114–130, Cham, 2014. Springer International Publishing.

[EWS14] Hassan Eldib, Chao Wang, and Patrick Schaumont. Smt-based verification of software countermeasures against side-channel attacks. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 62–77, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[EWTS14] Hassan Eldib, Chao Wang, Mostafa Taha, and Patrick Schaumont. Qms: Evaluating the side-channel resistance of masked software from source code. In *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, pages 209:1–209:6, New York, NY, USA, 2014. ACM.

[GP99] Louis Goubin and Jacques Patarin. Des and differential power analysis the "duplication" method. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems*, pages 158–172, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[GV10] Christophe Giraud and Vincent Verneuil. Atomicity improvement for elliptic curve scalar multiplication. In Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny, editors, *Smart Card Research and Advanced Application*, pages 80–101, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Lon07] Patrick Longa. *Accelerating the Scalar Multiplication on Elliptic Curve Cryptosystems Over Prime Fields*. PhD thesis, University of Ottawa, 06 2007.

[Mes00] Thomas S. Messerges. Using second-order power analysis to attack dpa resistant software. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2000*, pages 238–251, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[PRB09] E. Prouff, M. Rivain, and R. Bevan. Statistical analysis of second order differential power analysis. *IEEE Transactions on Computers*, 58(6):799–811, June 2009.

[Ron14] Franck Rondepierre. Revisiting atomic patterns for scalar multiplications on elliptic curves. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications*, pages 171–186, Cham, 2014. Springer International Publishing.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of aes. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 413–427, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[SP06] Kai Schramm and Christof Paar. Higher order masking of the aes. In David Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, pages 208–225, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Wal01] Colin D. Walter. Sliding windows succumbs to big mac attack. In *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '01, pages 286–299, London, UK, UK, 2001. Springer-Verlag.

[WS17] Chao Wang and Patrick Schaumont. Security by compilation: An automated approach to comprehensive side-channel resistance. *ACM SIGLOG News*, 4(2):76–89, May 2017.