Cryptographie légère pour l'internet des objets : implémentations et intégrations sécurisées.

Alexandre Adomnicai

N°d'ordre NNT : 2019LYSEM021

THESE de DOCTORAT DE L'UNIVERSITE DE LYON
opérée au sein de
l'Ecole des Mines de Saint-Etienne

Ecole Doctorale N° 488
Sciences, Ingénierie, Santé

Spécialité de doctorat : Microélectronique

Soutenue publiquement le 24/09/2019, par:
Alexandre ADOMNICAI

Cryptographie Légère pour l'Internet des Objets : Implémentations et Intégrations Sécurisées

---

Devant le jury composé de :

Maryline LAURENT	Professeur, Télécom SudParis	Présidente
Pascal BENOIT	Maître de conférences, Université de Montpellier	Rapporteur
Giorgio DI NATALE	Directeur de recherche, CNRS	Rapporteur
Karima BOUDAOUD	Maître de conférences, Université Nice Sophia Antipolis	Examinatrice
Jacques FOURNIER	Ingénieur de recherche, CEA-Leti	Directeur de thèse
Assia TRIA	Ingénieur de recherche, CEA-Leti	Co-directrice de thèse
Laurent MASSON	Directeur R&D, Trusted Objects	Invité

Spécialités doctorales	Responsables :	Spécialités doctorales	Responsables
SCIENCES ET GENIE DES MATERIAUX	K. Wolski Directeur de recherche	MATHEMATIQUES APPLIQUEES	O. Roustant, Maître-assistant
MECANIQUE ET INGENIERIE	S. Drapier, professeur	INFORMATIQUE	O. Boissier, Professeur
GENIE DES PROCEDES	F. Gruy, Maître de recherche	SCIENCES DES IMAGES ET DES FORMES	JC. Pinoli, Professeur
SCIENCES DE LA TERRE	B. Guy, Directeur de recherche	GENIE INDUSTRIEL	N. Absi, Maitre de recherche
SCIENCES ET GENIE DE L'ENVIRONNEMENT	D. Graillot, Directeur de recherche	MICROELECTRONIQUE	Ph. Lalevée, Professeur

EMSE : Enseignants-chercheurs et chercheurs autorisés à diriger des thèses de doctorat (titulaires d'un doctorat d'Etat ou d'une HDR)

ABSI	Nabil	MR	Génie industriel	CMP
AUGUSTO	Vincent	CR	Image, Vision, Signal	CIS
AVRIL	Stéphane	PR2	Mécanique et ingénierie	CIS
BADEL	Pierre	MA(MDC)	Mécanique et ingénierie	CIS
BALBO	Flavien	PR2	Informatique	FAYOL
BASSEREAU	Jean-François	PR	Sciences et génie des matériaux	SMS
BATTON-HUBERT	Mireille	PR2	Sciences et génie de l'environnement	FAYOL
BEIGBEDER	Michel	MA(MDC)	Informatique	FAYOL
BLAYAC	Sylvain	MA(MDC)	Microélectronique	CMP
BOISSIER	Olivier	PR1	Informatique	FAYOL
BONNEFOY	Olivier	PR	Génie des Procédés	SPIN
BORBELY	Andras	MR(DR2)	Sciences et génie des matériaux	SMS
BOUCHER	Xavier	PR2	Génie Industriel	FAYOL
BRODHAG	Christian	DR	Sciences et génie de l'environnement	FAYOL
BRUCHON	Julien	MA(MDC)	Mécanique et ingénierie	SMS
CAMEIRAO	Ana	MA(MDC)	Génie des Procédés	SPIN
CHRISTIEN	Frédéric	PR	Science et génie des matériaux	SMS
DAUZERE-PERES	Stéphane	PR1	Génie Industriel	CMP
DEBAYLE	Johan	MR	Sciences des Images et des Formes	SPIN
DEGEORGE	Jean-Michel	MA(MDC)	Génie industriel	Fayol
DELAFOSSE	David	PR0	Sciences et génie des matériaux	SMS
DELORME	Xavier	MA(MDC)	Génie industriel	FAYOL
DESRAYAUD	Christophe	PR1	Mécanique et ingénierie	SMS
DJENIZIAN	Thierry	PR	Science et génie des matériaux	CMP
BERGER-DOUCE	Sandrine	PR1	Sciences de gestion	FAYOL
DRAPIER	Sylvain	PR1	Mécanique et ingénierie	SMS
DUTERTRE	Jean-Max	MA(MDC)		CMP
EL MRABET	Nadia	MA(MDC)		CMP
FAUCHEU	Jenny	MA(MDC)	Sciences et génie des matériaux	SMS
FAVERGEON	Loïc	CR	Génie des Procédés	SPIN
FEILLET	Dominique	PR1	Génie Industriel	CMP
FOREST	Valérie	MA(MDC)	Génie des Procédés	CIS
FRACZKIEWICZ	Anna	DR	Sciences et génie des matériaux	SMS
GARCIA	Daniel	MR(DR2)	Sciences de la Terre	SPIN
GAVET	Yann	MA(MDC)	Sciences des Images et des Formes	SPIN
GERINGER	Jean	MA(MDC)	Sciences et génie des matériaux	CIS
GOEURIOT	Dominique	DR	Sciences et génie des matériaux	SMS
GONDRAN	Natacha	MA(MDC)	Sciences et génie de l'environnement	FAYOL
GONZALEZ FELIU	Jesus	MA(MDC)	Sciences économiques	FAYOL
GRAILLOT	Didier	DR	Sciences et génie de l'environnement	SPIN
GROSSEAU	Philippe	DR	Génie des Procédés	SPIN
GRUY	Frédéric	PR1	Génie des Procédés	SPIN
HAN	Woo-Suck	MR	Mécanique et ingénierie	SMS
HERRI	Jean Michel	PR1	Génie des Procédés	SPIN
KERMOUCHE	Guillaume	PR2	Mécanique et Ingénierie	SMS
KLOCKER	Helmut	DR	Sciences et génie des matériaux	SMS
LAFOREST	Valérie	MR(DR2)	Sciences et génie de l'environnement	FAYOL
LERICHE	Rodolphe	CR	Mécanique et ingénierie	FAYOL
MALLIARAS	Georges	PR1	Microélectronique	CMP
MOLIMARD	Jérôme	PR2	Mécanique et ingénierie	CIS
MOUTTE	Jacques	CR	Génie des Procédés	SPIN
NAVARRO	Laurent	CR		CIS
NEUBERT	Gilles			FAYOL
NIKOLOVSKI	Jean-Pierre	Ingénieur de recherche	Mécanique et ingénierie	CMP
NORTIER	Patrice	PR1	Génie des Procédés	SPIN
O CONNOR	Rodney Philip	MA(MDC)	Microélectronique	CMP
PICARD	Gauthier	MA(MDC)	Informatique	FAYOL
PINOLI	Jean Charles	PR0	Sciences des Images et des Formes	SPIN
POURCHEZ	Jérémy	MR	Génie des Procédés	CIS
ROUSSY	Agnès	MA(MDC)	Microélectronique	CMP
ROUSTANT	Olivier	MA(MDC)	Mathématiques appliquées	FAYOL
SANAUR	Sébastien	MA(MDC)	Microélectronique	CMP
SERRIS	Eric	IRD		FAYOL
STOLARZ	Jacques	CR	Sciences et génie des matériaux	SMS
TRIA	Assia	Ingénieur de recherche	Microélectronique	CMP
VALDIVIESO	François	PR2	Sciences et génie des matériaux	SMS
VIRICELLE	Jean Paul	DR	Génie des Procédés	SPIN
WOLSKI	Krzystof	DR	Sciences et génie des matériaux	SMS
XIE	Xiaolan	PR0	Génie industriel	CIS
YUGMA	Gallian	CR	Génie industriel	CMP

Mise à jour : 30/08/2018

À ma mère

Remerciements

Nombreux sont ceux qui ont contribué, de près ou de loin, à la rédaction de ce manuscrit.

Je tiens à remercier en premier lieu mon équipe encadrante, Assia, Jacques et Laurent, pour la confiance et la liberté qu'ils m'ont accordées tout au long de ces trois années. Je remercie également l'équipe SAS pour son accueil, et tout particulièrement Mounia et Elias pour leur bonne humeur inconditionnelle, Damien pour ses analyses sportives et Benjamin pour nos échanges et collaborations scientifiques. Je n'oublie évidemment pas mes collègues de Trusted Objects, Hervé, Jean-Pierre, Mathieu, Sami, Ugo et Youssiph pour toutes ces vives discussions lors de nos repas d'équipe. Bien que mon implication au sein du projet collaboratif PACLIDO ne fasse pas partie intégrante de ce manuscrit, je souhaite exprimer ma gratitude aux différents membres avec lesquels j'ai eu plaisir à travailler, notamment l'équipe crypto composée d'Alexis, Christophe, Julien, Kévin, Léo, Marine, Paul et Virginie.

Il m'est inconcevable de ne pas citer ceux qui ont partagé mon éducation et/ou ont contribué à mon épanouissement personnel depuis mon plus jeune âge. Ceux avec qui j'ai vécu toutes ces expériences qui font de moi ce que je suis aujourd'hui : Antho, Auguste, Aziza, Burni, Clément, Dlabit, Djoul, Elnonito, François, Gary, Karim, Léo, Lola, Louis, Louizinio, Paul, Rimka, Sewa, Sly, Vichezer et Vince pour n'en citer qu'une poignée.

Mes pensées vont également à Yves, qui m'a conseillé maintes fois sur le comportement à adopter en tant que doctorant, fort de son expérience de directeur de recherche à l'Institut Necker Enfants Malades. Je me remémore avec plaisir nos soirées dans la chambre de Simon, à converser de tout et de rien, et continue à tirer des leçons de ces échanges.

Ces trois années en Provence m'ont aussi permis de découvrir la vie de couple. Je remercie de tout mon cœur Marie d'avoir eu le courage de s'exiler à mes côtés, de m'avoir accompagné lors de nombreux déplacements, et plus généralement de me rendre profondément heureux chaque jour en partageant ma vie.

Finalement, je dédie cette thèse à ma mère qui m'a inculqué l'importance de suivre des études supérieures. Son parcours m'impressionne à un tel point qu'il m'était impossible de baisser les bras, même lors de mes moments de doute à l'adolescence. J'espère la rendre fière avec ce manuscrit, et lui exprime ma plus sincère reconnaissance et profonde admiration.

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Chapitre 1

Introduction

Sommaire

1.1 L'Internet des objets

1.1.1 Définition

L'Internet des Objets (IdO) est l'un des nouveaux paradigmes auxquels les sociétés humaines se trouvent confrontées. Le concept sous-jacent remonte aux années 1980, lorsqu'un informaticien de l'université de Tokyo, Ken Sakamura, invente le premier système d'exploitation libre [Sak87]. Le succès de ce dernier amène le professeur Sakamura à imaginer un monde dans lequel des ordinateurs miniatures seraient enfouis dans toute sorte d'objets, d'où la notion de *"computing everywhere"* qu'il popularise à l'époque. Il parlera ensuite de *"ubiquitous computing"* et démontrera l'intérêt d'implanter des puces électroniques dans des objets physiques [TS95].

En 1988, c'est Mark Weiser, alors chef du laboratoire Ubicomp au Xerox PARC, qui pose les bases théoriques de l'informatique ubiquitaire en énonçant les principes fondamentaux : l'ordinateur devrait assister l'homme dans son activité ; l'ordinateur ne devrait pas être intrusif ; l'ordinateur devrait être un prolongement de notre inconscient, *etc.* En particulier, il explique que l'informatique ubiquitaire repose sur une architecture distribuée qui comporte un noyau central minime et des périphériques disposant d'une large autonomie [Wei91].

Il faudra attendre plus d'une décénnie avant de voir le terme *"internet of things"* émerger. En 1999, Kevin Ashton, alors gestionnaire de marque chez Procter & Gamble, utilise la notion d'IdO pour la première fois lors d'une présentation devant les patrons de la multinationale, afin d'améliorer l'efficacité de la chaîne d'approvisionnement. Il faudra ensuite attendre plusieurs années pour que des définitions formelles soient énoncées. En 2012, l'Union internationale des télécommunications définit l'IdO comme une *"infrastructure mondiale pour la société de l'information, qui permet de disposer de services évolués en interconnectant des objets (physiques ou virtuels) grâce aux technologies de l'information et de la communication interopérables existantes ou en évolution"*.

Bien que l'IdO ait son origine dans une logique marchande où les articles de consommation doivent être suivis tout au long de leur progression dans la chaîne d'approvisionnement (voire tout au long de leur cycle de vie), il constitue une révolution aux applications innombrables. Ses champs d'application sont si vastes qu'ils concernent aussi bien les services à la personne (*e.g.*, cannes intelligentes pour malvoyants) que la sécurité routière (*e.g.*, voitures autonomes) ou encore l'économie d'énergie (*e.g.*, bâtiments intelligents).

1.1.2 Arrière-plan technique

Un objet connecté intègre généralement un, voire même plusieurs capteurs. Il peut s'agir d'un simple bouton destiné à détecter une pression ou de dispositifs beaucoup plus complexes destinés à détecter des variations environnementales (*e.g.*, luminosité, température, pression). Lors de la fabrication d'un objet connecté, ces capteurs sont soudés sur une carte électronique dotée d'un processeur afin de traiter les données et de les transférer vers le composant émetteur, ce qui nous amène à la notion de connectivité. Les besoins en termes d'énergie et de débit sont si disparates qu'il existe une multitude de technologies de connectivité pour l'IdO. Afin de répondre aux différents cas d'usages, trois grandes catégories de réseaux sont généralement considérées. La figure 1.1 illustre les disparités entre chaque catégorie, selon différents critères.

La première désigne les réseaux locaux à courte portée communément utilisés de nos jours tels que le WiFi et le Bluetooth. Ces technologies, qui ont une portée inférieure à 100m, furent initialement conçues pour échanger des données multimédia comme de la musique ou des vidéos. Le principal inconvénient de ces technologies est la nécessité

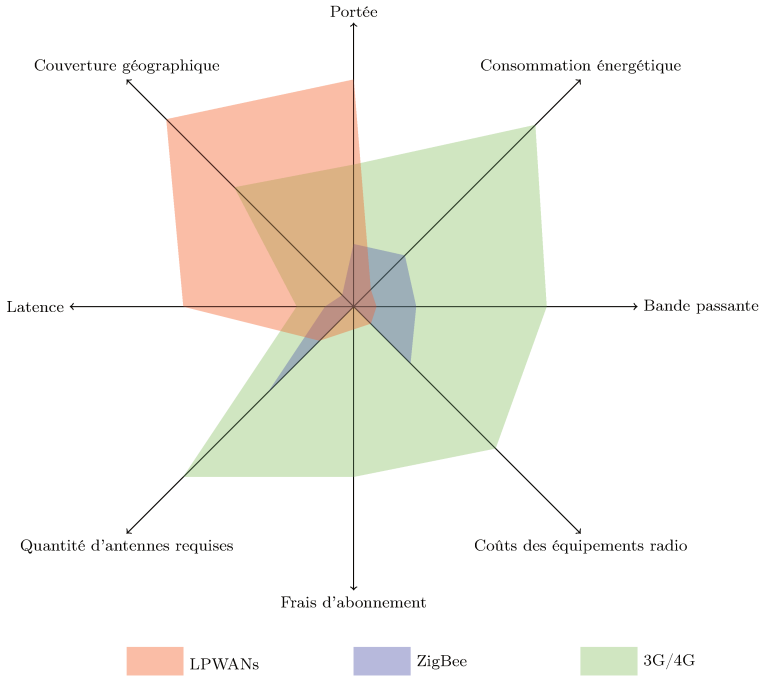


FIGURE 1.1 – Comparaison de différents réseaux dédiés à l'IdO [Egl15]

d'une passerelle entre l'objet et Internet (*e.g.*, routeur WiFi ou smartphone équipé du Bluetooth). Par ailleurs, la consommation de courant est non négligeable étant donnée la nature des données transmises. Les réseaux classiques de télécommunications utilisés par nos smartphone afin de se connecter à Internet (*e.g.*, 4G) présentent l'avantage d'être entièrement déployés à l'échelle mondiale et d'offrir des débits particulièrement élevés. En contrepartie, ils sont souvent onéreux et extrêmement énergivores. S'il est envisageable de recharger un téléphone tous les soirs, cela n'est pas le cas pour l'écosystème de l'IdO. La dernière catégorie concerne les réseaux basse consommation à longue portée, communément appelés LPWAN pour *Low-Power Wide-Area Networks*, qui ont été spécialement conçus pour les objets connectés. Ces derniers utilisent des basses fréquences qui permettent d'atteindre une consommation de courant particulièrement faible moyennant un débit relativement bas. Parce que la plupart des cas d'usages de l'IdO ne nécessitent pas l'échange d'une grande quantité d'informations, un faible débit ne constitue pas systématiquement un désavantage. De surcroît, ces réseaux opèrent sur les bandes de fréquence 868MHz qui présentent l'avantage d'être gratuites et libres de droits. Les acteurs sur ce marché sont nombreux, attirés par les perspectives économiques liées au développement des objets connectés dans les années à venir. À ce jour, plusieurs technologies semblent sortir du lot notamment Sigfox et LoRa qui sont nées en France, et NB-IoT dévelopée par le consortium international 3GPP.

1.1.3 Les enjeux sécuritaires

D'une part, l'IdO laisse entrevoir des promesses considérables, tant pour les entreprises que pour les administrations et les citoyens. D'autre part, son déploiement représente un défi majeur en termes de sécurité et de respect de la vie privée. En effet, des attaques à l'encontre d'objets connectés, toutes plus impressionnantes les unes que les autres, agitent régulièrement la toile du fait de leur impact ou de leur originalité. À titre d'exemple, la plus grande attaque par déni de service jamais exécutée repose sur un logiciel malveillant dédié aux objets connectés, nommé Mirai [AAB+17]. Les appareils infectés par Mirai recherchent en permanence sur Internet des adresses IP correspondant

à des objets connectés, et tentent de s'y connecter grâce à une table d'identifiants et de mots de passe par défaut afin d'y installer le logiciel malveillant. Une fois Mirai installé, les objets continuent de fonctionner normalement, malgré une lenteur occasionnelle et une augmentation de l'utilisation de la bande passante. C'est ainsi que plusieurs attaques par déni de serivce ont été exécutées en septembre 2016, dont une atteignant plus de 600 Gbps de trafic réseau. Dans un autre registre, en juillet 2017, un aquarium connecté dans un casino nord-américain a été utilisé pour extraire plus de 10GB de données du réseau de l'établissement [Sch17].

Ces évènements donnent lieu à une double lecture de l'IdO. La première optimiste, qui consiste à voir un progrès social considérable qui aboutira à la mise en œuvre de services intelligents pour une efficience accrue dans divers domaines. La seconde pessimiste, qui consiste à voir un vecteur de fragilisation de la société tout entière en mettant l'accent sur les vulnérabilités déjà exploitées et les prochaines à venir. En plus des besoins en matière de sécurité afin de prévenir les cyberattaques et d'assurer le bon fonctionnement de l'IdO, de nombreux experts ont tiré la sonnette d'alarme concernant les risques d'atteinte à la vie privée. En effet, les données traitées, collectées et transmises par ces objets qui nous entourent révèlent des informations sur notre environnement, nos habitudes, comportements, activités privées voire notre profil psychologique. Considérons le cas des montres connectées qui mesurent le rythme cardiaque, la tension artérielle et les kilomètres parcourus : qu'advient-il des masses de données collectées ? Sont-elles exclusivement utilisées par l'application associée à la montre en question ou sont-elles revendues à des tiers ? Les utilisations potentielles sont multiples du fait que ces données intéressent des assureurs, des établissements de crédit ou tout type d'entreprise commerciale susceptible de proposer de nouveaux produits ou services sur la base de ces données. Afin de responsabiliser les acteurs de l'IdO, les institutions légifèrent activement sur la gestion des données engendrées. Le règlement général sur la protection des données (RGPD) adopté par le Parlement européen en 2016, stipule que la collecte des données à caractère personnel par les objets connectés doit être proportionnée et pertinente par rapport à l'usage qu'il est prévu d'en faire. Ce texte inscrit également la notion de *"data protection by design"* qui consiste à considérer les contraintes de confidentialité dès la conception du produit.

Outre les problèmatiques liées aux données personnelles, la sécurité est essentielle au déploiement de l'IdO. En effet, les objets étant accessibles à distance de par leur connectivité, des citoyens malintentionnées pourraient être tentés de les faire dysfonctionner. Par exemple, dans le cas d'une ville équipée de lampadaires connectés, une attaque à l'encontre de ces derniers pourrait semer la peur et l'insécurité auprès de la population. Ainsi, pour répondre aux éxigences sécuritaires de l'IdO, il est nécessaire d'avoir recours à différents outils, dont des algorithmes cryptographiques.

1.2 La cryptologie

1.2.1 Définition et terminologies

La cryptologie, étymologiquement la science du secret, englobe deux champs d'étude. Tout d'abord la *cryptographie*, qui à l'aide d'un secret, a pour but de garantir les propriétés de confidentialité (l'information n'est accessible qu'à ceux dont l'accès est autorisé), d'intégrité (l'information n'a pas été altérée) et d'authenticité (l'information est intègre atteste sa provenance). Ensuite la *cryptanalyse*, qui consiste à mettre en échec les algorithmes cryptographiques afin d'extraire ou d'altérer les informations protégées, sans avoir accès au secret.

Afin de garantir la confidentialité d'une information, l'expéditeur *chiffre* le *texte clair* M à l'aide d'une fonction $\mathscr{E}$ paramétrée par une *clef* K, et obtient ainsi un *texte chiffré* C. Le destinataire *déchiffre* C à l'aide d'une fonction D paramétrée par une clef K' afin d'obtenir M. L'action de retrouver le texte clair à partir du chiffré sans la connaissance de la clef est appelée *décryptage*. Les procédures de chiffrement et de déchiffrement peuvent être exprimées mathématiquement comme suit :

$$\mathscr{E}_K(M) = C \qquad\qquad \mathscr{D}_{K'}(C) = M\,.$$

Dans le cas où la même clef est utilisée pour chiffrer et déchiffrer (*i.e.*, $K = K'$), on parle de chiffrement *symétrique*. Dans le cas contraire, on parle de chiffrement *asymétrique*.

1.2.2 Cryptographie

1.2.2.1 Cryptographie symétrique

Algorithmes de chiffrement La cryptographie est utilisée depuis l'aube de notre civilisation pour dissimuler des informations secrètes. Un des chiffrements les plus emblématiques de l'Antiquité est le chiffre de César, que ce dernier utilisait pour communiquer avec ses armées. Il s'agit d'un chiffrement par décalage, le principe étant de remplacer chaque lettre du texte clair par une lettre à distance fixe, en conservant l'ordre alphabétique. La clef est donc définie par la longueur du décalage, qui ne peut prendre que 25 valeurs pour l'alphabet latin. Par conséquent, un attaquant ayant connaissance du mécanisme de chiffrement est en mesure d'essayer toutes les clefs possibles, une par une, en un temps raisonnable. S'en suivirent de nombreuses avancées en cryptologie afin d'améliorer les techniques de chiffrement et de cryptanalyse au fil des siècles.

En 1883, le cryptologue Auguste Kerckhoffs énonce les principes de Kerckhoffs [Ker83]. L'un d'eux stipule que la sécurité d'une fonction de chiffrement doit reposer uniquement sur la clef, et non sur le secret de la définition de la fonction. En d'autres termes, la clef doit pouvoir prendre suffisamment de valeurs pour qu'une recherche exhaustive soit impossible en pratique.

L'avènement des ordinateurs au XX^e siècle révolutionne la cryptanlayse puisque cette dernière peut être dorénavant automatisée. En 1949, Claude Elwood Shannon fonde les bases de la cryptographie moderne en introduisant la notion d'entropie d'un langage et celle de chiffrement parfait [SW49]. Notamment, il énonce qu'afin de construire un système cryptographique sûr, il est nécessaire que la redondance au sein du texte clair ne soit pas visible sur le texte chiffré. Pour ce faire, Shannon définit la notion de diffusion, qui indique que chaque bit du texte clair doit affecter un grand nombre de bits du texte chiffré, et celle de confusion, qui dit que chaque bit du texte chiffré doit dépendre des bits du texte clair et de la clef de manière hautement non-linéaire.

En 1977, le Data Encryption Standard (DES) est proposé comme standard de chiffrement par le *US National Bureau of Standards*. Dès sa publication, le DES est critiqué par la communauté cryptographique pour sa taille de clef trop petite (64 bits mais uniquement 56 effectifs). À la fin des années 1990, différentes attaques à son encontre ont été démontrées. Notamment celle réalisée en 1999 par distributed.net a réussi à retrouver la clef de chiffrement en moins de 24 heures en s'appuyant sur environ 100 000 ordinateurs connectés à Internet [Zan01, McN99].

En 1997, l'agence américaine *National Institute Standard of Technology* lance un appel à candidature pour remplacer le DES. En 2001, une instance particulière du candidat Rijndael est sélectionnée comme vainqueur de la compétition et devient l'Advanced Encryption Standard (AES). L'algorithme est décliné en trois configurations qui utilisent

respectivement une clef de 128, 192 et 256 bits. Après une vingtaine d'années de cryptanalyse, l'AES est toujours considéré comme mathématiquement sûr et reste l'algorithme de chiffrement symétrique le plus utilisé à travers le monde.

L'AES et le DES constituent tous les deux des *chiffrements par bloc*, ce qui signifie qu'ils prennent n bits du message clair en entrée et produisent n bits chiffrés en sortie. Afin de chaîner de manière sûre les appels aux primitives dans le but de chiffrer des données de longueur arbitraire, les chiffrements par bloc sont en pratique combinés avec des *modes d'opération*. Une autre variante, les *chiffrements à flot*, chiffrent chaque bit indépendamment les uns des autres. Bien qu'utilisés dans divers cas d'usage (*i.e.*, l'algorithme A5/1 pour les données GSM [Can11]), les chiffrements à flot sont généralement délaissés au profit de leurs homologues opérant par bloc. Cela s'explique principalement par la possibilité de transformer un chiffrement par bloc en un chiffrement à flot à l'aide d'un mode d'opération adéquat [Dwo07]. De plus, les chiffrements par bloc sont généralement plus simples à analyser et par conséquent, ont été significativement plus étudiés par la communauté scientifique. Cependant, les chiffrements à flot restent d'un grand intérêt car ils permettent généralement d'atteindre des performances compétitives pour une taille d'implémentation très compacte [wol10].

Fonctions de hachage cryptographiques Outre la propriété de confidentialité d'un message, il est aussi désirable de vouloir en assurer l'intégrité. Pour ce faire, il est courant de s'appuyer sur une *fonction de hachage*. Ces fonctions ont pour rôle d'associer, à une donnée de taille arbitraire, une image de taille fixe appelée *empreinte*. Une fonction de hachage repose sur un algorithme public où aucune clef ou autre donnée secrète n'intervient dans le calcul. Néanmoins, on qualifie ces fonctions de cryptographiques car elles respectent des propriétés qui apportent des garanties sécuritaires.

Assurer l'authenticité Bien que les fonctions de hachage assurent la propriété d'intégrité d'une donnée, en aucun cas elle n'en assure l'authenticité. En effet, l'algorithme étant public et ne nécessitant aucun secret, il est possible de modifier le message et de calculer la nouvelle empreinte associée. Afin de garantir l'authenticité, il est nécessaire d'avoir recours à l'utilisation d'un secret. Comme pour les fonctions de hachage, le principe est d'associer à une donnée de taille arbitraire une image de taille fixe. Dans ce cas, l'image en question est appelée *message authentication code* (MAC). Ces algorithmes s'appuient généralement sur des primitives cryptographiques usuelles. Par exemple, on peut utiliser une fonction de hachage en la jumelant avec la construction HMAC [BCK96] standardisée par le NIST en 2001, ou encore se baser sur un algorithme de chiffrement par bloc en utilisant la construction CBC-MAC [cbc99]. Il existe également des algorithmes de *chiffrement authentifiés* qui assurent à la fois confidentialité, authenticité et intégrité. Certains modes d'opération permettent, à partir d'un simple chiffrement par bloc, d'aboutir à ce type de construction *(e.g.*, Galois/Counter mode [Dwo07]).

1.2.2.2 Cryptographie asymétrique

L'inconvénient majeur de la cryptographie symétrique est la nécessité d'échanger la clef de chiffrement au préalable afin d'établir un canal de communication sécurisé. Afin de s'affranchir de cette contrainte, en 1976, Whitfield Diffie et Martin Hellman introduisent la notion de cryptographie asymétrique [DH76].

Échange de clef Diffie-Hellman Également appelée crytographie à clef publique, elle assume que chaque entité possède une paire de clefs. La première, nommée clef privée, est uniquement connue par son propriétaire et lui sert à déchiffrer les messages qui lui

sont adressés. La seconde, nommée clef publique, est connue de tous et est utilisée pour chiffrer les messages à l'attention du propriétaire de la clef privée associée. Ces techniques sont fondées sur le concept de *fonction à sens unique* et à *brèche secrète*. Une fonction à sens unique est facile à calculer mais difficile (calculatoirement parlant) à inverser. Néanmoins, l'existence d'une brèche secrète (*i.e.*, la clef privée) permet, à ceux qui en ont connaissance, de facilement inverser la fonction à sens unique. La publication de Diffie et Hellman illustre comment deux entités, chacune en possession d'une telle paire de clefs, peuvent procéder à l'établissement d'un secret partagé qui peut être utilisé comme clef symétrique par la suite.

Cryptosystème RSA Il fallut attendre 1978 pour que Ronald Rivest, Adi Shamir et Leonard Adleman publient le premier système de chiffrement asymétrique [RSA78]. La fonction à sens unique à brèche secrète de l'algorithme RSA repose sur la *décomposition de grands nombres en facteurs premiers*, qui est un problème mathématique difficile à résoudre. Afin de chiffrer ou déchiffrer un message, il faut avoir recours à des opérations d'arithmétique modulaire dont le module public n est le produit de deux grands nombres premiers secrets p et q. Décrypter un message revient à retrouver les facteurs premiers de n. Ainsi, pour assurer la sécurité de l'algorithme RSA, il convient de choisir p et q suffisamment grands pour que le procédé de factorisation soit impossible en pratique. Lors de sa conception, il était recommandé d'utiliser un module n de 512 bits (soit le produit deux nombres premiers de 256 bits). Avec l'évolution de la puissance de calcul des ordinateurs, les différents organismes de sécurité recommandent l'utilisation d'un module d'au moins 3072 bits pour une utilisation au-delà de 2030.

Cryptographie sur courbes elliptiques Au milieu des années 1980, l'utilisation des courbes elliptiques pour la crytographie à clef publique, nommée ECC pour *Elliptic Curve Cryptography*, est suggérée par Neal Koblitz [Kob87] et Victor Miller [Mil86]. Le problème mathématique sous-jacent est celui du *logarithme discret*, comme pour l'échange de clef Diffie-Hellman. Le principal avantage des courbes elliptiques est que les structures mathématiques qui en découlent sont plus complexes à manipuler que celles définies par les entiers modulo n. Ainsi, résoudre le problème du logarithme discret sur une courbe elliptique est réputé plus difficile, ce qui permet d'utiliser des tailles de clefs plus petites pour un même niveau de sécurité. De plus, les calculs arithmétiques sous-jacents opèrent sur de plus petits entiers, ce qui permet d'atteindre des implémentations compactes pour les systèmes embarqués. Le tableau 1.1 compare les différentes tailles de clefs pour plusieurs niveaux de sécurité exprimés en bits. En cryptographie, un algorithme qui jouit d'un niveau de sécurité de n bits signifie que 2^n opérations sont nécessaires pour réduire sa sécurité à néant.

TABLEAU 1.1 – Comparatif des tailles de clefs (en bits) par niveau de sécurité [ECR18]

Bits de sécurité	Taille de module RSA	Taille de clef ECC
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

1.2.3 Cryptanalyse

La cryptanalyse est la science qui a pour but de mettre en péril un algorithme cryptographique sans avoir accès au secret. La méthode la plus naïve qui soit, appellée *attaque par force brute*, consiste à tester chaque valeur possible de la clef. Pour tout algorithme respectant les principes de Kerckhoffs, cette attaque est impossible à mener en pratique étant donné que l'ensemble des valeurs possibles étant trop important. Le tableau 1.2 fournit quelques ordre de grandeurs significatifs, qui illustrent la complexité de considérer les 2^n hypothèses de clef pour différentes valeurs de n.

TABLEAU 1.2 – Ordres de grandeur significatifs

Nombre de grains de sable dans le Sahara	$10^{23} \approx 2^{77}$
Nombres d'atomes constituant la Terre	$10^{51} \approx 2^{170}$
Nombres d'atomes constituant l'Univers	$10^{77} \approx 2^{265}$

La cryptanalyse ne désigne pas nécessairement un moyen pratique de casser un algorithme cryptographique, mais toute méthode qui exploite une faiblesse dans la conception ce dernier afin d'offrir une complexité inférieure à l'attaque par force brute. Par exemple, la meilleure attaque publiée à ce jour à l'encontre du chiffrement AES-128 requiert 2^{126} opérations [TW15], ce qui est quatre fois moins qu'une attaque par force brute mais reste largement infaisable en pratique. Les différentes techniques de cryptanalyse sont classifiées selon les modèles sur lesquels elles reposent. Certains d'entre eux sont basés sur des hypothèses non réalistes comme par exemple, une trop grande quantité de stockage ou un nombre important de textes clairs connus. Néanmoins, les attaques qui en découlent permettent de souligner certaines faiblesses dans la conception de l'algorithme.

Ces attaques analysent uniquement les algorithmes cryptographiques tels qu'ils sont définis mathématiquement. Néanmoins, la cryptographie moderne a pour but d'être finalement exécutée par un processeur. Pour ce faire, les algorithmes doivent être implémentés en pratique de manière logicielle ou matérielle. En donnant vie à ces objets mathématiques abstraits, nous leur procurons des défauts, un comportement, voire un caractère qui sont corrélés aux informations qu'ils traitent. L'exploitation de ces informations afin de mettre en échec la sécurité de l'algorithme en question est applée cryptanalyse physique.

1.3 Les attaques physiques

1.3.1 Principe et classification

L'instanciation logicielle ou matérielle d'un algorithme cryptographique le soumet aux dures réalités des lois de la physique et introduit des nouveaux vecteurs d'attaque. Les attaques physiques consistent à tirer profit des phénomènes physiques liés à l'implémentation de mécanismes de sécurité afin de contourner ces derniers. Comme pour tout système physique, les techniques d'attaque sont variées et dépendent de la cible que l'on cherche à mettre en échec. À titre d'exemple, considérons le cas du coffre-fort avec serrure à combinaison mécanique. Un attaquant peut d'abord tenter de récupérer la combinaison qui protège l'accès au coffre à l'endroit où elle est stockée. Si cette option n'est pas envisageable, il peut alors tenter de mettre en échec la serrure à l'aide d'un stéthoscope en exploitant les cliquetis émis par les disques mécaniques, simplement en manipulant le cadran. Ainsi, l'implémentation physique du système donne lieu à une fuite d'information sonore qui permet de contourner la complexité de l'attaque par force

brute. Ce type d'attaque physique est appellé *attaque par observation*. Finalement, il reste également la possibilité d'altérer le fonctionnement de la serrure, à l'aide d'outils puissants. Ce type d'attaque physique est appellé *attaque par perturbation*. La technique à adopter dépend non seulement des moyens à disposition de l'attaquant, mais également des contraintes liées au niveau de dégradation de la cible.

Les techniques d'attaques à l'encontre de fonctionnalités sécuritaires implémentées sur des composants électroniques sont tout aussi variées. Ces dernières sont classifiées selon le degré d'altération du composant. On distinguait initialiement deux principales catégories : les *attaques non-invasives* qui n'affectent pas l'intégrité du circuit, contrairement aux *attaques invasives* [AK96]. En 2002, Sergei Skorobogatov et Ross Anderson introduisent une troisième catégorie à mi-chemin entre les deux précédentes, nommée *attaques semi-invasives* [SA02].

1.3.1.1 Attaques invasives

Les attaques invasives commencent généralement par une étape de décapsulation, qui consiste à ôter le boîtier du composant, afin de mettre le cricuit à nu. Une fois décapsulé, l'attaquant a un accès direct aux différents éléments du circuit ce qui lui permet d'effectuer différents types d'attaques. Parmi les plus puissantes, les attaques par *micro-sondage* consistent à placer une micro-sonde suffisament proche d'un fil conducteur de façon à pouvoir lire la valeur des signaux internes. Ainsi, en accédant au bus de donnée du CPU, il est possible d'extraire les données contenues en mémoire qui transitent sur ce bus, telles que des clefs cryptographiques. Néanmoins, ces attaques sont extrêmement coûteuses à mettre en pratique du fait qu'elles nécessitent beaucoup de matériel ainsi qu'une grande expertise.

1.3.1.2 Attaques semi-invasives

Tout comme les attaques invasives, les attaques semi-invasives nécessitent d'avoir accès à la surface de la puce et donc une phase de décapsulation, totale ou partielle. En revanche, aucune connexion directe à la surface métallique n'est requise, ce qui réduit considérablement les besoins en termes de matériel. Le principe est d'avoir accès aux entrailles du composant afin d'observer ou de perturber ce dernier de manière plus précise qu'avec une attaque non-invasive.

1.3.1.3 Attaques non-invasives

Les attaques non-invasives ne nécessitent pas l'accès à la surface du circuit attaqué et opèrent sans affecter son intégrité. De plus, elles sont souvent peu coûteuses du fait du peu de matériel qu'elles requièrent. De ce fait, elles sont considérées comme étant les plus réalistes dans de nombreux cas pratiques. Comme pour les attaques semi-invasives, les attaques non-invasives peuvent aussi bien être des attaques par observation que des attaques par perturbation.

1.3.2 Attaques par observation

La génèse des attaques par canaux auxiliaires sur circuits intégrés remonte à 1996, lorsque Paul Kocher publie les premiers travaux qui mettent en échec la sécurité d'algorithmes cryptographiques asymétriques en analysant leur temps d'exécution [Koc96]. Trois ans plus tard, il démontre la possibilité d'exploiter les variations de consommation de courant et utilise ce canal auxiliaire afin de venir à bout d'une implémentation embarquée de l'algorithme DES [KJJ99]. En 2001, Jean-Jacques Quisquater et David Samyde

soulignent le fait qu'un attaquant peut également tirer parti du rayonnement eléctromagnétique [QS01]. De nombreux travaux de recherche suivirent afin d'identifier de nouveaux canaux auxilaires. Par exemple en 2004, Adi Shamir et Eran Tromer émettent l'hypothèse que l'analyse acoustique d'un circuit électronique pourrait être concluante. Leur hypothèse est validée en 2014 lorsqu'ils présentent la première attaque via ce canal auxiliaire à l'encontre de l'algorithme RSA [GST14]. Plus récemment, les publications des attaques Meltdown [LSG+18] et Spectre [KHF+19] ont révélé que les techniques d'exécution spéculative, implémentées dans le but d'accélérer la vitesse de traitement des processeurs, peuvent être exploitées afin d'accéder à des données sensibles stockées en mémoire. Cependant, les optimisations responsables de ces attaques ne sont implémentées que sur les processeurs les plus sophistiqués, qui ne sont pas destinés à l'IdO.

Parmi tous les canaux auxiliaires susmentionnés, le temps d'exécution, la consommation de courant ainsi que le rayonnement eléctromagnétique sont les plus largement exploités à l'égard des composants embarqués, notamment pour leur simplicité et efficacité en pratique. Le reste de cette section présente en détails la façon dont ces phénomènes physiques peuvent être exploités en cryptanalyse.

1.3.2.1 Fuites par temps de calcul

Les fuites par temps de calcul peuvent provenir de l'implémentation elle-même, ou bien de l'architecture du processeur sur lequel l'algorithme est exécuté.

On appelle implémentation non constante une implémentation qui contient des branches conditionnelles qui dépendent d'une donnée en entrée de l'algorithme. Si les temps d'exécution des différentes branches conditionnelles diffèrent, il est alors possible de déduire laquelle est exécutée, et par conséquent, de retrouver la valeur de la condition de test. Ces attaques sont non-invasives et ne nécessitent pas une proximité physique avec la cible dès lors qu'il est possible d'envoyer des requêtes cryptographiques à distance et de mesurer les temps de réponse correspondants. Les algorithmes asymétriques sont particulièrement sujets aux implémentations non constantes, notamment parce qu'ils requièrent des opérations sur de grands entiers. Néanmoins, ces branches conditionnelles sont généralement faciles à contourner au prix d'un temps d'exécution globalement plus long, mais constant. À titre d'exemple, l'algorithme 1.1a présente une branche conditionnelle dépendant d'une variable s tandis que l'algorithme 1.1b présente une manière simple de la convertir en temps constant.

Algorithme 1.1a Test conditionnel non constant

```
1: si s = 1 alors
2:     retourner a
3: sinon
4:     retourner a + b
5: fin si
```

Algorithme 1.1b Test conditionnel constant

```
1: retourner a + (1 − s)b
```

Le temps d'exécution d'une primitive cryptographique peut également varier selon la plateforme sur laquelle elle s'exécute, et ce même en l'absence de branche conditionnelle. En effet, la plupart des processeurs embarquent aujourd'hui des mémoires caches qui enregistrent temporairement des données lues en mémoire afin de diminuer le temps d'un accès ultérieur à ces mêmes données, l'accès à la mémoire externe étant coûteux pour un CPU. Cela devient problématique lorsque les algorithmes utilisent des tables de précalculs pour accélérer leur exécution. Très répandue au sein des algorithmes symétriques, cette technique consiste à précalculer pour toutes les valeurs possibles en entrée, les valeurs correspondantes en sortie. Le processeur se retrouve donc à exécuter des accès mémoires

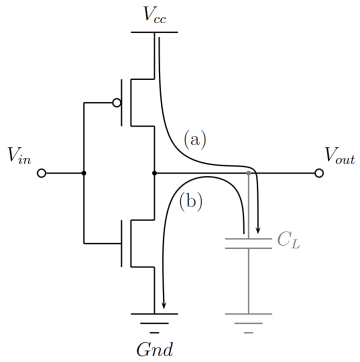


FIGURE 1.2 – Inverseur CMOS

de type $T[n \oplus k]$ où T désigne la table de précalculs, n une partie du texte clair et k une partie de la clef. Selon si la donnée à laquelle on cherche à accéder a déjà été placée dans un cache ou non, l'accès mémoire sera plus ou moins rapide. Par conséquent, l'exécution de l'algorithme ne s'effectue pas en temps constant et dépend du temps requis par l'accès à $T[n \oplus k]$, qui dépend lui même de la valeur de la sous-clef k. Ainsi, l'accès à une donnée dans un tableau, qui s'effectue théoriquement en temps constant, introduit des vulnérabilités une fois instancié sur un processeur. Ce vecteur d'attaque a d'abord été théoriquement étudié en 2002 par Dan Page afin de casser le DES [Pag02]. Les premières attaques pratiques suivirent par la suite : en 2003 contre le DES [TSS+03] et en 2005 contre l'AES [Ber05].

1.3.2.2 Fuites par consommation de courant

De la même manière qu'un compteur électrique doit parfois faire face à des pics de consommation, un circuit intégré a une consommation variable en fonction des opérations qu'il doit traiter. Par exemple, un inverseur *Complementary Metal-Oxyde Semiconductor* (CMOS), qui constitue la brique élémentaire de tout processeur, est lui-même constitué de deux transistors, respectivement appelés transistors PMOS et NMOS. Ce sont les changements d'état de ces transistors qui sont responsables des variations de consommation de courant, comme illustré par la figure 1.2. Ces transistors se comportent comme des vannes et des interrupteurs. Lorsque la tension d'entrée V_{in} passe à l'état logique 0, le transistor NMOS s'ouvre tandis que le PMOS se ferme et charge le condensateur C comme illustré par la flèche (a). Par la suite, lorsque V_{in} est à l'état logique 1, le transistor NMOS se ferme tandis que le PMOS s'ouvre, ce qui ramène à la masse les deux bornes du condensateur C_L et induit le phénomène de décharge comme illustré par la flèche (b). Lorsqu'il n'y a pas de variation de la tension d'entrée, la consommation du circuit est très faible et est appelée *consommation statique*. En revanche, lorsque la tension d'entrée change d'état logique, soit le condensateur se vide dans la masse via le transistor NMOS, soit il se charge via le transistor PMOS. Ce phénomène est appelé *consommation dynamique*.

Par conséquent, les fluctuations de consommation peuvent être analysées à l'aide d'outils statistiques afin d'obtenir des informations sur les données manipulées par le circuit. Différentes techniques d'analyse sont détaillées en section 4.2. Bien que la mesure des fluctuations de consommation puisse nécessiter des modifications sur le circuit (*e.g.*, l'insertion d'une resistance) ces attaques sont généralement considérées comme non-invasives.

1.3.2.3 Fuites par rayonnement électromagnétique

Grâce aux travaux de James Clark Maxwell [Max], il est connu que la circulation d'un courant dans un conducteur génère des champs électromagnétiques au voisinage de ce dernier. Par conséquent, les variations de consommation de courant induisent des rayonnements électromagnétiques qui constituent un canal auxiliaire supplémentaire. Bien que les fuites à l'origine des deux canaux susmentionnés soient directement liées, l'exploitation des émanations électromagnétiques présente plusieurs avantages en pratique.

Mesures localisées Le principal inconvénient de la consommation de courant comme canal auxiliaire est le fait qu'il n'y a pas que les condensateurs des transistors qui induisent une consommation dynamique. Par conséquent, les mesures de tension aux bornes d'une résistance reflètent l'activité globale du composant, y compris les consommations parasites. Afin de ne prendre en compte que les mesures liées à l'activité des transistors lors de la manipulation du secret, l'exploitation des émanations électromagnétiques constitue une alternative pertinente. En effet, les sondes électromagnétiques (généralement des boucles ou des solénoïdes) focalisent uniquement les lignes de champs électromagnétiques présentes à l'extrémité de leurs capteurs. Ainsi, en utilisant des sondes assez précises, il est alors possible de capter les fuites à un endroit précis tout en omettant les champs émis sur le reste de la surface du composant.

Intégrité du composant cible Une mesure d'émanations électromagnétiques nécessite simplement de positionner une sonde à la surface du composant. Comme le stétoscope pour l'exemple du coffre-fort, il est généralement nécessaire de coupler la sonde à un amplificateur afin d'accroître l'amplitude du signal mesuré. Dans le cas où l'attaquant n'a pas connaissance de l'architecture du composant qu'il essaie de mettre en échec, il doit néanmoins réaliser des cartographies spatiales en faisant varier la position de la sonde afin de localiser les zones de fuites. Ces attaques peuvent donc être exécutées de manière non-invasive. Néanmoins, il n'est pas sans intérêt de les réaliser de façon semi-invasive en plaçant la sonde au plus près du circuit, certains boîtiers intégrant un dispositif de blindage électromagnétique afin de réduire les fuites.

1.3.3 Attaques par perturbation

Tandis que les attaques par observation se limitent à l'analyse des fuites liées à un calcul cryptographique, les *attaques par perturbation* ont pour but d'introduire des erreurs dans ces calculs. Ces erreurs peuvent être exploitées afin de contourner des mécanismes de sécurité. Par exemple dans le cas d'une authentification par mot de passe, il est possible de forcer l'authentification à réussir en toutes circonstances (*e.g.*, même avec un mauvais mot de passe) en perturbant le composant lors de l'étape de vérification. Il est également possible de perturber un calcul cryptographique à des fins cryptanalitiques, en exploitant les résultats érronés dans le but de retrouver la clef.

Les prémices des attaques par perturbation remontent à 1978, lorsque des ingénieurs d'Intel découvrent que la désintégration radioactive de l'uranium et du thorium présents dans les boîtiers d'encapsulation émettent des particules α qui altèrent les données stockées en mémoire [MW78]. L'année suivante, un programme de recherche financé par la NASA démontre que les rayonnements cosmiques ont des effets indésirables sur différentes technologies mémoires [SPB+79, KBA+79]. Le risque que représente ce genre d'attaques à l'encontre d'un algorithme cryptographique est démontré théoriquement en 1996 par Dan Boneh, Richard DeMillo et Richard Lipton qui expliquent comment retrouver une clef privée RSA à partir d'une unique signature erronée [BDL97]. Peu après,

Eli Biham et Adi Shamir introduisent un nouveau modèle d'attaque généralisé à tous les algorithmes de chiffrement par bloc, appelé DFA pour *Differential Fault Analysis* [BS97]. Il existe de nombreux moyens techniques pour provoquer une faute. Les plus populaires d'entre eux sont présentés ci-dessous. À noter que la réalisation pratique d'une telle attaque nécessite généralement une synchronisation temporelle précise afin d'injecter la faute au bon moment.

1.3.3.1 Variations de tension d'alimentation et d'horloge

Il existe de nombreux moyens pour pertuber un circuit afin de provoquer une faute lors d'un calcul. Typiquement, un circuit est capable de tolérer une variation d'alimentation électrique de ±10% par rapport à sa tension nominale. En dehors de cette marge de tolérence, le circuit ne fonctionne plus correctement. Par exemple, en réduisant la tension d'alimentation, les signaux mettent plus de temps à traverser les transistors et ne peuvent pas être stabilisés à leur valeur correcte lors du front d'horloge suivant. Ainsi, si le laps de temps durant lequel la tension est réduite est assez court, une erreur sera générée [ABF+03]. De la même manière, il est possible de produire une faute en raccourcissant un unique cycle d'horloge en dessous du temps nécessaire aux signaux pour se stabiliser [ADN+10]. Bien que le matériel requis pour la mise en pratique de ces attaques reste abordable, elles présentent l'inconvénient d'être peu précises, dans le sens où les erreurs générées peuvent affecter plusieurs registres et compliquer l'exploitation des fautes.

1.3.3.2 Tirs laser

Un moyen beaucoup plus pointu est l'utilisation d'un laser. Cette technique, qui nécessite la décapsulation du composant et éventuellement une diminution de l'épaisseur du silicium afin d'assurer une bonne pénétration du faisceau laser, est d'une précision inégalée. En effet, un paramétrage judicieux du laser (longueur d'onde, largeur du faisceau, ...) peut permettre de cibler précisément le transistor à perturber [SA02]. En revanche, les équipements les plus sophistiqués sont très onéreux.

1.3.3.3 Injection électromagnétique

À mi-chemin entre les attaques laser et les attaques par perturbation d'horloge ou de tension, les attaques par injection de fautes électromagnétiques offrent un bon compromis entre coût, précision et simplicité de mise en œuvre. Ces attaques consistent à placer une antenne électromagnétique à quelques millimètres du composant et d'envoyer une pulsation de courant dans cette antenne. Le couplage électromagnétique entre l'antenne et les lignes métalliques à la surface de la puce provoque alors des courants dans ces lignes couplées (bus de données, ligne d'alimentation, ...). Ces attaques peuvent être réalisées de manière non-invasive et il permettent de cibler une partie précise du composant, selon la position de l'antenne.

1.3.3.4 Rowhammer

Il est également possible de créer des fautes matérielles de manière purement logicielle. En 2014, des chercheurs d'Intel et de l'université de Carnegie Mellon ont démontré en pratique la possibilité d'inverser de manière déterministe les bits d'une cellule de mémoire dynamique à accès aléatoire (DRAM) [KDK+14]. Ce phénomène, appelé Rowhammer, provient d'un effet secondaire imprévu de la technologie DRAM qui provoque une fuite de charge électrique dans des cellules mémoires, qui elle même provoque une intéraction électrique avec les cellules voisinnes. Cette intéraction peut modifier le contenu

des cellules mémoires impliquées. Ainsi, un programme informatique peut modifier des contenus de la mémoire sans même avoir les droits d'accès.

1.4 Contexte de la thèse

1.4.1 Acteurs

Les travaux présentés dans ce manuscrit de thèse ont été effectués dans le cadre du dispositif CIFRE (Convention Industrielle pour la Formation par la REcherche). Le principe de ce dernier est la collaboration entre une entreprise et un laboratoire en vue d'étudier des thématiques de recherche innovantes.

1.4.1.1 Trusted Objects

Fondée en 2014, Trusted Objects est une startup qui propose des solutions pour sécuriser les objets connectés. Son offre inclut principalement deux catégories de solutions : les logiciels de sécurité embarqués ainsi que les solutions de personnalisation et de provisionnement de clefs.

Logiciels de sécurité embarqués Les logiciels de sécurité embarqués développés par Trusted Objects sont destinés à être intégrés au sein d'un objet connecté afin d'augmenter le niveau de sécurité de ce dernier en apportant une garantie de sécurité face aux attaques physiques. Selon les cas d'usages, ces logiciels peuvent être utilisés uniquement dans le cadre de certains calculs cryptographiques, mais aussi plus largement pour protéger tous les points sensibles du protocole de communication (*e.g.*, gestion efficace d'un compteur en mémoire pour prévenir les attaques par rejeu).

Solutions de personnalisation et de provisionnement de clefs Peu importe l'application ou le schéma de sécurité qui lui est associé, un objet connecté a besoin à un moment ou un autre dans son cycle de vie de recevoir des clefs cryptographiques et d'autres attributs qui lui sont propres (*e.g.*, identifiants uniques et adresses réseaux). L'étape où ces données sont programmées dans la mémoire de l'objet est appelée *personnalisation*. Elle est souvent vue comme un lourd fardeau dans le processus de fabrication. Afin de pallier cet obstacle au développement de la sécurité dans l'IdO, Trusted Objects développe des offres de personnalisation adaptées aux nombreux microcontrôleur et réseaux du domaine.

1.4.1.2 Équipe Systèmes et Architectures Sécurisées

L'équipe SAS (Systèmes et Architectures Sécurisées) est une équipe de recherche commune entre le CEA-Tech et l'École des Mines de Saint-Étienne. Implantée au Centre de Microélectronique en Provence, ses thématiques de recherche concernent les attaques physiques, de la définition théorique de nouvelles attaques jusqu'à leurs réalisations en pratique, ainsi que la conception de contremesures.

1.4.2 Objectifs

Les logiciels de sécurité embarqués développés par Trusted Objects, tout comme la plupart des réseaux de l'IdO, utilisent à ce jour essentiellement des primitives cryptographiques traditionnelles, telles que l'AES, afin de sécuriser les communications. Bien que ces algorithmes garantissent un haut niveau de sécurité, ils n'ont pas été construits dans le but d'être particulièrement efficaces sur des plateformes contraintes [SNCC18]. Afin de

répondre à ce besoin, la cryptographie légère a pour but de concevoir de nouveaux outils adaptés à l'ensemble de l'écosystème de l'IdO.

Le but de cette thèse est de contribuer à l'étude de ces nouveaux algorithmes, plus particulièrement à l'étude de leurs implémentations et de leur facilité à intégrer des contremesures face aux attaques physiques. Plus précisément, nos travaux se sont concentrés sur trois algorithmes de chiffrement symétriques, nommés ChaCha20, ACORN et ASCON. Ces choix sont justifiés par le souhait d'être au plus près des besoins industriels. Tandis que ChaCha20 a d'ores et déjà fait ses preuves dans de nombreux produits déployés en pratique [IAN19], ACORN et ASCON sont les deux vainqueurs de la compétition cryptographique CAESAR pour les cas d'usage à ressources limitées. Plus récemment, ASCON a été proposé comme standard dans le cadre du projet de standardisation "Lightweight Cryptography" organisé par le NIST.

Nos principales contributions consistent en la définition théorique de nouvelles attaques par canaux auxiliaires à l'encotre de ces deux algorithmes et également en leur vérification en pratique à travers des expérimentations sur microcontrôleurs. En particulier, notre attaque contre ACORN est la première et unique attaque par observation publiée à l'encontre de cet algorithme à ce jour. Nous avons également appliqué les dernières contremesures publiées dans la littérature afin d'évaluer le coût de sécurisation de ces algorithmes vis-à-vis des attaques que nous avons introduites. Ces résultats remettent en question les avantages en termes de performances dont ChaCha20 bénéficie vis-à-vis des algorithmes symétriques traditionnels une fois sécurisé contre les attaques physiques.

1.4.3 Structure du manuscrit

La première partie de ce document, dédiée à l'état de l'art, est constituée de trois chapitres. Le premier introduit plus en détails les mécanismes de sécurité actuellement employés dans l'IdO en s'intéressant particulièrement à deux cas d'étude : le protocole de communication LoRaWAN et les réseaux Bluetooth mesh. Ce chapitre a non seulement pour but de donner un aperçu des techniques cryptographiques actuellement utilisées dans le cadre de l'IdO, mais également de souligner que l'intérêt d'intégrer des algorithmes cryptographiques légers varie selon les réseaux et les cas d'usage qu'ils définissent. Le second chapitre introduit la notion de cryptographie légère en traitant aussi bien les aspects d'implémentation des primitives cryptographiques que leurs définitions mathématiques. En particulier, les trois algorithmes sur lesquels porte cette étude, ChaCha, ACORN et ASCON, sont présentés en détails. Le troisième et dernier chapitre de cette partie dresse un état de l'art des attaques par observation, et plus particulièrement celles basées sur la consommation de courant et/ou les émanations électromagnétiques du composant. Ce chapitre a pour but de présenter les méthodes d'attaque ainsi que les contre-mesures considérées lors de nos travaux d'étude.

La deuxième partie, dédiée aux travaux d'implémentations et à la définition de nouvelles attaques, est également constituée de trois chapitres. Le premier présente nos travaux sur la primitive cryptographique ChaCha. Plus précisément, nous analysons sa capacité à résister aux attaques par observation autres que celles par temps de calcul, auxquelles il est intrinsèquement immunisé. Nous aboutissons à une attaque qui, expérimentations pratiques à l'appui, permet d'extraire la clef de chiffrement en exploitant les rayonnements électromagnétiques d'un microcontrôleur tel que ceux utilisés dans l'IdO. Nous évaluons ensuite le surcoût induit par l'intégration d'une contre-mesure permettant de prévenir cette attaque, et soulignons que son impact sur les performances est particulièrement élevé dans le cas de ChaCha, à cause de la structure mathématique sur laquelle il repose. Le second chapitre résume nos contributions à la compétition cryptographique CAESAR. Elles consistent en la définition de la première attaque par

observation à l'encontre d'ACORN et dresse un comparatif de performances entre ce dernier et ASCON, les deux algorithmes de la compétition à destination des systèmes embarqués contraints. Finalement, le dernier chapitre synthétise l'intégralité de nos résultats et expose les questions sous-jacentes ainsi que les perspectives d'approfondissement pour tenter d'y répondre.

Publications

[AFM18b] *Practical Algebraic Side-Channel Attacks Against ACORN* par Alexandre ADOMNICAI, Jacques FOURNIER et Laurent MASSON. Information Security and Cryptology – ICISC 2018, Seoul, Corée du Sud. https ://link.springer.com/chapter/10.1007/978-3-030-12146-4_20

[AFM18c] *Masking the Lightweight Authenticated Ciphers ACORN and Ascon in Software* avec Jacques FOURNIER et Laurent MASSON. BalkanCryptSec 2018, Iași, Roumanie. https ://eprint.iacr.org/2018/708.pdf

[AFM18a] *Hardware Security Threats Against Bluetooth Mesh Networks* par Alexandre ADOMNICAI, Jacques FOURNIER et Laurent MASSON. IEEE CNS 2018, Beijing, Chine. https ://ieeexplore.ieee.org/document/8433184

[AFM17] *Bricklayer Attack : A Side-Channel Analysis on the ChaCha Quarter Round* par Alexandre ADOMNICAI, Jacques FOURNIER et Laurent MASSON. IndoCrypt 2017, Chennai, Inde. https ://link.springer.com/chapter/10.1007/978-3-319-71667-1_4

[ALC+16] *On the importance of considering physical attacks when implementing lightweight cryptography* par Alexandre ADOMNICAI, Benjamin LAC, Anne CANTEAUT, Jacques FOURNIER, Laurent MASSON, Renaud SIRDEY et Assia TRIA. NIST LWC Workshop 2016, Gaithersburg, États-Unis. https ://hal.inria.fr/cea-01436006v1

Première partie

État de l’art

Chapitre 2

Sécurité pour l'internet des objets

« *Computer security is now everything security.* »

Bruce Schneier

Sommaire

2.1 Défis technologiques

2.1.1 Hétérogénéité des plateformes

L'IdO est caractérisé par un panel hétérogène d'appareils embarqués, allant de simples capteurs jusqu'à des composantes critiques de systèmes sophistiqués. Contrairement aux ordinateurs classiques, ces objets ont généralement pour rôle d'exécuter une tâche très spécifique et reposent sur des systèmes d'exploitation légers (*e.g.*, RIOT [BHG+13]) qui bénéficient d'une consommation énergétique et d'une utilisation mémoire minimales. Ces systèmes d'exploitation ont été construits dans le but d'adresser le large éventail de microcontrôleurs de l'IdO.

Durant la dernière décennie, les microcontrôleurs 32-bit ont vu leurs prix diminuer et leurs performances augmenter. Ces progrès en ont fait la plateforme de référence pour l'IdO [FK14, Res17]. Cependant, la faible consommation énergétique des microcontrôleurs 8-bit et 16-bit permettent à ces derniers de rester compétitifs pour les cas d'usages qui nécessitent peu de puissance de calcul et un fonctionnement sur batterie.

En plus des critères susmentionnés, la sélection d'une plateforme pour un objet connecté doit aussi prendre en compte le niveau de sécurité désiré. Idéalement, les fonctionnalités de sécurité devraient faire partie intégrante de la conception du circuit, comme c'est le cas pour la solution ARM Cortex-M35P [ARM18]. La réalité est telle que la plupart de ces solutions sont onéreuses et adressent principalement les plateformes 32-bit à ce jour. Une alternative est d'implémenter les fonctionnalités de sécurité en logiciel lorsqu'il est impossible d'ajouter, pour des raisons techniques et/ou économiques, un composant externe de type Secure Element. Au delà d'une éventuelle dégradation des performances, une implémentation logicielle occupe une partie plus ou moins importante de l'espace mémoire à disposition selon sa taille de code. Par conséquent, l'intégration de primitives cryptographiques constitue un premier défi qu'il est nécessaire d'adresser afin de construire des protocoles de sécurité en aval.

2.1.2 Implémentations cryptographiques embarquées

2.1.2.1 Le cas particulier de l'AES

Bien que l'AES n'ait pas été initialement conçu pour s'exécuter sur des plateformes embarquées, plusieurs travaux ont démontré qu'il est possible d'intégrer cet algorithme aux objets les plus contraints tels que les étiquettes RFID (radio frequency identification) [FSZW, HC14]. L'algorithme de chiffrement AES opère sur un état interne de 16 octets qui consiste en une matrice 4×4, où l'ordonnancement des octets est décrit en figure 2.1, et repose principalement sur une fonction de tour. Cette fonction de tour, illustrée en figure 2.2, se décompose en quatre sous-fonctions :

- Add Round Key (ARK) : Ajoute une clef de tour (dérivée de la clef initiale) à l'état interne courant.
- Sub Bytes (SB) : Assure la notion de confusion telle que définie par Shannon. Chaque octet est traité indépendamment des autres. En pratique, il est courant d'utiliser

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

FIGURE 2.1 – Ordonnancement des octets dans l'état interne de l'AES

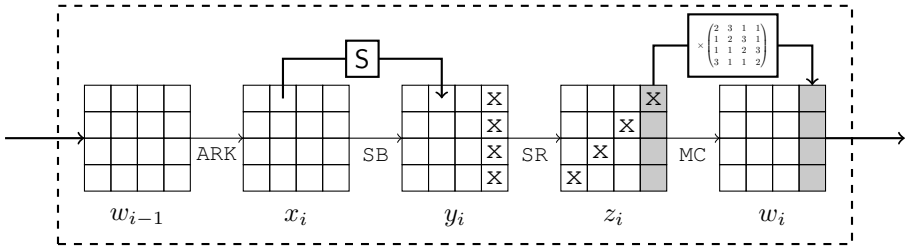


FIGURE 2.2 – Fonction de tour de l'AES

une table de précalcul.

- Shift Rows (SR) : Décale chaque octet de la ligne j de j crans vers la gauche.
- Mix Columns (MC) : Assure la notion de diffusion (avec SR) telle que définie par Shannon. Chaque colonne est multipliée par la matrice $\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$.

Le nombre de tours ainsi que la dérivation des clefs de tour dépendent de la taille de la clef utilisée pour l'AES, la version 128-bit étant la plus compacte.

De nombreuses études ont été réalisées dans le but d'optimiser au maximum les implémentations de l'AES aussi bien en matériel [BJM+14, BBR16, UMHA16] qu'en logiciel [OBSC10, SS17]. Les résultats démontrent que l'AES-128 permet d'atteindre de bonnes performances sur microcontrôleur 8-bit, mais uniquement à condition d'utiliser une table de précalcul pour la sous-fonction SB. En effet, calculer cette sous-fonction de manière dynamique induit une surcharge importante car elle intègre des inversions dans le corps $\mathbb{F}_{256}[X]/(X^8+X^4+X^3+X+1)$ qui nécessitent beaucoup d'opérations élémentaires. Par conséquent, l'AES ne permet pas d'aboutir à une implémentation qui soit compacte et efficace à la fois. De plus, cet algorithme n'est pas optimisé pour l'intégration de contre-mesures face aux attaques par observation. À titre d'exemple, une fois sécurisé contre les fuites par temps de calcul et par consommation, on observe un temps d'exécution 13 fois supérieur à la version non sécurisée sur ARM Cortex-M4 [SS17].

2.1.2.2 Cryptographie asymétrique

Si l'intégration d'algorithmes cryptographiques symétriques au sein de plateformes embarquées contraintes n'est pas toujours aisée, il en va de même pour les variantes asymétriques. En effet, la cryptographie asymétrique est d'autant plus difficile à implémenter efficacement du fait qu'elle nécessite des opérations mathématiques complexes sur de grands entiers. Il était initialement admis que ces primitives étaient trop gourmandes en termes de calculs pour être exécutées sur les plateformes les plus minimalistes telles que les étiquettes RFID (*e.g.*, *"Unfortunately asymmetric cryptography is too heavy to be implemented on a tag"* [ADO06]) jusqu'à ce que des travaux affirment le contraire [BGK+07, ALOW15]. Cependant, les implémentations logicielles de ces algorithmes restent extrêmement lourdes pour de nombreuses plateformes, notamment celles qui ne disposent pas de multiplieur matériel telles que le microcontrôleur 16-bit MSP430 [Tex06], où une multiplication modulaire sur de grands entiers peut se traduire par des milliers de cycles d'horloge selon la taille des opérandes.

Le défi que représente l'implémentation d'une telle primitive cryptographique n'est en réalité que la partie visible de l'iceberg. En effet, la cryptographie asymétrique nécessite l'utilisation de *certificats électroniques* qui attestent que la clef publique appartient bien à l'interlocuteur espéré, afin d'éviter les attaques de type *Man-In-The-Middle* (MITM). Plus généralement, la gestion des clefs pour l'IdO soulève de nombreuses problématiques.

2.1.3 Gestion des clefs

La norme X.509 [CSF+08] est un standard pour les infrastructures à clefs publiques traditionnelles. Elle définit de nombreux paramètres comme par exemple le format des certificats électroniques. Cette norme est notamment utilisée par le protocole Transport Layer Security (TLS) [Res18], partie intégrante du protocole HyperText Transfer Protocol Secure (HTTPS) [Res00] qui permet d'accéder à des pages web de manière sécurisée. Plus précisément, lorsqu'un internaute accède à un site web, les deux entités s'échangent leurs clefs publiques afin d'en dériver une clef symétrique qui servira à chiffrer et authentifier les futures requêtes et réponses de chacun. En théorie, l'étape d'authentification au travers de certificats est optionnelle. Dans le cas du protocole HTTPS, seuls les serveurs web sont en possession d'un certificat X.509 afin que le client puisse s'assurer qu'il établit une session de sécurité avec le site web qu'il visite. La réciproque est rarement utile car le principe d'un site web est d'être accessible à tous. Dans le cas de l'IdO, où chaque objet a potentiellement besoin d'authentifier ses homologues avec lesquels il communique, l'utilisation de certificats X.509 est beaucoup plus compliquée et soulève de nombreux défis.

De nombreux réseaux ne supportent que des petites tailles de paquets (*e.g.*, 12 octets pour Sigfox) qui sont nettement inférieures à celle d'un certificat X.509 (*e.g.*, ≈ 200 octets pour une clef ECC de 256 bits), entraînant une surconsommation de la bande passante. La tâche la plus problématique concerne la vérification de la validité du certificat. Un certificat X.509 intègre deux dates pour définir sa durée de validité : "`not before`" et "`not after`". L'objet en charge de la vérification du certificat doit par conséquent posséder ou être connecté à une horloge temps réel de confiance, ce qui n'est pas toujours le cas. En plus des dates de validité, il est également nécessaire de vérifier si aucun élément de la chaîne de certification n'a été révoqué. La révocation d'un certificat désigne l'action de rendre ce dernier ineffectif avant sa date d'expiration, en raison d'une faille de sécurité ou de la mise à jour d'une paire de clefs dans la chaîne de certification. La vérification d'une potentielle révocation pour un certificat nécessite d'intérroger un serveur externe, entraînant une surcharge supplémentaire.

Les difficultés techniques énoncées ci-dessus entraînent des mauvaises pratiques concernant la gestion de clefs cryptographiques asymétriques de la part de nombreux vendeurs d'objets connectés. À titre d'exemple, l'analyse de 32 000 firmwares d'objets connectés a permis de retrouver 41 certificats auto-signés ainsi que les clefs privées associées, qui étaient utilisés par approximativement 35 000 objets au total [CZFB14]. Par conséquent, comme en témoignent les deux cas d'études présentés ci-dessous, la sécurité de nombreux réseaux de l'IdO repose principalement sur de la cryptographie symétrique.

2.2 LoRa

2.2.1 Introduction

LoRaWAN définit le protocole de communication ainsi que l'architecture système du réseau tandis que LoRa définit la couche physique qui assure la connectivité longue distance. La technologie de modulation des ondes radios sur laquelle repose LoRa a été créée par la start-up grenobloise Cycleo, fondée en 2009 avant d'être rachetée par Semtech en 2012 pour 21 millions de dollars. Afin de promouvir sa technologie, Semtech crée l'alliance LoRa en 2015 qui rassemble plus de 500 entreprises. LoRaWAN est le fruit d'un travail collaboratif au sein de ce consortium, ce qui a pour effet de rassurer les utilisateurs, étant un gage de transparence et de pérennité.

2.2.2 Fonctionnement

2.2.2.1 Opérateurs et réseaux

Contrairement aux réseaux Sigfox, pour lesquels la société toulousainne du même nom contrôle l'intégralité de la chaîne de valeur (du protocole de communication jusqu'au service de réseau), LoRaWAN compte autant de réseaux que d'opérateurs. En France, Bouygues Telecom et Orange qui font tous deux partie de l'alliance LoRa, sont les principaux opérateurs français et couvrent chacun plus de 30 000 communes, soit 95% de la population.

De plus, LoRaWAN étant une solution open-source qui opère sur des bandes de fréquences libres et gratuites, il est possible de mettre en place des réseaux locaux sans passer par un opérateur, à l'instar des réseaux Wi-Fi. Cela nécessite néanmoins l'installation de passerelles propriétaires (modèles basiques pour quelques centaines d'euros) et le développement d'un serveur applicatif.

2.2.2.2 Architecture

Au sein d'un réseau LoRaWAN, on peut distinguer cinq catégories d'éléments décrits ci-dessous.

- **Les nœuds** : il s'agit d'objets connectés équipés de capteurs et d'une antenne LoRa afin d'émettre et de recevoir des informations.
- **Les passerelles** : elles ont pour seul but de transférer les informations provenant des nœuds vers le serveur réseau (et vice-versa). Elles ont une portée de 2 km en zones urbaines et 15 km en zones rurales.
- **Le serveur réseau** : au centre de l'architecture, il a pour rôle d'assurer la sécurité (authentification des paquets, vérifications des métadonnées, ...) et de transférer les données applicatives contenues dans les paquets aux serveurs applicatifs.
- **Les serveurs applicatifs** : ils sont destinés à analyser les informations recueillies par les nœuds et à communiquer les résultats aux utilisateurs.
- **Les serveurs d'activation** : ils ont pour rôle d'appairer les nœuds avec le serveur réseau et les serveurs applicatifs dans le cas des activations *Over-The-Air* (OTA).

Les réseaux LoRaWAN permettent une communication bidirectionnelle entre les nœuds et les serveurs applicatifs auxquels ils sont associés. Pour ce faire, ils s'appuient sur une topologie en étoile où les nœuds communiquent des informations aux passerelles. Ces dernières les transmettent ensuite au serveur réseau qui les transfère à son tour au serveur applicatif.

2.2.2.3 Performances

La technologie LoRa, qui opère sur les bandes de fréquence autour de 868 MHz en Europe, utilise une modulation à étalement de spectre. Plus le facteur d'étalement est grand, plus la portée augmente au détriment du débit, car ce dernier est transmis sur une plus longue période. LoRa supporte six différents facteurs d'étalements (de 7 à 12). La taille maximale des paquets qui transitent sur le réseau dépend directement de ce paramètre. Le tableau 2.1 indique, pour chaque facteur d'étalement, le temps nécessaire pour transmettre un paquet de taille maximale.

Les bandes de fréquences utilisées par les LPWAN sont libres et gratuites mais sont soumises à des règles. En Europe, l'organisme en charge de ces règlementations, l'Institut Européen des Normes des Télécommunications, impose un coefficient d'utilisation limite de 1% [reg14]. En d'autres termes, un nœud ne peut émettre que 1% du temps. Ce

coefficient a pour rôle de limiter le temps d'utilisation de la bande passante afin d'éviter qu'un émetteur ne la monopolise et entraîne une saturation du réseau.

TABLEAU 2.1 – Temps de transmission d'un paquet de taille maximale (bande passante de 125 kHz)

Facteur d'étalement	Taille maximale des paquets (en octets)	Temps de transmission (en ms)	Attente avant retransmission (en ms)
12	59	2466	244134
11	59	1314	130086
10	59	616	60984
9	123	615	60885
8	230	615	60885
7	230	348	34452

2.2.3 Sécurité

2.2.3.1 Vue d'ensemble

Au sein d'un réseau LoRaWAN, on distingue deux couches de sécurité : une au niveau réseau et une autre au niveau applicatif. Les propriétés d'authenticité, intégrité et confidentialité des paquets reposent toutes sur l'algorithme de chiffrement par bloc AES en version 128-bit.

LoRaWAN v1.0 spécifie qu'un nœud doit posséder deux clefs symétriques de 128 bits, applées *Network Session Key* (`NwkSKey`) et *Application Session Key* (`AppSKey`), chacune respectivement en charge de la sécurité au niveau réseau et applicatif. Plus précisément, les clefs `NwkSKey` assurent l'intégrité et l'authenticité des paquets réseaux tandis que les clefs `AppSKey` assurent la confidentialité des données applicatives. Depuis la publication de la version LoRaWAN v1.1, la clef `NwkSKey` est en réalité divisée en plusieurs clefs, chacune ayant une tâche spécifique. Par souci de simplicité, `NwKSKey` désigne l'ensemble de ces clefs dans la suite de ce chapitre.

L'utilisation de deux clefs distinctes marque une séparation nette entre les différents types de données. Une clef `NwkSKey` est uniquement partagée entre un nœud et le serveur réseau alors qu'une clef `AppSKey` est uniquement partagée entre un nœud et le serveur applicatif. Ainsi, les opérateurs qui entretiennent l'infrastructure des réseaux ne sont pas en mesure de lire les données applicatives chiffrées à l'aide des `AppSKey`. Chaque nœud possède sa propre paire de clefs `NwkSKey` et `AppSKey` afin que la compromission d'un nœud n'affecte pas les autres. La figure 2.3 illustre le rôle des clefs au sein du protocole LoRaWAN.

2.2.3.2 Confidentialité

En premier lieu, les données applicatives sont chiffrées à l'aide de la clef `AppSKey` via l'algorithme AES utilisé avec le mode d'opération compteur (CTR) [Dwo01]. Le mode CTR permet de transformer un chiffrement par bloc en un chiffrement par flot. Le principe est de générer une sortie pseudo-aléatoire en chiffrant les valeurs successives d'un compteur. Le texte chiffré est obtenu en calculant le XOR de cette sortie pseudo-aléatoire avec le texte clair, comme indiqué en figure 2.4a. Le mode CTR est symétrique dans le sens où les procédés de chiffrement et de déchiffrement sont identiques. Cette propriété est d'un grand intérêt pour les systèmes embarqués car il n'est pas nécessaire d'avoir recours à la fonction de déchiffrement de l'AES ce qui permet d'aboutir à une implémentation plus

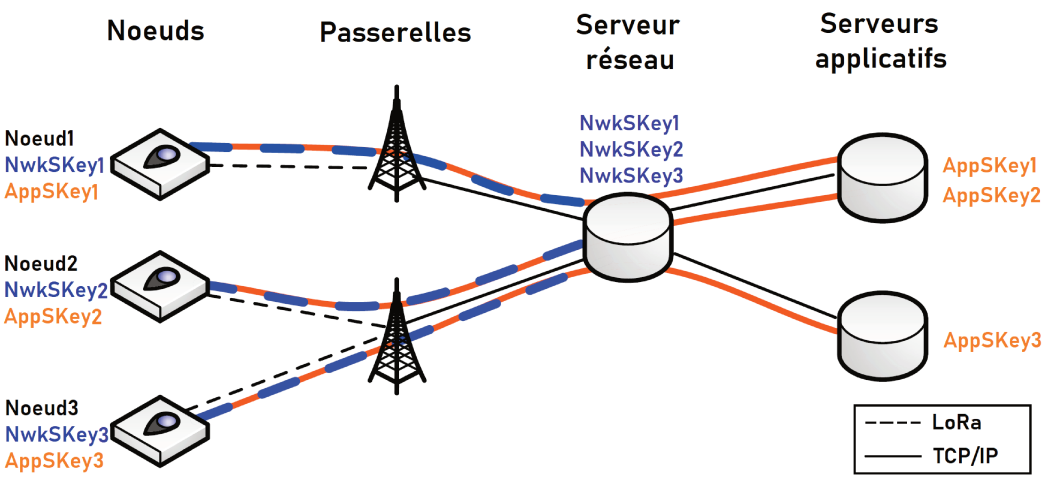
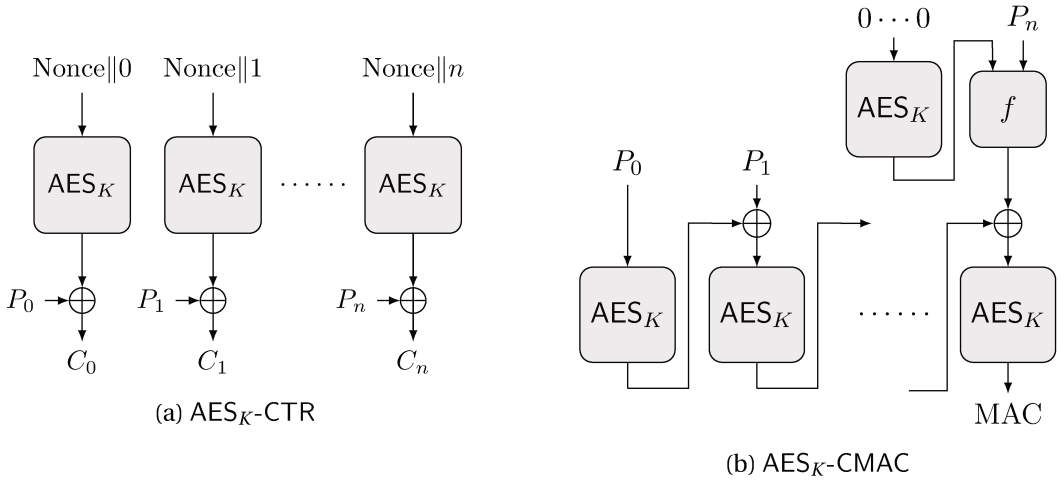


FIGURE 2.3 – Rôle des clefs au sein du protocole LoRaWAN

compacte. Le nombre d'éxécution de l'AES, au minimum un, dépend directement de la longueur du message à chiffrer.

2.2.3.3 Intégrité et authenticité

Une fois les données applicatives chiffrées, un en-tête contenant des métadonnées (*e.g.*, l'adresse de l'émetteur utilisée pour communiquer sur le réseau notée `DevAddr`) est ajouté. Le résultat de cette concaténation est ensuite protégé en intégrité et authenticité à l'aide de la clef `NwkSKey` et de l'algorithme AES combiné avec le mode CMAC [IK03]. Le mode CMAC est une extension du mode CBC-MAC obtenue en ajoutant un calcul sur le dernier bloc du message à authentifier, afin de supporter des messages de n'importe quelle longueur. Il est illustré en figure 2.4b où f, qui dépend des bits de poids faibles de ses entrées, désigne la dérivation du dernier bloc.



(a) AES$_K$-CTR

(b) AES$_K$-CMAC

FIGURE 2.4 – Modes d'opération CTR et CMAC

La figure 2.5 résume les étapes nécessaires à la sécurisation de données applicatives. Il est à noter que ce processus requiert au minimum trois exécution de l'AES, une pour le mode CTR et deux autres pour le mode CMAC.

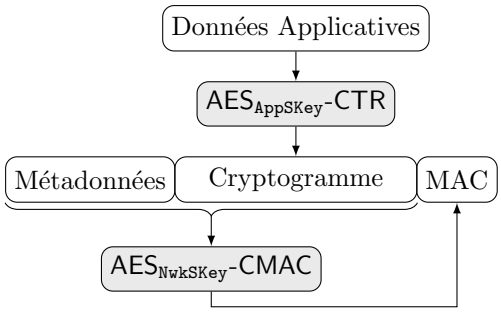


FIGURE 2.5 – Sécurisation de données applicatives au sein d'un réseau LoRa

2.2.3.4 Distribution des clefs

Bien que la gestion de clefs symétriques s'affranchisse des complications induites par les certificats électroniques, elle n'en demeure pas moins complexe. Dans le cas de LoRaWAN, il existe deux procédures de distribution de clefs : les activations OTA ou par personnalisation.

Activation par Personnalisation L'activation par personnalisation consiste à programmer les clefs `NwkSKey` et `AppSKey` ainsi que l'adresse de l'objet au sein du réseau `DevAddr` dans les objets connectés avant qu'ils ne soient mis en service. Les clefs en question étant symétriques, elles doivent être respectivement communiquées au serveur réseau et au serveur applicatif de manière sûre, ce qui complique le processus de déploiement en pratique. De plus, ces clefs sont statiques et ne constituent pas des clefs de session au sens propre du terme.

Contrairement aux cartes SIM et aux modems Internet, un objet connecté n'est pas nécéssairement associé à un opérateur dès son processus de fabrication afin de laisser une certaine flexibilité à l'utilisateur. Dans ce cas, l'activation par personnalisation n'est pas envisageable et c'est pourquoi une seconde option a été envisagée par l'alliance LoRa.

Activation OTA L'activation OTA contourne les problématiques d'activation par personnalisation en faisant intervenir un mécanisme de dérivation de clef. Le principe est de programmer deux clefs maîtres par nœud, notées `AppKey` et `NwkKey`, à partir desquelles seront dérivées les clefs de session `AppSKey` et `NwkSKey`. Deux identifiants uniques sont également programmés, `DevEUI` et `AppEUI`, afin d'identifier le nœud et le serveur applicatif associé.

Dès lors qu'un nœud destiné à une activation OTA est mis en service, il envoie une requête d'activation au serveur réseau. Les clefs maîtres étant uniquement connues du serveur d'activation, le serveur réseau se contente de lui transfèrer la requête. Le serveur d'activation vérifie si la requête est légitime et, le cas échéant, en accuse la réception et génère les clefs de session `NwkSKey` et `AppSKey` ainsi que l'adresse réseau du nœud `DevAddr`. Les paramètres `NwkSKey` et `DevAddr` sont ensuite transmis au serveur réseau tandis que `AppSKey` est transmis au serveur applicatif. Une fois l'accusé de réception reçu, le nœud dérive à son tour les clefs de session en interne. Le processus de dérivation utilise l'AES comme fonction pseudoaléatoire et est décrit par l'équation 2.1, où `DevNonce` est généré par le nœud et inclus dans la requête d'activation tandis que `JoinNonce` est généré par le serveur d'activation et inclus dans l'accusé de réception. Contrairement à l'activation par personnalisation, la variante OTA offre la possibilité de renouveller les

clefs de session d'un nœud en rejouant le protocole d'activation.

$$\begin{aligned} \texttt{AppSKey} &= \mathsf{AES}_{\texttt{AppKey}}(\texttt{0x02} \,\|\, \texttt{JoinNonce} \,\|\, \texttt{NetID} \,\|\, \texttt{DevNonce}) \\ \texttt{NwkSKey} &= \mathsf{AES}_{\texttt{NwkKey}}(\texttt{0x01} \,\|\, \texttt{JoinNonce} \,\|\, \texttt{NetID} \,\|\, \texttt{DevNonce}) \end{aligned} \tag{2.1}$$

À noter qu'il est essentiel que le serveur d'activation soit isolé du serveur réseau afin d'assurer que ce dernier n'ait pas accès à la clef `AppSKey` et donc aux données applicatives. Cela introduit donc un nouvel acteur au sein de l'architecture LoRa. Par exemple, la société française Actility propose des solutions de serveur d'activation aux différents opérateurs LoRaWAN à travers le monde.

Si LoRa permet de couvrir de nombreux cas d'usage de l'IdO, sa topologie en étoile n'est pas nécessairement adaptée à tous les scénarios puisqu'un nœud est uniquement capable de communiquer avec le serveur. Ainsi, pour communiquer des informations à un autre nœud, l'émetteur doit d'abord transmettre le message au serveur qui le relaiera ensuite au destinataire. D'autres technologies sont développées pour mieux adrésser les cas d'usage où les objets communiquent fréquemment les uns avec les autres, comme la technologie Bluetooth mesh.

2.3 Réseaux Bluetooth maillés

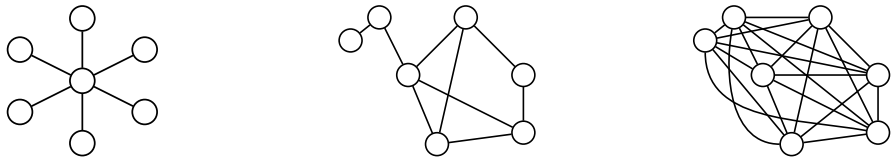
2.3.1 Introduction

La technologie Bluetooth a été introduite en 1994 par Jaap Haartsen, alors ingénieur chez Ericsson. Son but était de proposer une alternative sans fil aux câbles de données RS-232 qui, depuis les années 1960, constituaient la norme pour connecter des périphériques à un ordinateur. Jaap Haarsten a su transcender ce standard en utilisant des ondes radio ultra haute fréquence (UHF) dans la bande libre Industrielle, Scientifique et Médicale (ISM) comprise entre 2,4 et 2,45 GHz, afin d'envoyer des paquets de données sur des distances de quelques mètres. L'appellation Bluetooth est une référence au roi Harald Blatland, qui signifie dent bleue en danois, qui réunifia les royaumes du Danemark, de Norvège et de Suède au X$^{\text{ème}}$ siècle. En 1998, Ericsson ainsi que Nokia, Intel, Toshiba et IBM fondèrent le groupe de travail Bluetooth Special Interest Group (SIG) afin de promouvoir et standardiser la technologie Bluetooth.

En 2001, Nokia a commencé à développer une technologie sans fil basée sur Bluetooth dans le but de minimiser la consommation énergétique afin d'être compatible avec de nouveaux types d'équipements tels que des montres ou des capteurs embarqués. Les résultats ont d'abord été publiés en 2004 sous le nom de Bluetooth Low End Extension [HLK04]. En 2010, cette technologie est intégrée dans la version 4.0 de la spécification Bluetooth sous la marque Bluetooth Smart, et est couramment appelée Bluetooth Low Energy (BLE).

Dans le cadre d'un réseau BLE, chaque nœud communique par le biais d'un canal indépendant avec un dispositif de supervision central (*e.g.*, un smartphone), formant ainsi une topologie de réseau en étoile. Cette architecture n'est pas adaptée à certains types d'applications. Par exemple, dans une grande maison, un radiateur connecté que l'on souhaiterait activer à distance pourrait être hors de portée du dispositif de supervision. Autre complication, les réseaux BLE peuvent uniquement gérer un nombre limité de connexions simultanées. Dans le cas d'une application industrielle gérant un grand nombre de nœuds, il serait alors impossible de contrôler ces derniers à l'aide d'une seule commande, ce qui induirait une latence. Un moyen de répondre à ces problématiques est l'utilisation d'une topologie maillée, tel qu'illustré par la figure 2.6b. Contrairement à un

réseau en étoile, un réseau maillé permet de contrôler simultanément plusieurs nœuds mais également de surmonter les limitations de portée.



(a) Topologie en étoile (b) Topologie maillée partielle (c) Topologie maillée complète

FIGURE 2.6 – Différentes topologies de réseau

De nombreuses solutions propriétaires furent développées afin d'adapter la topologie maillée aux réseaux BLE, aussi bien par des académiques que par des industriels [DG17]. Les intérêts à l'égard des réseaux BLE maillés ont amené le Bluetooth SIG à proposer un standard pour cette technologie, dont la première version 1.0 fut publiée en juillet 2017 sous le nom de Bluetooth mesh [SIG17].

2.3.2 Fonctionnement

2.3.2.1 Concept

La version 1.0 du Bluetooth mesh suit un modèle "*publish/subscribe*" où les messages sont classés par catégories auxquelles les destinataires s'abonnent. Un nœud au sein d'un réseau Bluetooth peut s'abonner à plusieurs adresses, tandis qu'il ne peut émettre des messages qu'à une seule adresse. Il existe deux types d'adresses :

- Les adresses unicast, qui sont propres à chaque nœud et qui permettent d'identifier ces derniers.
- Les adresses de groupe, qui identifient un groupe de nœud au sein du réseau.

Chaque nœud inclut automatiquement sa propre adresse unicast dans sa liste d'abonnements. Dans le cas où un nœud veut rejoindre un groupe spécifique, il doit ajouter l'adresse de groupe qui correspond à sa liste d'abonnements.

2.3.2.2 Communication

Comme le Bluetooth classique, la variante Low Energy opère sur la bande ISM comprise entre 2,4 et 2,45 GHz. Plus précisément, le BLE utilise 40 canaux de 2402 à 2480 MHz, espacés les uns des autres de 2 MHz. Parmi ces 40 canaux, 3 sont classifiés comme canaux d'annonce (*advertisement channels*) et 37 autres comme canaux de connection (*connection channels*). Classifier les canaux en deux catégories permet à la technologie BLE de disposer deux modes de communication entre les nœuds. En mode annonce, un nœud émet des paquets sur les trois canaux d'annonce et ces paquets sont accessibles à tout autre nœud à l'écoute sur ces trois canaux. En mode connecté, un lien est établi entre deux nœuds et eux seuls peuvent alors communiquer sur des canaux de connection négociés.

Les réseaux Bluetooth maillés reposent sur la technologie BLE et utilisent le mode annonce de communication. Contrairement au BLE, ils utilisent un mécanisme de routage par inondation, qui consiste à ce que chaque nœud du réseau répète à ses voisins directs les messages qu'il reçoit. Ce mécanisme de routage augmente la fiabilité du réseau dans le sens où il peut exister différents chemins qui connectent l'émetteur au destinataire. En revanche, beaucoup de bande passante est gâchée pour envoyer un paquet en plusieurs exemplaires. Afin de remédier à cet inconvénient, seuls certains nœuds, appelés

relayeurs, sont en charge d'appliquer le routage par inondation. De plus, chaque nœud doit maintenir un mécanisme de cache qui contient les derniers messages reçus/relayés et chaque message inclus un champs *Time-to-Live* (`TTL`) qui est décrémenté à chaque retransmission. Ainsi, un message ne sera relayé que si le nœud est un relayeur et si le champs `TTL` est supérieur à 1.

2.3.2.3 Performances

Le temps de transmission d'un paquet au sein d'un réseau maillé dépend, outre le débit du réseau et la taille du paquet, du nombre de saut qu'il doit effectuer avant d'atteindre son destinataire. Il est donc plus compliqué de donner des chiffres précis comme cela a été fait précédemment pour la technologie LoRa. Néanmoins, quelques études ont été réalisées pour évaluer les performances des réseaux Bluetooth maillés. Notamment, les auteurs de [BRSH18] donnent des résultats expérimentaux pour le temps de transmission d'un paquet ainsi que son accusé de réception, reportés dans le tableau 2.2, pour différentes configurations de réseau.

TABLEAU 2.2 – Temps de transmission d'un paquet de 40 octets et son accusé de réception à 1 Mbps [BRSH18]

Nombre de sauts	Temps de transmission (en ms)
1	24
2	48
3	70
4	94

À noter que contrairement à LoRa, les bandes de fréquences utilisées par la technologie Bluetooth ne sont pas soumises à une réglementation sur le coefficient d'utilisation limite. Ce dernier est défini au cas par cas pour assurer un bon fonctionnement du réseau et dépend directement de l'application.

2.3.3 Sécurité

2.3.3.1 Différents types de clefs

Tous les messages qui transitent au sein d'un réseau Bluetooth maillé sont chiffrés et authentifiés au niveau réseau et applicatif. On peut distinguer trois différents types de clefs.

- **Les clefs de réseaux** (`NetKey`). Une clef de type `NetKey` est une clef symétrique de 128 bits partagée par tous les nœuds au sein d'un même réseau. La possession d'une telle clef par un nœud définit son appartenance au réseau correspondant. Un réseau peut en réalité être divisé en plusieurs sous-réseaux (jusqu'à 4096 au total), chacun possédant sa propre clef `NetKey`. Chaque réseau et/ou sous-réseau est publiquement identifié par un identifiant de 7 bits noté `NID`.
- **Les clefs applicatives** (`AppKey`). Une clef de type `AppKey` est une clef symétrique de 128 bits partagée par tous les nœuds au sein d'une même application et assure la protection des données applicatives en confidentialité, intégrité et authenticité. Ces clefs sont utilisées pour distinguer les droits d'accès aux différentes applications au sein d'un même réseau.
- **Les clefs objets** (`DevKey`). Chaque nœud possède une unique clef symétrique de 128 bits connue du nœud et du dispositif de supervision central. Ces clefs sont

utilisées afin de configurer les nœud, par exemple donner ou retirer l'accès à de nouveaux réseaux et/ou applications.

2.3.3.2 Gestion des clefs

Pour installer un réseau Bluetooth maillé, un dispositif de supervision central doit en premier lieu générer une clef `NetKey` et allouer un espace d'adresses unicast. Lorsqu'un objet tente de rejoindre le réseau, il avertit le dispositif de supervision qui initie ensuite le mécanisme d'approvisionnement. Ce dernier consiste à l'établissement d'un secret partagé à l'aide de l'algorithme asymétrique ECDH. Le secret partagé obtenu est utilisé pour dériver une clef de session afin de communiquer au nœud les données d'approvisionnement telles que son adresse unicast, sa clef de configuration `DevKey` et la clef du réseau principal `NetKey`. La clef `DevKey` peut ensuite être utilisée pour configurer le nœud en lui communiquant des clefs de sous-réseaux `NetKey` et/ou applicatives `AppKey`. Le dispositif de supervision est responsable de la génération et de la distribution des clefs. Un nœud peut théoriquement posséder jusqu'à 4096 clefs `NetKey` et `AppKey`, respectivement. Le mécanisme d'approvisionnement est illustré par la figure 2.7.

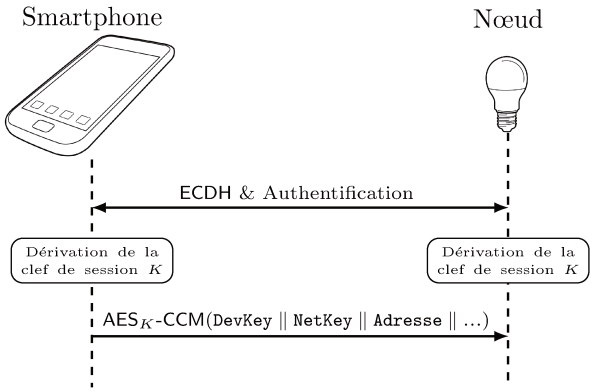


FIGURE 2.7 – Mécanisme d'approvisionnement

Une étape d'authentification a également lieu lors de la négociation du secret partagé avec l'algorithme ECDH afin de prévenir les attaques de type MITM. Plusieurs options d'authentification sont proposées. Toutes ces options sont qualifiées d'hors bande, ce qui signifie que l'authentification a recours à un canal de communication différent de celui en cours d'accès (*i.e.*, les bandes de fréquence Bluetooth) afin de fournir un second facteur d'authentification.

- **Entrée hors bande**. L'objet qui cherche à rejoindre le réseau tire un nombre aléatoire et le communique à l'utilisateur selon ses capacités. Par exemple, si l'objet en question est une ampoule, elle pourrait clignoter un certain nombre de fois. Dans le cas où l'objet dispose d'un écran, un simple affichage suffirait. L'utilisateur est ensuite chargé de communiquer cet aléa au dispositif de supervision.
- **Sortie hors bande**. Le principe est le même que ci-dessus mais les rôles sont inversés : on identifie le dispositif de supervision au lieu de l'objet. Le dispositif de supervision tire un nombre aléatoire, l'affiche sur son écran et l'utilisateur doit ensuite le communiquer à l'objet de manière approprié. Par exemple, si l'objet est un interrupteur, l'utilisateur pourrait l'activer et le désactiver un certain nombre de fois.
- **Hors bande statique**. Cela consiste à pré-programmer dans les objets et le dispositif de supervision des valeurs statiques secrètes qui seront utilisées pour authentifier les deux parties.

Outre les trois méthodes présentées ci-dessus, il est également possible de ne pas intégrer d'authentification, ouvrant la porte aux attaques de type MITM démontrées en pratique à maintes reprises [MPJ11, Mel18]. Le fait que l'authentification s'effectue hors bande et non au travers de certificats électroniques laisse une plus grande liberté pour la gestion des clefs asymétriques. La clef publique n'a pas besoin d'être signée par une autorité de certification et peut donc être générée à la volée ou encore être pré-programmée lors du processus de fabrication.

2.3.3.3 Mécanismes Cryptographiques

La sécurité des messages, aussi bien au niveau réseau qu'applicatif, est assurée à l'aide de l'AES en version 128-bit combiné avec le mode d'opération CCM [WHF03]. Le mode CCM est un mode qui permet d'obtenir, à partir d'un chiffrement par bloc, un chiffrement dit authentifié qui assure à la fois les propriétés de confidentialité, authenticité et intégrité des données. Le mode CCM consiste en la combinaison du mode CTR et du mode CBC-MAC. Les chiffrements authentifiés offrent la possibilité d'authentifier une partie des données sans les chiffrer. Ces données sont appelées données additionnelles et sont généralement des métadonnées non sensibles. Dans le cas des réseaux Bluetooth maillés, l'intégralité du message est chiffrée et par conséquent, aucune donnée additionnelle n'est considérée. L'algorithme AES_K-CCM, illustré par la figure 2.8, utilise une fonction de formattage pour partitionner en plusieurs blocs le message à authentifier. Le premier bloc B_0 contient uniquement le nonce et autres métadonnées (*e.g.*, la longueur du message). Ainsi, la partie d'authentification requiert au minimum deux exécutions de la primitive AES, tout comme le chiffrement, soient quatre exécutions au total.

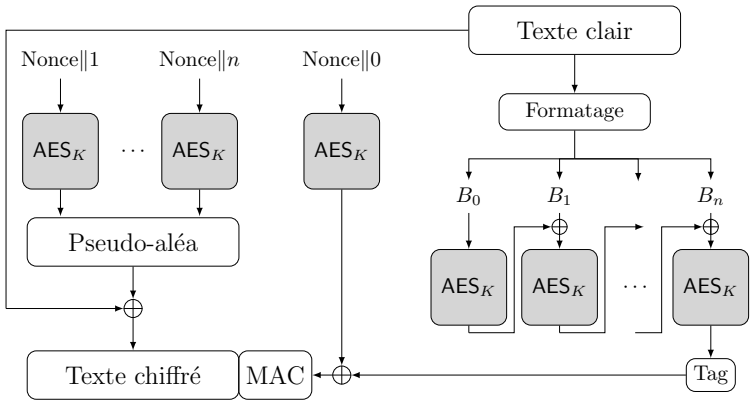


FIGURE 2.8 – AES_K-CCM sans données additionnelles à authentifier

Une fois les données applicatives protégées à l'aide de la clef `AppKey` correspondante, la concaténation du cryptogramme et de l'adresse du destinataire du paquet est à nouveau protégée au niveau réseau. La clef utilisée n'est pas la clef `NetKey` mais une clef dérivée de cette dernière, notée `EncryptionKey`, dont la dérivation est détaillée par l'algorithme 2.1. Ensuite, des métadonnées telles que l'adresse de l'émetteur et le champs `TTL` sont ajoutées au cryptogramme obtenu après la seconde couche de chiffrement. Afin d'empêcher une éventuelle analyse de ces métadonnées qui pourrait révéler des informations sur le réseau, ces métadonnées sont "obfusquées" à l'aide d'une clef notée `PrivacyKey`, elle aussi dérivée de la clef `NetKey`.

Algorithme 2.1 Dérivation des informations d'identification et de sécurité d'une clef `NetKey`

$S \leftarrow \text{AES}_{0\cdots0}\text{-CMAC}(\text{“smk2”})$
$T \leftarrow \text{AES}_{S}\text{-CMAC}(\texttt{NetKey})$
$T_1 \leftarrow \text{AES}_{T}\text{-CMAC}(\texttt{0x0001})$
$T_2 \leftarrow \text{AES}_{T}\text{-CMAC}(T1 \,\|\, \texttt{0x0002})$
$T_3 \leftarrow \text{AES}_{T}\text{-CMAC}(T2 \,\|\, \texttt{0x0003})$
$\texttt{NID} \,\|\, \texttt{EncryptionKey} \,\|\, \texttt{PrivacyKey} \leftarrow (T_1 \,\|\, T_2 \,\|\, T_3) \bmod 2^{263}$

Le terme “obfuscation” est préféré à celui de “chiffrement” car la manière de protéger ces métadonnées n'est pas théoriquement sûre. En effet, l'obfuscation consiste en la construction d'un bloc de 128 bits en concaténant l'indexe du vecteur initial (`IVI`), un paramètre propre à chaque réseau, et les 7 premiers octets du cryptogramme obtenu précédemment. Ce bloc est ensuite chiffré à l'aide de l'AES en utilisant la clef `PrivacyKey` afin de générer une sortie pseudo-aléatoire qui servira à chiffrer les métadonnées à l'aide de l'opérateur XOR. Le fait est que la probabilité de générer la même sortie pseudo-aléatoire pour un même `IVI` est de $\left(2^{56}\right)^{-1}$, ce qui est faible d'un point de vue cryptographique. Cependant, étant donné que le paramètre `IVI` est incrémenté dès lors qu'un nœud du réseau a émis plus de 2^{24} messages, la surface d'attaque est négligeable. Le procédé de sécurisation d'un paquet, illustré par la figure 2.9, requiert au minimum $2 \times 4 + 1 = 9$ exécutions de la primitive AES.

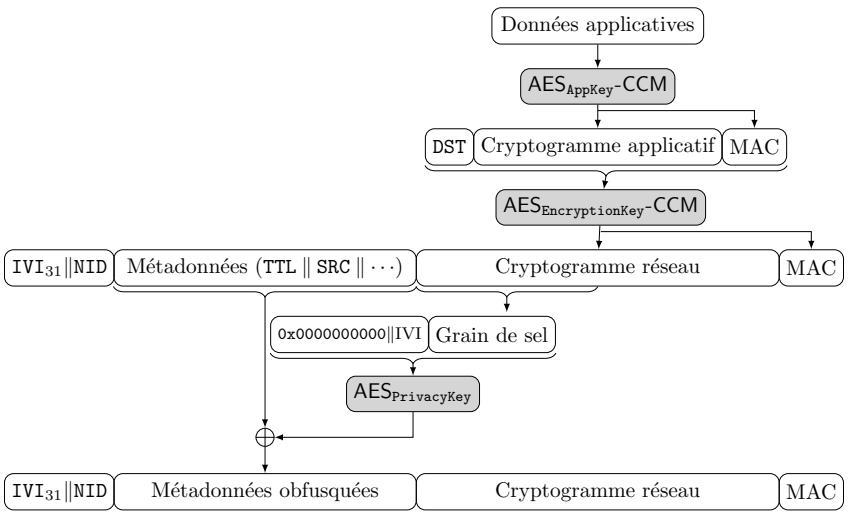


FIGURE 2.9 – Construction d'un paquet à partir de données applicatives au sein d'un réseau Bluetooth maillé

Lorsqu'un nœud reçoit un message en scannant un canal d'annonce, ce dernier doit d'abord vérifier s'il appartient au réseau défini par l'identifiant `NID`. Le cas échéant, il doit d'abord traiter le paquet au niveau réseau à l'aide de la clef `NetKey` associée. Cependant, l'identifiant `NID` est uniquement composé de 7 bits alors qu'un nœud peut théoriquement appartenir jusqu'à 4096 réseaux. Par conséquent, dès lors que le nombre de sous-réseaux est supérieur à $2^7 = 128$, il existe nécessairement au moins deux clefs `NetKey` qui sont associées à un même identifiant `NID`, bien que ce soient des sous-réseaux distincts. De plus, ce cas de figure est possible même pour un plus petit nombre de sous-réseaux étant donné que les identifiants `NID` sont dérivés pseudo-aléatoirement comme indiqué par l'algorithme 2.1. Dans ce cas, le nœud doit tester toutes les différentes clefs afin d'identifier la bonne comme détaillé par l'algorigramme de la figure 2.10.

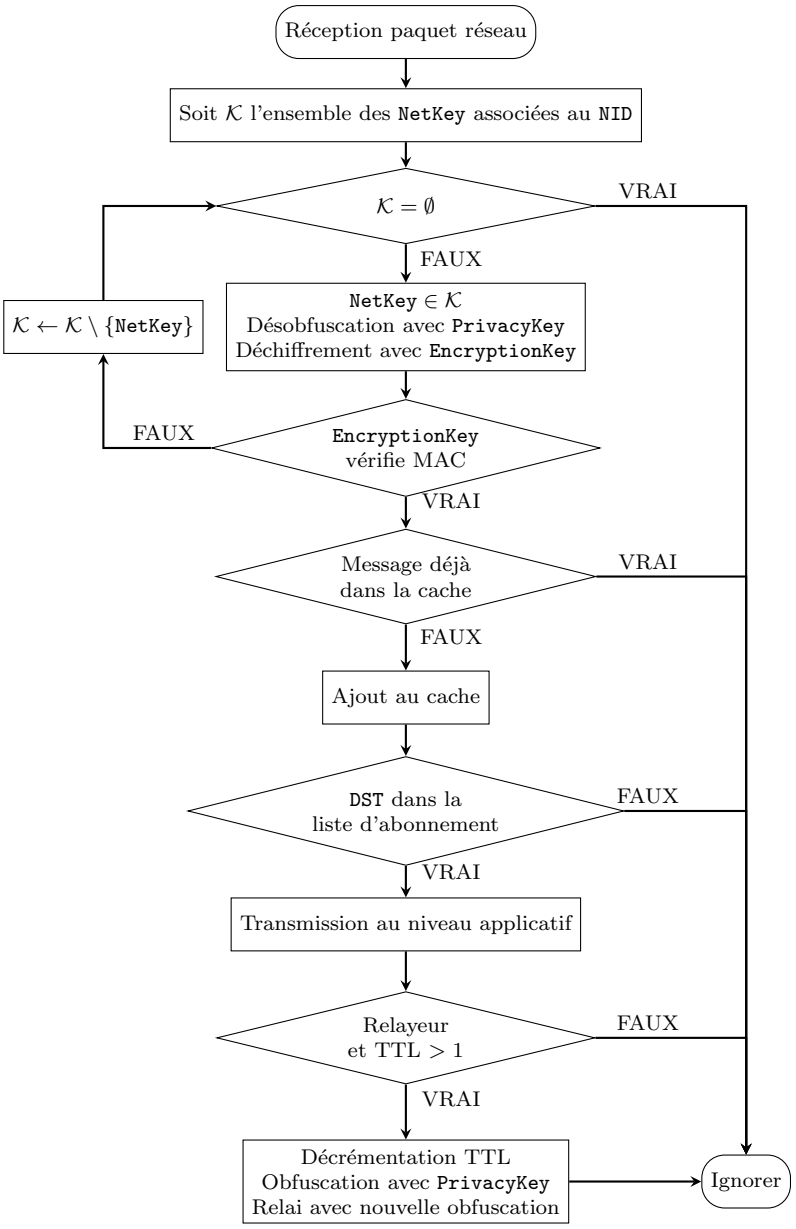


FIGURE 2.10 – Algorigramme de la réception d'un paquet au sein d'un réseau Bluetooth maillé

Si la sécurisation de données applicatives nécessite *a minima* 9 exécutions de l'AES, l'impact de ce dernier sur la réception d'un paquet est également non négligeable. En effet, si une seule clef `NetKey` est associée à l'identifiant `NID`, alors il suffit de désobfusquer les métadonnées et de déchiffrer le paquet au niveau réseau, ce qui requiert au minimum 4 exécutions de l'AES. De plus, si le nœud agit comme relai au sein du réseau maillé, il doit réobfusquer les métadonnées après décrémentation de la valeur `TTL` avant de relayer le paquet. Dans ce cas, le simple traitement d'un paquet réseau requiert *a minima* 5 exécutions de l'AES, voire davantage si plusieurs clefs `NetKey` sont associées à l'identifiant `NID`.

2.4 Synthèse

De nombreux protocoles de télécommunications dédiés à l'IdO, à l'instar des deux présentés dans ce chapitre, intègrent la sécurité au sein même de leur spécification.

Les réseaux Bluetooth maillés et LoRa partagent des similitudes en termes de sécurité, qui sont communes à de nombreux réseaux de l'IdO. Premièrement, on distingue deux couches de sécurité afin d'aboutir à une nette démarcation entre les données réseaux et applicatives. Deuxièmement, la sécurité repose essentiellement, voire exclusivement, sur de la cryptographie symétrique. En l'occurence, l'algorithme de chiffrement par bloc AES fait office de couteau suisse en assurant les propriétés de confidentialité, authenticité et intégrité à l'aide de différents modes d'opération, mais joue également le rôle de fonction pseudo-aléatoire afin de construire des mécanismes de dérivation de clefs.

Bien que non optimal pour les plateformes à faibles ressources, l'AES présente l'avantage d'avoir été étudié en large et en travers par la communauté cryptographique durant de nombreuses années. De ce fait, il bénéficie d'une garantie de sécurité ainsi qu'une panoplie d'implémentations de référence qu'aucun autre chiffrement symétrique ne peut se targuer d'avoir à ce jour. Ce sont pour ces raisons que l'AES est l'algorithme symétrique utilisé pour sécuriser les réseaux de l'IdO. Si les performances de ce dernier sont acceptables dans certains cas, les avantages qu'apporterait un algorithme spécialement conçu pour les plateformes contraintes seraient multiples.

Au premier abord, les réseaux de type LPWAN tels que LoRa n'auraient pas grand intérêt à remplacer l'AES par primitive cryptographique plus performante. En effet, ces technologies privilégient la portée des communications plutôt que le débit. Ainsi, selon la configuration du réseau, l'envoi d'un message peut nécessiter plusieurs secondes et l'attente avant réémission plusieurs minutes comme précisé par le tableau 2.1. Cependant, le gain de temps qui pourrait être acquis au travers d'une primitive cryptographique plus performante permettrait d'économiser de l'énergie au fil du temps, augmentant la durée de vie des objets fonctionnant sur batterie. De plus, une taille de code plus compacte permettrait aux objets les plus contraints d'intégrer des fonctionnalités supplémentaires.

Dans le cas des réseaux Bluetooth maillés, les intérêts sont d'autant plus flagrants étant donné que le simple relai d'un paquet nécessite de nombreux appels à la primitive cryptographique. Afin que cela n'affecte pas les performances réseaux ou la durée de vie des objets, certains fabricants proposent des *System-on-Chip* (SoC) munis d'un coprocesseur cryptographique qui permettent d'atteindre des meilleures performances grâce à des implémentations matérielles dédiées (*e.g.*, le nRF52840 équipé de la TrustZone Cryptocell-310 [Sem18]). Cependant, ces solutions sont onéreuses ce qui peut constituer un frein économique au développement de certaines solutions.

Ces deux exemples dressent un tableau de la situation actuelle. Les enjeux économiques et technologiques de l'IdO poussent les différents acteurs dans une course au développement, y compris les ingénieurs en charge de construire des protocoles de télécommunications adaptés. À ce jour, l'AES reste une valeur sûre avec des capacités d'intégration raisonnables, ce qui fait de cet l'algorithme l'outil favori pour assurer des propriétés de sécurité. Cependant, depuis plus d'une décénnie maintenant, la communauté cryptographique s'intéresse à la constructions d'algorithmes dits "légers" afin de répondre aux besoins de l'IdO.

Chapitre 3

Cryptographie légère

« *When I grew up, if you called someone a lightweight it was a little bit an insult. [...] Let's take that negative term of lightweight and build something positive.* »

Matt Scholl, Lightweight Cryptography Workshop 2016

Sommaire

3.1 Contexte

La cryptographie légère, aussi appelée cryptographie à bas coût, n'a pas de définition bien établie. Selon le dictionnaire Larousse, on attribue l'adjectif léger à une chose "dont la densité est peu élevée ou moins élevée qu'une autre". Cette définition illustre bien le fait que la notion de légèreté est contextuelle et dépend directement de la référence comparative. De manière générale, la cryptographie légère désigne l'ensemble des cryptosystèmes pouvant être intégrés dans des plateformes à ressources limitées. Il peut s'agir de ressources limitées en temps, espace, portes logiques voire énergie. La construction d'une primitive de chiffrement pouvant satisfaire les contraintes de tous les objets connectés, en termes d'intégration mais également de sécurité, est un réel défi.

Si, initialement, il semblait raisonnable de concevoir des algorithmes avec un faible niveau de sécurité (*e.g.*, ≤ 80 bits de sécurité) pour les applications à faibles risques, la tendance s'est peu à peu infléchie. Le but de la cryptographie légère est d'offrir un niveau de sécurité équivalent à la cryptographie traditionnelle tout en minimisant les coûts d'implémentation. La réduction de ces coûts peut être obtenue à l'aide de nouvelles constructions cryptographiques ou d'optimisations (*e.g.*, compromis temps-mémoire) spécifiques à certains cas d'utilisations. Les différentes unités de mesures utilisées pour évaluer l'efficacité d'un algorithme cryptographique diffèrent selon la nature de l'implémentation (*i.e.*, matérielle ou logicielle).

3.1.1 Implémentations logicielles

3.1.1.1 Coûts d'implémentation

Les critères que l'on cherche à optimiser sont principalement la taille d'implémentation, les ressources mémoires ainsi que le temps de calcul. Pour les implémentations logicielles, la taille d'implémentation désigne la taille du code (mémoire Flash ou ROM) et les ressources mémoires désignent la consommation de la RAM. La vitesse d'exécution, aussi appelée débit, est généralement exprimée en cycles par octet (cpb pour *cycles per byte*) et est calculée en moyennant de nombreuses mesures. Ces trois métriques dépendent directement les unes des autres et sont liées à la technique d'implémentation adoptée.

3.1.1.2 Techniques d'implémentation

Tables de précalcul De nombreuses implémentations cryptographiques logicielles utilisent des tables de précalcul afin d'optimiser le débit. À titre d'exemple, l'implémentation la plus rapide de l'AES sur les plateformes 32-bit, consiste en l'utilisation de 4 tables de précalculs T_i pour $0 \leq i \leq 3$ définies comme suit.

$$T_0[x] = \begin{pmatrix} 2 \\ 1 \\ 1 \\ 3 \end{pmatrix} \cdot S[x] \qquad T_1[x] = \begin{pmatrix} 3 \\ 2 \\ 1 \\ 1 \end{pmatrix} \cdot S[x] \qquad T_2[x] = \begin{pmatrix} 1 \\ 3 \\ 2 \\ 1 \end{pmatrix} \cdot S[x] \qquad T_3[x] = \begin{pmatrix} 1 \\ 1 \\ 3 \\ 2 \end{pmatrix} \cdot S[x] \tag{3.1}$$

Dans l'équation 3.1, x désigne un octet alors que $T_i[x]$ désigne un entier 32-bit non signé et $S[x]$ fait référence à l'application de la boîte-S. Ainsi, chaque table consiste en 256 entrées de 4 octets soit 1 ko, et le calcul d'une ronde de l'AES est défini par

$$\begin{pmatrix} x_i^{r+1} \\ x_{i+1}^{r+1} \\ x_{i+2}^{r+1} \\ x_{i+3}^{r+1} \end{pmatrix} = T_0[x_i^r] \oplus T_1[x_{i+4}^r] \oplus T_2[x_{i+8}^r] \oplus T_3[x_{i+12}^r] \oplus \begin{pmatrix} k_0^{r+1} \\ k_1^{r+1} \\ k_2^{r+1} \\ k_3^{r+1} \end{pmatrix} \qquad 0 \leq i \leq 3 \tag{3.2}$$

où x^r (resp. k^r) désigne l'état interne (resp. la clef) au tour numéro r et x_i (resp. k_i) désigne l'octet numéro i. Ainsi, le calcul d'une ronde de l'AES nécessite uniquement 16 accès mémoire et 16 opérations `XOR`. Cette implémentation de l'AES est celle utilisée par la librairie cryptographique OpenSSL, dévelopée en langage C et utilisée par de nombreux serveurs Internet. De plus, il est à noter que les tables T_i sont équivalentes les unes aux autres à une rotation près. Par conséquent, il est possible de tronquer 3 ko de mémoire au détriment de 12 rotations additionnelles par ronde.

Les tables de précalculs peuvent aussi être d'une grande utilité pour la cryptographie asymétrique. Par exemple, OpenSSL utilise de tels compromis temps-mémoire pour la cryptographie sur les courbes elliptiques, notamment pour la multiplication du générateur par un scalaire. Au delà des ressources mémoires qu'elles nécessitent, les tables de précalculs sont particulièrement sensibles aux attaques par analyse de temps de calcul pour les processeurs équippés de mémoires caches. Bien que la plupart des processeurs dédiés à l'IdO ne sont pas sophistiqués au point d'embarquer ce genre de fonctionnalité (*e.g.*, ARM Cortex-M3), des caches sont parfois rajoutés sur la plateforme afin d'augmenter les performances (*e.g.*, microcontrôleur STM32F407VG). Par conséquent, les implémentations embarquées favorisent des approches dites "régulières" qui permettent d'aboutir à un temps d'exécution constant.

Implémentations bitslicées Une technique pour implémenter un algorithme cryptographique en temps constant est de réécrire ce dernier au plus bas niveau, en utilisant uniquement des opérations logiques bit à bit, comme s'il s'agissait d'une implémentation matérielle. À titre d'exemple, considérons la boîte-S de l'AES. De nombreux circuits logiques ont été proposés pour cette composante, notamment celui présenté dans [BP10] qui est le plus compact à ce jour avec 115 portes logiques au total (dont 32 non-linéaires). Ainsi, l'application de la boîte-S à un octet nécessite 115 opérations, et ce, sans compter les opérations non logiques (*e.g.*, instruction `MOV`) requises pour sauvegarder certains résultats intermédiaires dans des variables temporaires. Pour ce faire, il est nécessaire de stocker chaque bit dans une variable dédiée. Cependant, il est possible de tirer profit du fait que cette fonction doit être appliquée à chaque octet de l'état interne. Au lieu de considérer chaque bit indépendamment, il est possible d'en regrouper plusieurs dans un même registre afin de paralléliser les calculs. Par exemple, un registre $\mathcal{R}_j$ de 16 bits pourrait contenir les bits numéro j de chaque octet x_i de l'état interne, notés $x_{i,j}$, comme illustré par la figure 3.1.

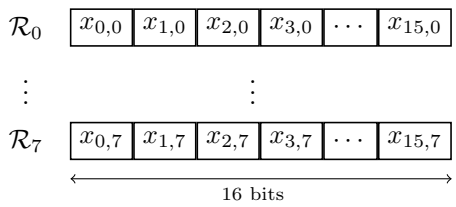


FIGURE 3.1 – Représentation bitslicée de l'état interne de l'AES

Ainsi, le calcul des 16 boîtes-S peut être effectué en parallèle en exécutant une seule fois la série d'instructions sur les registres $\mathcal{R}_j$. Il est à noter que le niveau de parallélisme est borné par l'architecture de la plateforme d'exécution. Une plateforme 8-bit nécessiterait deux fois plus d'instructions tandis qu'une plateforme 32-bit ou 64-bit permettrait de traiter plusieurs blocs en parallèle comme proposé dans la littérature [Kön08, KS09]. Cette technique appelée implémentation bitslicée (*bitsliced implementation*) a été introduite par Eli Biham en 1997 pour proposer une implémentation logicielle efficace du DES sur les architectures 64-bit [Bih97].

Il existe de nombreuses manières d'arranger les bits dans les registres selon leur taille et la spécification de l'algorithme. Dans le cas de l'AES, la représentation illustrée en figure 3.1 est uniquement optimale pour le calcul de la boîte-S. En effet, les autres sous-fonctions opèrent sur les lignes ou colonnes de l'état interne, ce qui nécessite d'avoir recours à des rotations bit à bit aditionnelles pour correctement agencer les registres selon les opérations. De nombreux algorithmes sont maintenant pensés pour favoriser les implémentations bitslicées par construction, à l'instar des LS-designs [GLSV15].

3.1.2 Implémentations matérielles

3.1.2.1 Intérêts

Dans un modèle d'exécution logiciel, le processeur réécrit le programme en tâches unitaires exécutées les unes à la suites des autres. Bien que les processeurs multicœurs permettent d'augmenter le nombre d'opérations exécutées simultanément en un cycle d'horloge, ils restent plus onéreux et énergivores que leurs homologues monocœurs, deux caractéristiques non souhaitables pour les plateformes à faibles ressources de l'IdO. Grâce à leur caractère spatial, les implémentations matérielles apportent quant à elles un gain significatif en termes de preformances par rapport aux processeurs. C'est pour cette raison que de nombreuses plateformes intègrent des implémentations cryptographiques matérielles. Par exemple, les processeurs dernière génération intègrent le jeu d'instruction AES-NI [Gue09, Gue10] introduit par Intel en 2008 et qui permet de calculer une ronde d'AES en un cycle d'horloge grâce à une implémentation matérielle.

3.1.2.2 Coûts d'implémentation

Pour les implémentations matérielles, la taille d'implémentation et l'empreinte mémoire sont regroupées sous une même métrique appelée *Gate Equivalent* (GE) qui mesure la complexité d'un circuit intégré, où 1 GE correspond à une porte logique `NAND`. Le débit d'une implémentation matérielle désigne généralement la quantité de données traitée par unité de temps (*e.g.*, le nombre d'octets par seconde). Ces deux métriques sont clairement dépendantes l'une de l'autre et sont directement liées aux techniques d'implémentation.

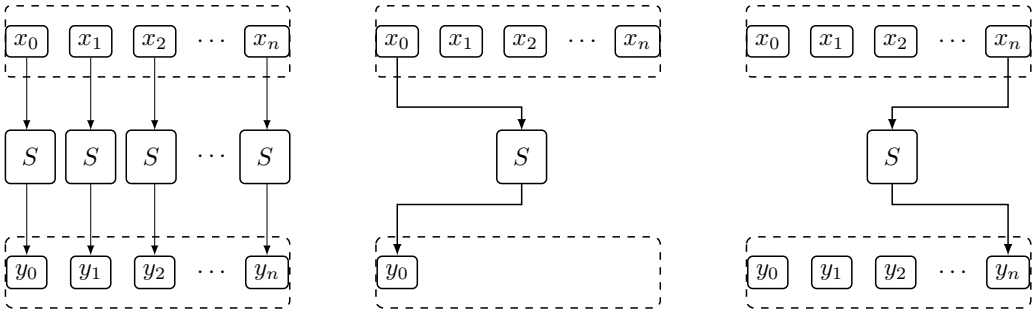
3.1.2.3 Techniques d'implémentation

Il existe principalement deux différentes approches pour une implémentation matérielle d'une primitive cryptographique. À titre d'exemple, reconsidérons l'exemple de la boîte-S de l'AES.

Implémentations matérielles itératives Les implémentations itératives permettent de minimiser le coût en GE. Le principe est de construire un circuit qui peut traiter les données de manière séquentielle. Dans le cas de la boîte-S de l'AES, cela consisterait en un unique circuit qui traiterait chaque octet de l'état interne indépendamment, comme indiqué par les figures 3.2b et 3.2c.

Implémentations matérielles pipelinées En contrepartie, les implémentations matérielles pipelinées consistent à dupliquer les circuits afin de paralléliser les calculs et d'augmenter le débit. Dans le cas de la boîte-S de l'AES, cela consisterait à en dupliquer le circuit 16 fois afin de traiter l'intégralité de l'état interne simultanément, comme illustré par la figure 3.2a.

Il est souhaitable pour un algorithme cryptographique léger d'être modulable et de pouvoir facilement s'adapter à chaque technique d'implémentation selon les coûts que l'on cherche à optimiser.



(a) Implémentation matérielle pipelinée

(b) Implémentation matérielle itérative ($t = 0$)

(c) Implémentation matérielle itérative ($t = n$)

FIGURE 3.2 – Différentes stratégies d'implémentations matérielles pour une boîte-S

3.1.3 Écosystème

3.1.3.1 Industrialisation

Plus d'une centaine d'algorithmes légers ont été développés durant les dernières décénnies par différents acteurs de la communauté cryptographique tels que des industriels, des académiques ou encore des agences gouvernementales [BP17]. Des algorithmes propriétaires, développés par des sociétés afin de répondre à leurs besoins, ont été utilisés durant de nombreuses années malgré leur faible niveau de sécurité. Un exemple phare est la solution d'antidémarrage de voiture Megamos commercialisée par la société EM Microelectronic dès la fin des années 1990. Le principe d'un antidémarrage de voiture est d'intégrer un récepteur radio dans la clef de contact ainsi qu'un émetteur radio à proximité de la serrure de démarrage. Ainsi, la clef de contact est authentifiée au moyen de méthodes cryptographiques lorsqu'elle est introduite dans la serrure de démarrage, afin de déverrouiller le boîtier électronique moteur de la voiture. Un tel mécanisme a pour rôle de prévenir les vols de voiture réalisés en connectant les câbles manuellement. En 2015, des chercheurs ont réussi à faire de la rétro-ingénierie sur les mécanismes cryptographiques embarqués dans les solutions d'antidémarrage de voiture Megamos [VGE15]. Ils sont parvenus à reconstruire l'algorithme de chiffrement utilisé, dont le fonctionnement était jusqu'alors gardé secret. Indépendamment des faiblesses constatées concernant la gestion des clefs et la protection en écriture de la mémoire, la sécurité de l'algorithme symétrique utilisant des clefs de 96 bits laisse à désirer et peut être cassée en exécutant 2^{56} chiffrements, voire 2^{49} en utilisant une table de précalcul de 12 To. Dix ans auparavant, la solution d'antidémarrage développée par Texas Instrument avait également été victime de rétro-ingénierie [BGS+05]. L'algorithme de chiffrement propriétaire DST40 utilisait des clefs de 40 bits seulement et pouvait être mis en échec par de simples attaques par force brute.

Ce type d'évènements a permis de sensibiliser les industriels aux limites de la sécurité par l'obscurité. De plus en plus d'entreprises rendent leurs constructions cryptographiques publiques. Par exemple, la société Qualcomm a récemment publié la spécification de son chiffrement par bloc QARMA [Ava17], utilisé pour authentifier les pointeurs sur les plateformes ARMv8.3 afin de contrer certains vecteurs d'attaque [Inc17].

3.1.3.2 Compétitions ouvertes

Lorsqu'un besoin en termes de primitives est largement reconnu par la communauté cryptographique, cette dernière a pour tradition d'organiser des compétitions afin d'étudier de nouvelles constructions ou de faire émerger de nouveaux standards, comme cela a été fait pour l'AES.

Le projet eSTREAM En 2004, alors que l'AES commence à s'imposer en tant que primitive symétrique de référence, la communauté cryptographique s'interroge sur le besoin de constructions dédiées pour les algorithmes de chiffrement à flot au lieu d'associer l'AES avec un mode opératoire adéquat (*e.g.*, mode CTR). De nombreux experts ont conclu que de telles constructions seraient bénéfiques pour les cas d'usage avec de fortes exigences en termes de débit, ainsi que pour les implémentations matérielles contraintes. C'est dans ce contexte que le réseau européen d'excellence en cryptologie ECRYPT a lancé le projet eSTREAM, avec pour ambition de définir de nouvelles primitives de chiffrement à flot. L'appel à soumissions fut publié en octobre 2004 et définissait deux profils différents : les chiffrements à flot pour les applications logicielles à fortes exigences de débit et ceux destinés aux implémentations matérielles à faibles ressources. Au total, 34 algorithmes furent soumis. Après plusieurs années d'analyse, un portfolio de huit algorithmes fut publié en avril 2008 [RB08]. Outre le portfolio final, le but de ce projet était d'étudier les techniques de conception de chiffrements à flot, moins bien maîtrisées que celles dédiées aux chiffrements par bloc. Cependant, peu de temps après la publication du portfolio final, la cryptanalyse en quelques secondes sur un simple ordinateur d'une instance de F-FCSR fut publiée [HJ08]. Ce résultat a nécessité la mise à jour du portfolio seulement quelques mois après sa publication en retirant l'algorithme concerné, soulignant la difficulté éprouvée par la communauté cryptographique à analyser ce type de constructions.

La compétition CAESAR Les protocoles de sécurité assurent non seulement la confidentialité des données mais aussi leur authenticité. Afin de répondre à ce besoin, la notion de chiffrement authentifié a été introduite par Mihir Bellare et Chanathip Namprempre en 2000 [BN00] et désigne les schémas cryptographiques qui permettent d'assurer ces deux propriétés simultanément. Deux ans plus tard, Phillip Rogaway complète cette définition en introduisant la notion de données additionnelles [Rog02], destinées à être authentifiées mais non chiffrées (*e.g.*, en-tête d'un paquet réseau). Une manière triviale de construire un tel schéma est de combiner plusieurs primitives cryptographiques (*e.g.*, HMAC et AES). Cette méthode, en plus d'être non optimale en termes de performances, peut être source de vulnérabilités dues à des erreurs d'implémentation (*e.g.*, l'utilisation d'une seule clef pour les deux primitives). Ainsi, des modes opératoires dédiés ont été conçus afin d'aboutir à un chiffrement authentifié avec données additionnelles à partir d'un simple chiffrement par bloc, notamment les modes CCM et GCM utilisés par de nombreux protocoles. Bien qu'ils soient plus efficaces en termes de performances, des vulnérabilités subsistent comme exposé dans plusieurs publications [Saa12, PC15, BZD+16]. C'est dans ce contexte que la compétition CAESAR (*Competition for Authenticated Encryption : Security, Applicability, and Robustness*) a émergé, avec l'objectif de proposer des chiffrements authentifiés qui offriraient des avantages en termes de performances et/ou de sécurité par rapport à l'AES-GCM. En 2014, l'appel à soumission reçut 57 propositions au total. L'ensemble des algorithmes est hétérogène, certains sont basés sur l'AES en proposant un nouveau mode opératoire (*e.g.*, AES-JAMBU) tandis que d'autres reposent sur de nouvelles constructions (*e.g.*, ACORN). En 2016, le comité d'évaluation de la compétition décida de distinguer trois cas d'usage, dont un concernant les applications légères à faibles ressources. Les candidats pour ce cas d'usage doivent respecter les critères listés ci-dessous.

- Bénéficier d'une implémentation compacte aussi bien en logiciel qu'en matériel.
- Atteindre un débit compétitif, et en particulier être efficace pour une faible quantité de données à traiter (e.g., paquet de 16 octets).
- Capacité d'intégrer efficacement des contremesures contre les attaques par canaux auxiliaires.

En mars 2018, les finalistes du concours furent annoncés et classés selon les différents cas d'usage définis. Pour les applications légères, les deux finalistes furent ACORN [Wu16] et ASCON [DEMS16]. En février 2019, la compétition prit fin en annonçant un portfolio qui contient deux algorithmes pour chaque cas d'usage : une recommandation principale et une secondaire. Ainsi, ACORN et ASCON font tous deux partie de ce portfolio mais ASCON est privilégié étant donné qu'il constitue la recommandation principale pour les applications légères à faibles ressources.

3.1.3.3 Standardisation

Les compétitions ouvertes organisées par la communauté cryptographique permettent d'aboutir à des algorithmes dont la spécification publique a été analysée par différents cryptanalystes durant plusieurs années, garantissant un certain niveau de sécurité. Seulement, à l'instar des deux concours eSTREAM et CAESAR, peu d'entre eux constituent des projets de standardisation à proprement parler. Deux des principaux projets de standardisation internationaux concernant la cryptographie légère actuellement en cours sont présentés ci-dessous.

ISO/IEC 29192 L'organisation internationale de normalisation (ISO) et la commission électrotechnique internationale (IEC) ont pour rôle de publier et maintenir les standards des technologies de l'information et des communications. La cryptographie légère constitue un projet de standardisation développé sous la référence ISO/IEC 29192. À ce jour, plusieurs algorithmes y sont définis, en particulier TRIVIUM pour les chiffrements par flot. En effet, bien que les compétitions cryptographiques ne débouchent pas nécessairement sur un standard, les vainqueurs constituent de bons candidats étant donné qu'ils ont déjà résisté à plusieurs années de cryptanalyse publique.

Ce standard n'est pas figé et sera étoffé durant les prochaines années. Par exemple, les chiffrements légers de l'agence nationale de la sécurité des États-Unis (NSA) SIMON et SPECK ont été à l'étude dès 2015 pour être inclus dans l'ISO/IEC 29192. La publication originale de ces deux algorithmes [BSS+13] ne justifie pas les choix de conception et ne contient aucune analyse de sécurité. En 2017, suite à la pression de la communauté cryptographique et du groupe de travail ISO/IEC concerné, la NSA publia une note sur les choix de conception [BSS+17]. Cependant, certaines zones d'ombre persistent telles que le choix des matrices utilisées pour la péparation des sous-clefs dans SIMON. Ce manque de transparence de la NSA justifia le refus d'inclure ces deux algorithmes à l'ISO/IEC 29192. D'autres algorithmes sont actuellement à l'étude, tel que le MAC CHASKEY [MMH+14].

Si ce standard spécifie plusieurs primitives cryptographiques légères prêtes à l'emploi, ces dernières ne conviennent pas nécessairement à tous les cas d'usage. Par exemple, l'ISO/IEC 29192 répertorie deux chiffrements par blocs qui sont CLEFIA [SSA+07] et PRESENT [BKL+07]. PRESENT, avec ses deux variantes utilisant une clef de 80 ou 128 bits, a été conçu avec pour objectif d'être particulièrement efficace pour les implémentations matérielles, sans se soucier de l'aspect logiciel. Sans surprise, il n'offre pas de bonnes performances lorsqu'il est implémenté sur microcontrôleur en langage C ou Assembleur comme souligné par plusieurs publications [CMM13, DCK+15]. Une technique d'optimisation fut récemment proposée et permet d'améliorer d'un facteur 8 sa vitesse d'exécution sur ARM Cortex-M3 [RAL17]. Cependant, même en bénéficiant de cette optimisation, la version 80-bit de PRESENT nécessite plus de cycles que l'implémentation bitslicée d'AES proposée par Peter Schwabe et Ko Stoffelen sur la même plateforme [SS17], soulignant son inefficacité en logiciel. Le second chiffrement par bloc, CLEFIA, utilise une clef de 128 bits et permet d'aboutir à des implémentations qui possèdent des caractéristiques similaires à l'AES aussi bien en logiciel [CMM13, HAI+17] qu'en matériel [SHAS08, HO12]. Le fait que

CLEFIA soit standardisé comme chiffrement par bloc léger alors qu'il ne constitue pas une réelle alternative à l'AES met l'accent sur le manque d'une définition formelle et bien établie de la cryptographie légère.

Le projet LWC du NIST En 2013, le NIST démarra le projet *LightWeight Cryptography* (LWC) dans le but d'étudier les algorithmes cryptographiques légers et d'établir une stratégie de standardisation. Deux ateliers de travail furent organisés en 2015 et 2016 afin d'identifier les limitations de la cryptographie conventionelle et l'intérêt de primitives dédiées aux cas d'usage à faibles ressources. En 2017, le NIST publia un rapport [NIS] détaillant le projet et annonçant la décision de créer un portfolio d'algortihmes cryptographiques symétriques légers au moyen d'un processus d'évaluation ouvert, comme pour AES et SHA-3. En 2018, le NIST publia l'appel à soumissions qui stipule que chaque candidat devra définir un chiffrement authentifié symétrique avec, en option, un mode de hachage. Le concours a commencé en 2019 et a reçu 56 soumissions [1], dont ASCON. Ce projet de standardisation devrait durer plusieurs années avant d'aboutir au portfolio final.

Ci-après sont présentés les différents schémas de conception étudiés par la suite.

3.2 Quelques constructions symétriques prometteuses

3.2.1 Structures ARX

3.2.1.1 Concept

Le terme ARX est un acronyme (Addition Rotation XOR) faisant référence à une classe d'algorithmes cryptographiques symétriques uniquement basés sur trois opérations élémentaires : l'addition modulaire, la rotation bit à bit et le XOR. Bien que l'origine de cette appelation remonte à 2009 [Wei09], le concept est antérieur et fut introduit par le chiffrement par bloc FEAL [SM87]. Mis à part le fait que ces algorithmes ne reposent que sur les trois opérations mentionnées ci-dessus, il n'existe aucune contrainte sur leur structure. Ainsi, on trouve dans la littérature des algorithmes ARX basés sur des construction traditionnelles (*e.g.*, SPECK et TEA [WN95] utilisent un réseau de Feistel), ou des constructions dédiées (*e.g.*, ChaCha20 [Ber08a]).

Les structures ARX se prêtent particulièrement bien aux implémentations logicielles étant donné la simplicité des opérations sur lesquelles elles reposent. Cela permet d'aboutir à des implémentations compactes et extrêmement efficaces [DCK+15], étant donné que chacune de ces opérations nécessite uniquement un cycle d'horloge. En effet, le modulo à considérer pour les additions est choisi en fonctions des architectures pour lesquelles on cherche à optimiser la plateforme (*e.g.*, 2^{32} pour les plateformes 32-bit), de manière à ce que la réduction modulaire soit transparente. Aussi, ces algorithmes sont intrinsèquement résistants aux attaques par temps de calcul étant donné qu'ils ne requièrent aucune table de précalculs.

Cependant, ces constructions souffrent de plusieurs inconvénients. Premièrement, les additions modulaires et les rotations bit à bit sont plus coûteuses à implémenter que de simples portes logiques XOR en matériel. Ainsi, contrairement aux implémentations logicielles qui reposent simplement sur l'unité arithmétique et logique (UAL) du processeur, les structures ARX ne constituent pas la meilleure option pour les implémentation matérielles. Ensuite, contrairement aux structures traditionnelles (*e.g.*, réseaux de subsitution-permutation) pour lesquelles il existe de nombreux outils de cryptanalyse qui permettent de dériver des bornes de sécurité face aux attaques linéaires et différentielles,

1. https ://csrc.nist.gov/projects/lightweight-cryptography/round-1-candidates

les constructions ARX dédiées ne bénéficient pas de telles preuves de sécurité [DPU+16]. Finalement, ces structures ne sont pas optimales pour intégrer des contremesures pour se protéger des attaques par consommation de courant [CG00, CGTV15], comme détaillé dans le chapitre 5.

3.2.1.2 La famille de chiffrements à flot ChaCha

Origine et applications ChaCha est une famille de chiffrement à flot ARX introduite par Daniel J. Bernstein en 2008. Elle est dérivée de SALSA20 [Ber08b], faisant partie du portfolio eSTREAM pour les applications logicielles à fortes exigences de débit, afin d'améliorer les propriétés de diffusion tout en conservant des performances smiliaires. Bien que publié il y a une dizaine d'années, ChaCha connut un gain de popularité en 2013, lorsque Google commença à étudier son intégration au protocole TLS afin de proposer une alternative à l'AES pour les plateformes démunies du jeu d'instructions AES-NI. En 2014, la plupart des connections HTTPS effectuées sous les smartphones Android depuis les navigateurs Chrome vers les serveurs de Google utilisait ChaCha au lieu de l'AES afin de réduire la latence et d'économiser la batterie [Bur14]. Plus récemment, la version 1.3 du protocole TLS fut publiée en 2018 [Res18] et intègre un nouveau AEAD ChaCha20-Poly1305 [LCM+16] basé sur ChaCha, conférant à ce dernier un fort gain de popularité. ChaCha devrait également avoir sa carte à jouer dans l'IdO étant donné les différents avantages qu'il présente par rapport à l'AES sur les plateformes embarquées démunies d'accélérateur matériel. Notamment, ChaCha20-Poly1305 est utilisé par les produits Apple HomeKit [App18] qui désignent des accessoires domotiques (*e.g.*, détecteur de fumée connecté).

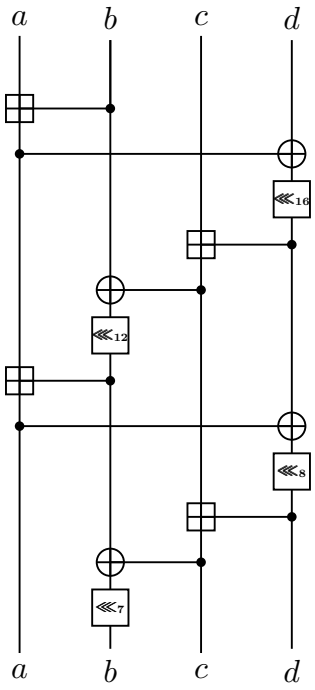
Spécification ChaCha utilise des clefs de 256 bits et fonctionne comme un chiffrement par bloc utilisé en mode CTR. Son état interne est constitué de 512 bits, représentés sous la forme d'une matrice 4×4 où chaque élément désigne un mot de 32 bits. Pour chaque chiffrement, la moitié de l'état interne est remplie avec la clef tandis que les deux quarts restants sont remplis avec un nonce et une constante, comme illustré par la figure 3.3. La constante `'expand 32-byte k'` a pour rôle de réduire la quantité de données au sein de l'état interne qu'un attaquant pourrait contrôler, tandis que le nonce, qui ne doit jamais être répété, est construit à partir d'un vecteur d'initialisation et d'un compteur de bloc.

`'expa'`	`'nd 3'`	`'2-by'`	`'te k'`
k_0	k_1	k_2	k_3
k_4	k_5	k_6	k_7
iv_0	iv_1	iv_2	iv_3

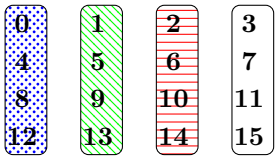
FIGURE 3.3 – Initialisation de l'état interne de ChaCha

Si ChaCha définit une famille de chiffrement à flot, son noyau est en réalité basé sur un générateur pseudo-aléatoire. À l'instar d'un chiffrement par bloc itératif, une fonction de tour est appliquée à l'état interne un certain nombre de fois afin de transformer le bloc initial en pseudo-aléa qui servira ensuite à chiffrer le texte clair à l'aide de l'opérateur `XOR`. Plus précisément, la fonction de tour de ChaCha consiste en quatre applications parallèles d'une fonction appelée "quart de tour" et notée Q dans ce manuscrit. Un quart de tour met à jour quatre mots de l'état interne (*i.e.*, un quart de bloc) comme défini par la figure 5.6a. Selon la parité du numéro de tour, les quarts de tour sont appliqués sur les colonnes ou les diagonales comme illustré par les figures 5.6b et 5.6c.

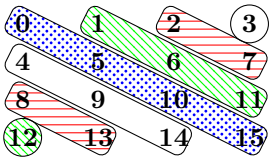
ChaCha$X$ fait référence à une instance de ChaCha appliquant X fois la fonction de tour (*e.g.*, $4X$ quarts de tours au total). Trois variantes sont définies avec respectivement 8, 12 ou 20 tours, définissant différents compromis performance-sécurité. Bien que la version à 12 tours semble suffisamment sécurisée pour assurer les 256 bits de sécurité [CM16],



(a) $Q(a,b,c,d)$



(b) Tour pair



(c) Tour impair

FIGURE 3.4 – La fonction quart de tour et ses applications

ChaCha20 est généralement privilégier car il permet de jouir d'une importante marge de sécurité tout en conservant de bonnes performances. C'est d'ailleurs la version à 20 tours qui a été standardisée par l'IETF [LCM+16]. Cependant, des ingénieurs de Google ont récemment proposé un mécanisme de chiffrement de disque dur [CB18] basé sur ChaCha12. Ce dernier a notamment été intégré dans la version 5.0 du noyau Linux [Edg19].

Si l'itération de X tours permet d'obtenir un pseudo-aléa suffisamment sûr, cela n'empêche pas un attaquant de retrouver le bloc initial (et donc la clef de chiffrement) via une attaque clair connu étant donné que les quarts de tour sont inversibles. ChaCha se prémunit contre ce chemin d'attaque en ajoutant à chaque élément de l'état interne son homologue de l'état initial. De nombreuses implémentations appliquent $\frac{X}{2}$ double tours au lieu de X tours afin d'éviter les tests conditionnels sur la parité du numéro de tour, comme détaillé par l'algorithme 3.1 qui résume l'intégralité du procédé de chiffrement.

Propriétés d'implémentation ChaCha utilise l'addition modulo 2^{32}, ce qui rend ses instances particulièrement adaptées aux plateformes embarquées 32-bit. En revanche, les plateformes 16-bit et 8-bit sont désavantagées car le calcul de l'addition nécessite la propagation de la retenue d'un registre à un autre et les rotations définies sur 32 bits nécessitent des opérations supplémentaires. Les plateformes plus avancées ne sont pas en reste étant donné qu'au sein d'un même tour, les quarts de tour sont indépendants les uns des autres et peuvent donc être appliqués en parallèle. Il est possible de tirer profit de cette propriété sur les processeurs récents en utilisant les instructions de type *Single Instruction Multiple Data* (SIMD) afin de paralléliser les calculs. Le principe de ces instructions est d'utiliser des registres, aussi appelés vecteurs, plus larges afin d'y stocker plusieurs mots auxquels on appliquera la même instruction en parallèle. Différents types d'instructions SIMD existent avec différentes tailles des vecteurs, allant de 128 (*e.g.*, SSE) à 512 bits (*e.g.*, AVX-512). Avec des registres de 128 bits, il est possible de calculer les 4 quarts de tour (*i.e.*, un tour complet) en parallèle en stockant chaque ligne de l'état interne dans

Algorithme 3.1 Chiffrement à l'aide de ChaCha$R$

Entrée(s):
Texte clair P de n bits
Clef de chiffrement k
Compteur ctr
Vecteur d'initialisation iv

Sortie(s): Texte chiffré C de n bits

pour i de 0 à $\lfloor \frac{n}{512} \rfloor$ **faire**
  $B \leftarrow \texttt{init}(k, ctr, iv)$ ▷ Initialisation de l'état interne
  $B' \leftarrow B$ ▷ Variable de travail
  pour j from 0 to $\frac{R}{2} - 1$ **faire** ▷ Double tour
    $Q(B'_0, B'_4, B'_8, B'_{12})$ ▷ Tour pair
    $Q(B'_1, B'_5, B'_9, B'_{13})$
    $Q(B'_2, B'_6, B'_{10}, B'_{14})$
    $Q(B'_3, B'_7, B'_{11}, B'_{15})$
    $Q(B'_0, B'_5, B'_{10}, B'_{15})$ ▷ Tour impair
    $Q(B'_1, B'_6, B'_{11}, B'_{12})$
    $Q(B'_2, B'_7, B'_8, B'_{13})$
    $Q(B'_3, B'_4, B'_9, B'_{14})$
  fin pour
  $B \leftarrow B \boxplus B'$ ▷ Addition finale
  $C_i \leftarrow P_i \oplus B$
  $ctr \leftarrow ctr + 1$
fin pour

un registre dédié. De plus, ChaCha peut être davantage optimisé si des vecteurs de plus grandes tailles sont disponibles [GG14].

Nous étudierons la sécurité de ChaCha au chapitre 5.

3.2.2 Fonctions éponges

3.2.2.1 La construction de l'éponge

Concept La construction de l'éponge a été introduite en 2007 [BDPVA07] avec comme but initial de construire des fonctions de hachage cryptographiques capables de produire des empreintes de n'importe quelle taille. Elle a notamment été utilisée pour la conception de la famille de fonctions de hachage KECCAK [BDPVA] qui a été désignée comme algorithme standard de hachage SHA-3 par le NIST en 2015.

La construction de l'éponge consiste en une construction itérative basée sur une fonction f opérant sur $b = r + c$ bits qui désigne la taille de l'état interne. Dans un premier temps, l'état interne est initialisé à zéro. Les données à traiter sont ensuite complétées par un processus de bourrage, afin d'obtenir une entrée dont la longueur est un multiple de r, puis divisées en bloc de r bits. Le traitement de ces données est réalisé en deux étapes comme illustré par la figure 3.5.

- Dans la première phase dite d'absorption, les blocs de données sont successivement intégrés à l'état interne en appliquant un XOR bit à bit et la fonction f à l'état interne. Cette phase prend fin lorsque tous les blocs ont été traités.
- Dans la seconde phase dite d'essorage, les r premiers bits de l'état interne sont retournés comme bloc de sortie avant d'appliquer une nouvelle fois f à l'état interne. Le nombre de blocs de sortie est variable et choisi par l'utilisateur.

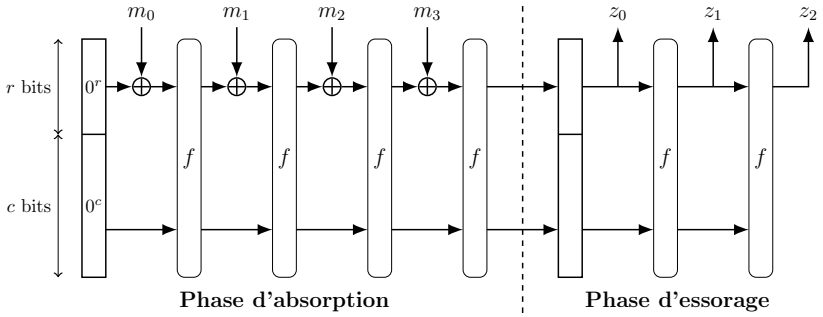


FIGURE 3.5 – Construction de l'éponge

Il est à noter que les c derniers bits ne sont jamais révélés et sont uniquement modifiés par l'application de la fonction f sur l'état interne. L'intérêt de ne pas dévoiler ces c derniers bits est de compliquer la tâche à un attaquant qui chercherait à calculer la préimage d'une empreinte. Ainsi, en fixant un paramètre c assez grand, il devient calculatoirement impossible de retrouver les c derniers bits et, par extension, de retrouver la préimage d'une empreinte. Le paramètre c est appelé la capacité de la fonction éponge et est directement lié à son niveau de sécurité. En contrepartie, le paramètre r est directement lié aux performances de la foncton éponge puisqu'il définit la quantité de données absorbée et essorée. La taille de l'état interne $b = r + c$ étant fixe, augmenter un paramètre revient à réduire le second, résultant en un compromis sécurité/performance.

Cryptographie à base de permutations La construction de l'éponge peut être utilisée pour concevoir différents types de primitives cryptographiques symétriques, autres que des fonctions de hachage (*e.g.*, le chiffrement à flot SPRITZ [RS16]). Plus généralement, cette construction est à l'origine d'une nouvelle catégorie d'algorithmes symétriques qui reposent uniquement sur une permutation (*permutation-based cryptography*), ce qui permet d'aboutir à des implémentations efficaces. Par exemple, l'implémentation de la permutation GIMLI [BKL+17a] ne nécessite aucun accès mémoire sur de nombreuses plateformes, s'affranchissant des surcharges liées, et bénéficie d'une taille de code très compacte (*i.e.*, deux tweets en langage C [BKL+17b]).

Différents modes éponges ont été proposé afin d'aboutir à une primitive cryptographique à partir d'une permutation de taille fixe. Notamment, la construction duplex permet de concevoir un chiffrement authentifié [BDPVA12] et est dérivée en plusieurs variantes [BDH+12]. La famille de chiffrements authentifiés ASCON, annoncée comme finaliste de la compétition CAESAR pour la catégorie applications à faibles ressources et basée sur un mode éponge, est présentée ci-dessous.

3.2.2.2 La famille de chiffrements authentifiés ASCON

Structure générale ASCON assure un niveau de sécurité de 128 bits pour la confidentialité et l'authenticité des données. ASCON repose sur une permutation notée p opérant sur 320 bits et un mode éponge dont le fonctionnement est décrit en figure 3.6. Plus précisément, ASCON utilise deux permutations notées p^a et p^b où a et b désignent le nombre d'itérations appliquées à p avec $a > b$.

Le procédé de chiffrement est divisé en quatre phases successives.

- **La phase d'initialisation** remplit l'état interne de 320 bits avec la clef (128 bits), le vecteur d'initialisation (128 bits) et le nonce (64 bits). La permutation p^a est ensuite appliquée à l'état interne avant d'appliquer un `XOR` avec la clef bourrée avec des zéros. L'application d'une permutation plus robuste au sens cryptographique suivie

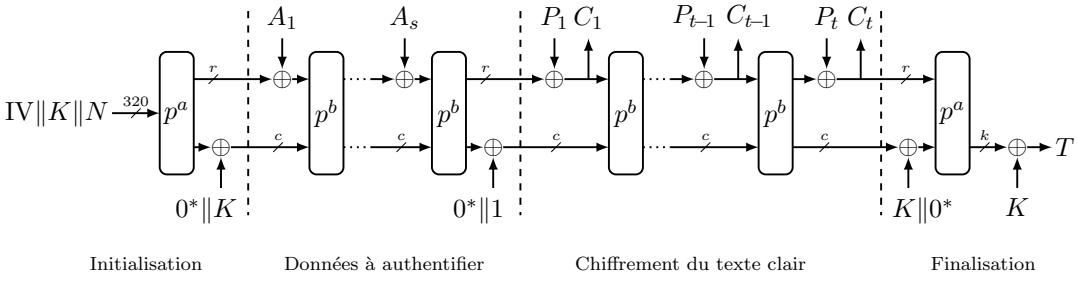


FIGURE 3.6 – ASCON en mode chiffrement

d'un XOR avec la clef assure que la compromission d'un état interne intermédiaire ne permette pas de retrouver l'état initial (*i.e.*, la clef).

- **Le traitement des données additionnelles** consiste à appliquer une phase d'absorption avec les données additionnelles complétées par un processus de bourrage et divisées en blocs de r bits. L'addition de $0^{329} \| 1$ à la fin de cette phase agit comme séparation de domaine afin de prévenir les attaques qui interfèrent le rôle du texte clair et des données additionnelles.
- Durant **la phase de chiffrement**, chaque bloc de texte clair est ajouté (via l'opérateur XOR) aux r premiers bits de l'état interne, le résultat constituant un bloc de de texte chiffré qui est ensuite extrait. Il s'ensuit une application de la permutation p^b, excepté pour le dernier bloc de texte clair. Ainsi, cette phase s'apparente à une phase d'essorage où le texte clair est injecté à l'état interne.
- **La phase de finalisation** ajoute la clef bourrée avec des zéros à l'état interne avant d'appliquer la permutation p^a. Le code d'authentification est défini par les 128 derniers bits de l'état interne ajouté à la clef de chiffrement. Tout comme le procédé de chiffrement, les ajouts de clefs et l'utilisation de p^a assurent que la compromission d'un état interne intermédiaire ne permet pas de forger des codes d'authentification.

Le procédé de déchiffrement diffère uniquement durant la phase de déchiffrement, où chaque bloc de r bit est extrait pour calculer le bloc clair correspondant avant d'être remplacé par le bloc chiffré.

ASCON contient deux instances, notées ASCON128 et ASCON128-A, dont les paramètres sont listés dans le tableau 3.1. Si les auteurs désignent ASCON128 comme la principale recommandation de ses auteurs, ASCON128-A permet d'atteindre des meilleures performances étant donné que la taille des blocs est doublée au détriment de la capacité. Afin de contrebalancer la dégradation du niveau de sécurité, le paramètre b est augmenté afin de renforcer le traitement des données à authentifier et à chiffrer.

TABLEAU 3.1 – Paramètres recommandés pour ASCON

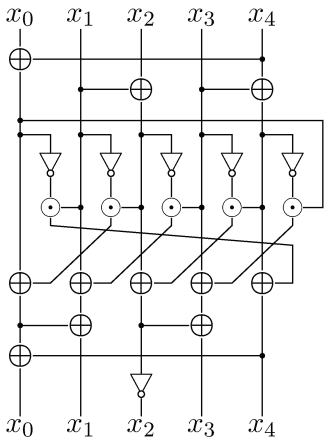
Instance	Taille en bits					a	b
	clef	nonce	MAC	r	c		
ASCON-128	128	128	128	64	256	12	6
ASCON-128A	128	128	128	128	192	12	8

La permutation p La permutation p, qui constitue la principale composante d'ASCON, consiste en un réseau de substitutions-permutations opérant sur un état de 320 bits

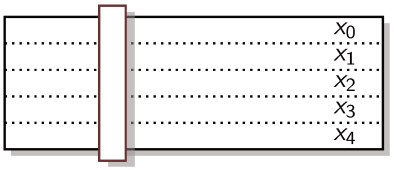
représenté par 5 variables de 64 bits notées x_i pour $0 \leq i \leq 4$. Ce réseau de substitutions-permutations consiste en trois opérations successives : un ajout de constante, une boîte-S et une couche linéaire.

Chaque ronde de p commence avec l'addition d'une constante à la variable x_2. La constante ajoutée dépend du numéro de tour et des paramètres a et b employés par l'instance d'ASCON utilisée.

La boîte-S utilisée par ASCON est définie sur 5 bits afin de favoriser les implémentations bitslicées par construction comme illustré par la figure 3.7b. Elle peut être implémentée à l'aide de 22 portes logiques et nécessite seulement un registre temporaire.



(a) Circuit de la boîte-S



(b) Approche bitslicée par construction

FIGURE 3.7 – La boîte-S 5-bit utilisée par ASCON

La couche linéaire a pour but d'apporter de la diffusion au sein de chaque variable, séparément. Elle repose uniquement sur l'opérateur `XOR` et des rotations bit à bit comme décrit ci-dessous.

$$
\begin{aligned}
x_0 &= x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28) \\
x_1 &= x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39) \\
x_2 &= x_2 \oplus (x_2 \ggg \ \ 1) \oplus (x_2 \ggg \ \ 6) \\
x_3 &= x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17) \\
x_4 &= x_4 \oplus (x_4 \ggg \ \ 7) \oplus (x_4 \ggg 41)
\end{aligned}
$$

Propriétés d'implémentation ASCON présente l'avantage d'être extrêment flexible pour les implémentations matérielles et offre la possiblité d'aboutir à des implémentations très compactes (*e.g.*, 2500 GE [GWDE15]) ou très performantes (*e.g.*, 13.2 Gbps [GWDE15]). De plus, l'aspect bitslice par construction permet également d'atteindre des bonnes performances en logiciel. La définition des variables sur 64 bits permet à ASCON d'être compétitif sur les processeurs avancés tout en restant attractif pour les plus petites plateformes. Généralement, il suffit de dupliquer les opérations bit à bit x fois pour un processeur n-bit où $n = \frac{64}{x}$, excepté pour la couche linéaire, où les rotations définies sur 64 bits se prêtent moins bien au découpage et nécessitent des opérations supplémentaires sur les plus petites plateformes.

3.2.3 Constructions dédiées aux chiffrements à flot

3.2.3.1 Registres à décalage à rétroaction

Les chiffrements à flots sont généralement basés sur des registres à décalage à rétroaction (FSR pour *feedback shift register*). Un tel mécanisme consiste en un registre contenant un état interne S décalé d'un cran à chaque étape. On note $S_{i,j}$ le j-ième bit de l'état interne après i décalages. Les bit injectés à l'état interne sont obtenus en appliquant une fonction de rétroaction Φ à l'état interne courant comme illustré par la figure 3.8. Le bits de sortie constitue un pseudo-aléa noté ks et utilisé pour chiffrer le texte clair à l'aide de l'opérateur `XOR`. Un chiffrement par flot peut être obtenu en initialisant l'état interne avec une clef secrète et un vecteur d'initialisation, et en le mettant à jour suffisament de fois afin d'obtenir un pseudo-aléa cryptographiquement sûr.

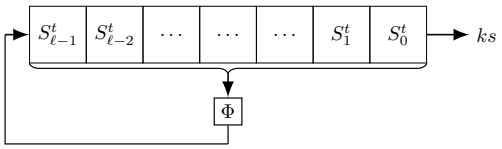


FIGURE 3.8 – Structure d'un registre à décalage à rétroaction (FSR)

Selon que Φ définit une fonction linéaire ou non-linéaire, on parle de registre à décalage à rétrocation linéaire (LFSR) ou non-linéaire (NFSR). Dans le cas d'un LFSR, les fonctions Φ couramment utilisées consistent à appliquer l'opérateur `XOR` à quelques bits de l'état interne. Ces LFSR sont peu coûteux à implémenter mais ne constituent pas un chiffrement sûr comme tels. En effet, il est nécessaire de faire intervenir une composante non-linéaire, sans quoi une attaque à clair connu permettrait d'obtenir un système d'équations linéaires des bits de clefs, pouvant être aisément résolu dès lors qu'il existe autant d'équations que d'inconnues (*e.g.*, en utilisant la méthode du pivot de Gauss). Plusieurs méthodes peuvent être employées pour contourner l'effet des propriétés de linéarité des LFSR. L'une d'entre elles consiste à appliquer une fonction, dite de filtrage, non-linéaire à l'état interne afin de produire la suite chiffrante, comme illustré par la figure 3.9.

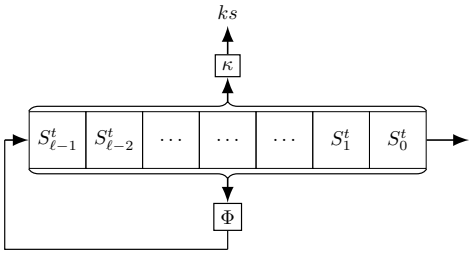


FIGURE 3.9 – Application d'une fonction de filtrage κ à un FSR

Afin d'accentuer les propriétés de non linéarité, de nouvelles constructions combinent ces méthodes avec des constructions de type NFSR. C'est le cas du chiffrement à flot ACORN [Wu16], finaliste de la compétition CAESAR pour la catégorie applications à faibles ressources.

3.2.3.2 Le chiffrement authentifié ACORN

Spécification ACORN assure un niveau de sécurité de 128 bits pour la confidentialité et l'authenticité des données. ACORN opère sur un état interne de 293 bits qui consiste en la concaténation de 6 LFSR ainsi que d'un registre additionnel de 4 bits comme illustré par la figure 3.10.

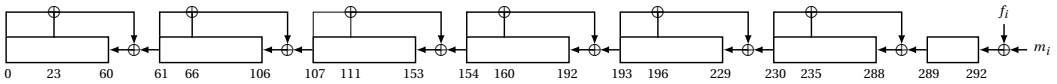


FIGURE 3.10 – La concaténation de 6 LFSR qui définit l'état interne d'ACORN. f_i et m_i désignent respectivement le bit de rétroaction et le bit d'entrée à l'étape i.

Bien qu'étant basée sur 6 LFSR, dont l'application des fonctions de rétroaction est effectuée à l'origine du processus de mise à jour de l'état interne, la structure globale d'ACORN est en réalité un NFSR étant donné que le bit de rétroaction est calculé par la fonction Φ non-linéaire définie ci-dessous

$$\Phi(S_i, ks_i, ca_i, cb_i) = S_{i,0} \oplus \neg S_{i,107} \oplus Maj(S_{i,244}, S_{i,23}, S_{i,160}) \oplus (ca_i \wedge S_{i,196}) \oplus (cb_i \wedge ks_i) \tag{3.3}$$

où $Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$ (*i.e.*, retourne le bit de majorité parmi x, y et z), $Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$ (*i.e.*, choisit, selon la valeur de x, de retourner y ou z), ca_i et cb_i désignent des bits de contrôle et ks_i désigne le pseudo-aléa à l'étape i. Ce bit pseudo-aléatoire est calculé à l'aide de la fonction de filtrage κ définie ci-dessous

$$\kappa(S_i) = S_{i,12} \oplus S_{i,154} \oplus Maj(S_{i,235}, S_{i,61}, S_{i,193}) \oplus Ch(S_{i,230}, S_{i,111}, S_{i,66}). \tag{3.4}$$

Une particularité d'ACORN est que le bit de rétroaction n'est pas injecté tel quel à l'état interne, il est d'abord combiné avec le bit d'entrée via l'opérateur `XOR`. Cette construction, la première de la sorte, permet d'obtenir la quasi-gratuité de la phase d'authentification. Une itération complète du chiffrement à flot ACORN est décrite par l'algorithme 3.2.

Algorithme 3.2 Fonction d'itération d'ACORN `iter`(S_i, m_i, ca_i, cb_i)

Entrée(s):
État interne S_i
Bit d'entrée m_i
Bits de contrôle ca_i et cb_i
Sortie(s):
État interne mis à jour S_{i+1}
Bit de sortie c_i

1: $S_{i,289} \leftarrow S_{i,289} \oplus S_{i,235} \oplus S_{i,230}$ ▷ Application des 6 fonctions de rétroaction linéaires
2: $S_{i,230} \leftarrow S_{i,230} \oplus S_{i,196} \oplus S_{i,193}$
3: $S_{i,193} \leftarrow S_{i,193} \oplus S_{i,160} \oplus S_{i,154}$
4: $S_{i,154} \leftarrow S_{i,154} \oplus S_{i,111} \oplus S_{i,107}$
5: $S_{i,107} \leftarrow S_{i,107} \oplus S_{i,66} \oplus S_{i,61}$
6: $S_{i,61} \leftarrow S_{i,61} \oplus S_{i,23} \oplus S_{i,0}$
7: $ks_i \leftarrow \kappa(S_i)$ ▷ Génération du pseudo-aléa
8: $c_i \leftarrow ks_i \oplus m_i$ ▷ Chiffrement du bit d'entrée
9: $f_i \leftarrow \Phi(S_i, ks_i, ca_i, cb_i)$ ▷ Génération du bit de rétroaction non-linéaire
10: **pour** j de 0 à 291 **faire**
11: $\quad S_{i+1,j} \leftarrow S_{i,j+1}$ ▷ Décalage d'un cran vers la gauche
12: **fin pour**
13: $S_{i+1,292} \leftarrow f_i \oplus m_i$ ▷ Injection du bit de rétroaction et du bit d'entrée

L'exécution d'ACORN est divisée en quatre phases successives. Les deux bits de contrôle ca_i et cb_i permettent une séparation de domaine entre les phases décrites ci-dessous. Leurs valeurs sont fixes et dépendent du numéro d'itération de l'algorithme.

- **La phase d'initialisation** consiste à appliquer la fonction d'itérations 1792 fois. L'état interne est initialisé à zéro avant d'itérer ce dernier avec les bits de clef et du

vecteur d'initialisation comme décrit par le tableau 3.2. Durant cette phase, le bit de sortie n'est d'aucune utilité étant donné qu'il n'y a aucune donnée à chiffrer.

TABLEAU 3.2 – Valeurs des entrées m_i, ca_i et cb_i durant la phase d'initialisation

	$0 \leq i \leq 127$	$128 \leq i \leq 255$	$i = 256$	$257 \leq i \leq 1791$
m_i	K_i	IV_{128-i}	$K_0 \oplus 1$	$K_{i \pmod{128}}$
ca_i	1	1	1	1
cb_i	1	1	1	1

- **Le traitement des données additionnelles** notées A consiste à appliquer la fonction d'itération à l'état l'interne avec ces données en entrée. L'état interne est mis à jour $|A|+256$ fois, avec les entrées données par le tableau 3.3. Ainsi, la fonction d'itération est exécutée *a minima* 256 fois et ce même en l'absence de données additionnelles à authentifier. Comme pour la phase d'initialisation, le bit de sortie n'est pas à considérer.

TABLEAU 3.3 – Valeurs des entrées m_i, ca_i et cb_i durant le traitement des données additionnelles

	$0 \leq i \leq \lvert A\rvert - 1$	$i = \lvert A\rvert$	$\lvert A\rvert + 1 \leq i \leq \lvert A\rvert + 127$	$\lvert A\rvert + 128 \leq i \leq \lvert A\rvert + 255$
m_{1792+i}	A_i	1	0	0
ca_{1792+i}	1	1	1	0
cb_{1792+i}	1	1	1	1

- **La phase de chiffrement** consiste à appliquer la fonction d'itération à l'état l'interne avec les données à chiffrer notées M en entrée. L'état interne est mis à jour $|M|+256$ fois, avec les entrées données par le tableau 3.4. Ici aussi, la fonction d'itération est exécutée *a minima* 256 fois. C'est la première phase où les bits de sortie sont à considérer (pour les $|M|$ premières itérations) étant donné qu'ils définissent le cryptogramme obtenu.

TABLEAU 3.4 – Valeurs des entrées m_i, ca_i et cb_i durant la phase de chiffrement

	$0 \leq i \leq \lvert M\rvert - 1$	$i = \lvert M\rvert$	$\lvert M\rvert + 1 \leq i \leq \lvert M\rvert + 127$	$\lvert M\rvert + 128 \leq i \leq \lvert M\rvert + 255$
$m_{2048+\lvert A\rvert+i}$	M_i	1	0	0
$ca_{2048+\lvert A\rvert+i}$	1	1	1	0
$cb_{2048+\lvert A\rvert+i}$	0	0	0	0

- **La phase de finalisation** permet de calculer le code d'authentification. Les bits de contrôles sont fixés à 1 tandis que les bits d'entrées sont nuls. La fonction d'itération est appliquée 768 fois et le code d'authentification est défini par les 128 derniers bits de sortie obtenus.

Le processus de déchiffrement est identique, étant donné qu'il suffit d'appliquer l'opérateur `XOR` au pseudo-aléa pour retrouver le texte clair à partir du texte chiffré.

Propriétés d'implémentation Tout comme la plupart des chiffrements à flot opérant bit par bit, ACORN présente d'excellentes propriétés lorsqu'il est implémenté en matériel. À tel point qu'il est le candidat de la compétation CAESAR qui atteint l'implémentation matérielle la plus compacte, et ce avec et sans contremesures contre les attaques par

canaux auxiliaires [DAF+18, FDA+18]. En revanche, l'approche bit par bit est non optimale concernant les implémentations logicielles, pour lesquelles les chiffrements à flot opérant sur des mots ont été proposés (*e.g.*, SOSEMANUK [BBC+08] sur 32 bits et LOISS [FFZ+11] sur 8 bits). Cependant, ACORN reste raisonnablement rapide en logiciel grâce à sa capacité de parallélisation. En effet, le plus petit des 6 LFSR étant constitué de 37 bits, cela signifie qu'il est possible de calculer jusqu'à 37 itérations en parallèle. Ainsi, les plateformes 8-bit, 16-bit et en particulier 32-bit peuvent bénéficier d'implémentations logicielles optimisées d'ACORN afin d'atteindre des performances compétitives.

3.3 Synthèse

Si le terme "cryptographie légère" est couramment utilisé pour désigner une solution cryptographique à faible empreinte destinée à s'exécuter dans un environnement à faibles ressources, son interprétation est sujette à discussion et est fortement liée au contexte. Par exemple, le chiffrement ChaCha20 n'a pas, à première vue, les caractéristiques d'un algorithme cryptographique dit "léger". En effet, ce dernier assure un niveau de sécurité de 256 bits et parce qu'il utilise un état interne d'une taille de 512 bits (*i.e.*, 4 fois celui de l'AES), son exécution nécessite une taille de RAM considérable (*i.e.*, plus d'1 Ko). Cependant, la compacité de son implémentation logicielle et sa résistance intrinsèque aux attaques par analyse de temps de calcul lui permettent d'être plus performant que l'AES-128 sur les plateformes embarquées démunies d'accélérateur cryptographique. Ainsi, en considérant une implémentation de l'AES offrant un même niveau de sécurité (*i.e.*, AES-256) et un temps d'exécution constant (*i.e.*, approche bitslicée) on peut alors considérer que ChaCha20 constitue une alternative légère pour les plateformes non fortement limitées en RAM.

À l'image des trois algorithmes symétriques présentés dans ce chapitre, de nombreuses constructions sont à l'étude par la communauté cryptographique afin de proposer de nouveaux algorithmes de chiffrement dotés de meilleures propriétés d'implémentation. Cependant, on peut identifier des propriétés communes à ces différentes approches, bien qu'elles diffèrent fondamentalement. Par exemple, ni ChaCha, ni ASCON, ni ACORN n'utilisent une étape de dérivation de la clef, contrairement à l'AES et aux chiffrements par blocs conventionnels. Cela permet d'économiser en temps de calcul ou en taille de code selon la façon dont les sous-clefs sont dérivées (*i.e.*, à la volée ou précalculées).

Un point qui n'a pas été abordé dans ce chapitre est la vulnérabilité des algorithmes de cryptographie légère face aux attaques physiques. En effet, ces algorithmes sont destinés aux plateformes embarquées, qui dans le contexte de l'IdO, peuvent se trouver à portée de main de tout un chacun et donc d'un potentiel attaquant. Bien que l'hypothèse d'une proximité physique s'avère inapropriée dans certains cas d'usage (*e.g.*, en domotique, où cela implique une violation de domicile), elle est tout à fait pertinente dans d'autres contextes (*e.g.*, villes intelligentes). L'application des attaques physiques aux objets connectés a été démontrée en pratique à plusieurs reprises [dMS10, OC16]. Notamment, une attaque par observation contre une ampoule connectée en conditions réelles a permis de retrouver la clef de chiffrement utilisée pour les mises à jour logicielles [RSWO17]. Les chercheurs ont ensuite pu installer un logiciel malveillant et le propager à d'autres lampes connectées via le réseau. Ces attaques illustrent la menace que représentent les attaques physiques envers ces algorithmes et par conséquent, le besoin que ces derniers aient la capacité d'intégrer efficacement des contremesures adaptées afin de s'en protéger.

Cette étude se limite à l'étude des attaques par observation, qui sont les moins onéreuses et les plus simples à mettre en pratique. Les canaux auxiliaires considérés par la

suite sont la consommation et le rayonnement électromagnétique, les fuites par temps de calcul étant contrées par les implémentations en temps constant qui sont prises en compte dès la phase de construction, comme l'illustrent les algorithmes présentés dans ce chapitre. Le chapitre suivant définit les outils de cryptanalyse physique ainsi que les contremesures associées, qui seront ensuite utilisés dans l'étude des trois algorithmes détaillés ci-dessus.

Chapitre 4

Attaques par consommation et contremesures

« *Dis-moi ce que tu manges, je te dirai ce que tu es* »

Anthelme Brillat-Savarin

Sommaire

4.1 Introduction

Parmi les attaques exploitant les variations de consommation (et/ou les rayonnements électromagnétiques), on distingue principalement deux types d'attaques. La première désigne les attaques basées sur le flot d'exécution d'une primitive cryptographique. À l'instar des attaques par temps de calcul, elles exploitent les caractères non réguliers d'une implémentation. Un exemple phare est l'attaque contre l'exponentiation rapide, qui est une méthode efficace pour calculer $y = x^e \pmod{n}$. Le principe est de considérer la représentation binaire $e_0 \| e_1 \| ... \| e_l$ de l'exposant e, initialiser y à 1 et pour chaque bit e_i on calcule $y = y^2 \pmod{n}$ si $e_i = 0$ ou $y = y^2 \times x \pmod{n}$ si $e_i = 1$. Ainsi, en observant la consommation du composant lors de ce calcul, il est possible de déduire chaque bit de l'exposant comme illustré par la figure 4.1. Ces attaques sont d'intérêt lorsque l'exposant désigne une valeur sensible (*e.g.*, une clef privée dans le cadre d'un déchiffrement RSA). Ces attaques sont connues sous le nom de *Simple Power Analysis* (SPA) et permettent de déduire des informations simplement en analysant visuellement la consommation de courant.

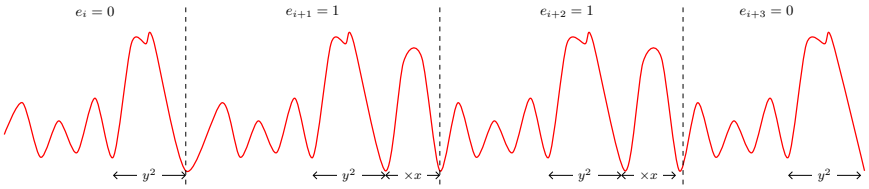


FIGURE 4.1 – Illustration d'une SPA contre l'exponentiation rapide

Tout comme les attaques par temps de calcul, les attaques exploitant les flots d'exécution non constants peuvent être contrées en implémentant une variante régulière de l'algorithme à protéger. La suite de ce chapitre est consacrée à la deuxième catégorie d'attaques par consommation, celles basées sur les données manipulées et qui s'appliquent à tout type d'algorithme cryptographique.

4.2 Attaques sur les données manipulées par l'unité de calcul

4.2.1 Vue d'ensemble

4.2.1.1 Principe

Lors de l'exécution d'une primitive cryptographique, la consommation de courant ainsi que les émanations électromagnétiques du composant sont directement liées aux données qu'il manipule, comme détaillé en section 1.3.2.2. Plus précisément, certaines variables intermédiaires dépendent d'une entrée connue et d'une petite partie de la clef, appelée *sous-clef* et notée k^*. Ces variables intermédiaires sont considérées comme sensibles et peuvent être la cible d'attaquants ayant un accès physique au composant. Le principe est, pour chaque hypothèse de sous-clef k, de prédire la valeur d'une variable intermédiaire qu'elle définit, et ce pour plusieurs exécutions avec des entrées connues et variables. Les enregistrements des fuites physiques produites durant ces exécutions, à l'aide des équipements adéquats (*e.g.*, oscilloscope), sont appelées *traces*. Pour un nombre de traces suffisamment élevé, les prédictions liées à l'hypothèse $k = k^*$ devraient manifester un lien statistique avec les fuites physiques significativement plus élevé que les prédictions pour lesquelles $k \neq k^*$. Ce lien statistique est calculé à l'aide d'un *distingueur*. Selon le distingueur utilisé, il est nécessaire d'appliquer un *modèle de fuite* aux prédictions afin d'établir un lien mathématique entre ces dernières et les fuites physiques. Les attaques

sur les données manipulées sont généralement exécutées avec une approche “diviser-pour-régner” où chaque sous-clef est ciblée indépendamment des autres.

4.2.1.2 Classification

Analyses univariées et multivariées De telles attaques sont catégorisées comme *univariées* lorsqu'elles traitent chaque échantillon t au sein d'une même trace indépendamment les uns des autres et comme *multivariées* lorsque plusieurs échantillons sont combinés afin d'extraire plus d'informations. Parce que les attaques multivariées ciblent généralement les implémentations dotées de contremesures contre les attaques univariées, elles sont présentées dans la section dédiée aux contremesures.

Attaques profilées Indépendamment du caractère univarié ou multivarié, une attaque peut être *profilée* ou non. Les attaques profilées furent introduites par Chari *et al.* [CRR02] et sont considérées comme les plus puissantes car elles modélisent la fuite du composant ciblé au préalable, contrairement aux attaques non profilées qui doivent en général faire une hypothèse générique sur le modèle de fuite du composant. Une attaque profilée se divise en deux étapes principales décrites ci-dessous.

- L'étape de *profilage* a pour but de caractériser les fuites par consommation du composant selon les valeurs qu'il manipule pendant les calculs cryptographiques. Plus précisément, pour chaque valeur v d'une variable intermédiaire dépendant d'une sous-clef k^*, l'attaquant estime la probabilité $P(t|v)$ où t désigne un échantillon de trace. Pour ce faire, il est nécessaire de faire des acquisitions pour différentes valeurs de sous-clef et différentes entrées. Ainsi, cela nécessite un accès ouvert (*e.g.*, possibilité de modifier au choix les clefs de chiffrement, choix des textes à chiffrer) à un composant identique au composant ciblé, ce qui n'est pas toujours réaliste en pratique. De nombreuses techniques peuvent être utilisées pour cette étape de profilage, telles que des attaques par modèle (*template attacks*) [CRR02] ou encore des méthodes d'apprentissage automatique (*machine learning*) [HGDM+11] ou profond (*deep learning*) [MPP16].
- L'étape d'*exploitation* désigne la phase d'attaque. Elle a pour but d'extraire de l'information sur des valeurs manipulées par le composant ciblé, en comparant les fuites à la caractérisation obtenue lors de l'étape de profilage. Pour ce faire, le maximum de vraisemblance est calculé pour chaque hypothèse de clef. Pour une caractérisation suffisamment précise (*i.e.*, effectuée avec un grand nombre de traces), une seule trace peut suffir pour retrouver le secret [PPM17, WH17a].

Attaques à l'aveugle Lorsqu'il est question d'attaques par canaux auxiliaires, il est généralement admis que l'attaquant a connaissance de l'entrée et/ou de la sortie de l'algorithme ciblé. Ainsi, chaque trace est associée à une entrée et/ou une sortie, nécessaire pour prédire la variable intermédiaire ciblée et l'attaque est dite “à textes connus”. Si l'attaquant a recours à des entrées avec des particularités spécifiques (*e.g.*, non uniformément distribuées) alors l'attaque est dite “à textes choisis”. Avoir la main sur les entrées de l'algorithme ciblé peut permettre d'améliorer l'efficacité d'une attaque [VCS10] voire d'introduire de nouveaux vecteurs d'attaque [SWP03]. Cependant, la connaissance du texte d'entrée peut être une hypothèse irréaliste en pratique pour diverses raisons (*e.g.*, normes protocolaires). Les attaques dites “à l'aveugle” [LDLL14, CR17] ont été introduites afin de contourner ces cas de figure. Elles reposent sur le fait que les distributions conjointes $(x, \varphi(k, x))$, où φ désigne une fonction paramétrée par une sous-clef k, peuvent différer selon les valeurs de k . Ainsi, un attaquant peut construire les distributions conjointes pour chaque valeur de k afin de les comparer à celle obtenue par un canal auxiliaire. En

pratique, cela nécessite d'indentifier précisément quels échantillons correspondent aux fuites liées aux valeurs x et $\varphi(k,x)$ respectivement, et ensuite de convertir ces échantillons en valeurs $(n,\varphi(k,x))$. Par conséquent, il est préférable d'appliquer un modèle de fuite M aux distributions conjointes afin de faciliter cette étape. Bien qu'ayant pour but de rendre les attaques par canaux auxiliaires possibles dans des cas pratiques où la connaissance des textes manipulés est irréaliste, l'utilisation de contremesures telles que l'aléatoirisation de l'ordre d'exécution (*cf.* section 4.3.2) compliquent grandement le calcul de la distribution conjointe et rendent les attaques à l'aveugle inefficaces.

La suite de cette section est consacrée aux attaques univariées non profilées. Notre étude se focalise sur ces attaques car ce sont celles qui nécessitent le moins de conditions à satisfaire pour leur mise en pratique.

4.2.2 Fonctions de sélection

Une *fonction de sélection* désigne une fonction qui, à partir d'une hypothèse de sous-clef et d'une entrée connue, définit une variable intermédiaire à cibler. Ces fonctions sont choisies de manière à ce qu'elles soient efficaces d'un point de vue calculatoire, en faisant en sorte que la sous-clef à considérer soit assez petite pour facilement parcourir toutes ses valeurs. Dans le cas des chiffrements par bloc, elles désignent généralement un état interne proche du début ou de la fin du processus de chiffrement et/ou déchiffrement. À titre d'exemple, la fonction de sélection couramment utilisée contre AES cible la sortie de la boîte-S durant le premier tour, comme définie par l'équation 4.1. Étant donné que AES traite chaque octet indépendemment, il est possible d'appliquer une approche diviser-pour-régner en réitérant l'opération pour chaque octet de clef, soit 16 fois.

$$\begin{array}{rccccc} \varphi & : & \mathbb{F}_{2^8} & \times & \mathbb{F}_{2^8} & \rightarrow & \mathbb{F}_{2^8} \\ & & (k & , & x) & \mapsto & S(k \oplus x) \end{array} \tag{4.1}$$

S'il est généralement conseillé de cibler l'état interne de l'AES après l'application de la boîte-S (*i.e.*, $S(k \oplus x)$) plutôt que simplement après l'addition de la clef de tour (*i.e.*, $k \oplus x$), c'est parce que la notion de non-linéarité est une forte valeur ajoutée pour une fonction de sélection [BCO04]. En effet, la propriété de non-linéarité assure une forte distinguabilité entre la bonne et les mauvaises hypothèses de sous-clef.

Plus généralement, le choix d'une fonction de sélection dépend non seulement de l'algorithme ciblé, mais également de son implémentation. Selon ces deux critères, une fonction de sélection peut être complexe dans le sens où les hypothèses à réaliser ne désignent pas une sous-clef telle quelle, mais une combinaison de plusieurs sous-clefs. Dans ce cas, l'attaque par canaux auxiliaires est dite "algébrique" [RS10] puisqu'elle nécessite une étape supplémentaire, qui est de reconstituer la clef à partir des combinaisons de sous-clefs en utilisant des techniques algébriques.

4.2.3 Modèles de fuite

Un modèle de fuite a pour but d'estimer le lien entre des grandeurs observables (les fuites physiques) et des grandeurs internes (les variables intermédiaires). Ces modèles diffèrent selon les architectures matérielles. Dans le cas des fuites par consommation de courant, les modèles de fuites présentés ci-dessous sont les deux principalement utilisés dans la littérature.

4.2.3.1 Distance de Hamming

La distance de Hamming entre deux variables correspond au nombre de bits distincts entre ces deux variables. Ainsi, pour deux variables $x = \sum_{i=0}^{n-1} x_i 2^i$ et $y = \sum_{i=0}^{n-1} y_i 2^i$ définie

sur n bits, leur distance de Hamming est définie par

$$\mathsf{HD}(x,y) = \sum_{i=0}^{n-1} \left(x_i \oplus y_i\right). \tag{4.2}$$

Ce modèle de fuite suppose que la consommation de courant associée à l'inversion de la valeur d'un bit est supérieure à celle associée au mantien d'un bit à sa valeur précédente, ce qui est cohérent avec le fonctionnement d'un inverseur CMOS, illustré en figure 1.2. Ainsi, les modèles basés sur la distance de Hamming sont couramment utilisés en pratique [BCO04, MOP07].

4.2.3.2 Poids de Hamming

Le poids de Hamming d'une variable est défini par le nombre de bits à 1 qu'elle contient. Ainsi, pour un processeur n-bit, le poids de Hamming d'un registre $r = \sum_{i=0}^{n-1} r_i 2^i$ est défini par

$$\mathsf{HW}(r) = \sum_{i=0}^{n-1} r_i\,. \tag{4.3}$$

Le modèle du poids de Hamming suppose que la consommation de courant associée à la manipulation d'une variable intermédiaire par le circuit est proportionnelle au nombre de bits à 1 qu'elle contient, et donc, que la manipulation d'un 1 logique consomme différemment que la manipulation d'un 0 logique. Il est à noter que ce modèle néglige également la consommation de courant associée aux transitions, ce qui est en contradiction avec le fonctionnement d'un inverseur CMOS. Cependant, ce modèle reste pertinent en pratique pour plusieurs raisons. En particulier, sur certains composants, le bus de données est préchargé, c'est-à-dire remis à zéro (ou à son complément) avant chaque écriture [MDH+13]. Dans ce cas, la distance équivaut directement au poids de Hamming du registre r (*i.e.*, $\mathsf{HD}(0,r) = \mathsf{HW}(r)$) et par conséquent l'état de référence ne joue aucun rôle. Ainsi, le modèle basé sur le poids de Hamming est à privilégier dans ces cas de figure, mais également lorsque l'attaquant a peu d'informations sur l'architecture du composant et/ou n'a pas connaissance des valeurs intermédiaire qui précèdent la variable définie par la fonction de sélection.

4.2.4 Distingueurs

Les outils statistiques utilisés pour déterminer l'hypothèse de sous-clef la plus pertinente sont appelés *distingueurs*. La première attaque par observation basée sur les données manipulées [KJJ99] utilise la *différence des moyennes* comme distingueur et de ce fait a popularisé le terme *Differential Power Analysis* (DPA) pour désigner ce type d'attaques. Dans la suite de cette section, différents distingueurs sont présentés dans le cadre d'une attaque contre l'application de la première boîte-S dans AES.

4.2.4.1 Différence des moyennes

Soient $T^1[1\cdots m],\cdots,T^n[1\cdots m]$ des traces de m échantillons correspondant aux fuites de n exécutions d'une implémentation AES traitant respectivement les blocs $B^1,\cdots,B^n$ avec une même clef K. Pour chaque hypothèse de sous-clef k, un vecteur de prédiction $P^k[1\cdots n]$ est construit avec $P^k[i] = \varphi(k, b^i)$ où b^i définit la composante du bloc B^i qui interragit avec la sous-clef k^* afin de définir la variable intermédiaire ciblée.

L'utilisation de la différence des moyennes comme distingueur a pour but de partitionner les traces selon un critère de discrimination appliqué aux prédictions. S'il existe

plusieurs manières de partitionner les traces [MDS99], Kocher *et al.* proposèrent initialement de le faire selon la valeur d'un bit j. Ainsi, les traces sont divisées en deux catégories : celles pour lesquelles $\varphi(k, b^i)_j = 0$ et celles pour lesquelles $\varphi(k, b^i)_j = 1$. Une fois l'ensemble des courbes de courant partitionnées en deux groupes, la trace différentielle $\Delta_k[1 \cdots n]$ est calculée pour chaque hypothèses de sous-clef k en calculant la courbe moyenne de chacun des groupes et en soustrayant l'une à l'autre, comme défini par l'équation 4.4. L'hypothèse de sous-clef associée à la trace différentielle qui exhibe les plus forts biais statistiques (*e.g.*, échantillons minimum et maximum et écart type) définit, en théorie, la bonne sous-clef.

$$\Delta_k[t] = \frac{\sum_{i=1}^{n} \varphi(k, b^i)_j \cdot T^i[t]}{\sum_{i=1}^{n} \varphi(k, b^i)_j} - \frac{\sum_{i=1}^{n} \left(1 - \varphi(k, b^i)_j\right) \cdot T^i[t]}{\sum_{i=1}^{n} \left(1 - \varphi(k, b^i)_j\right)} \tag{4.4}$$

Un des principaux avantages des attaques DPA est qu'elles ne nécessitent pas d'hypothèse forte sur le modèle de consommation en fonction des données. Il suffit que la consommation de courant, liée à la manipulation d'une variable intermédiaire, diffère en moyenne selon la valeur d'un bit de cette variable. En contrepartie, les attaques DPA sont sujettes au problème dit des "pics fantômes" qui désignent des traces différentielles Δ_k avec $k \neq k^*$ qui exhibent des biais statistiques anormalement élevés et qui peuvent perturber l'identification de la bonne hypothèse de sous-clef. Si le bruit lors des acquisitions était initialement considéré comme la source principale de ce phénonmène, il a été observé que les propriétés des boîtes-S ont également une part de responsabilité [CC05].

4.2.4.2 Cœfficient de corrélation de Pearson

Afin de pallier la problématique des pics fantômes, Brier, Clavier et Olivier proposèrent une méthode appelée *Correlation Power Analysis* (CPA) [BCO04]. À l'instar de la DPA, la première étape consiste à calculer, pour chaque hypothèse de sous-clef k, un vecteur de prédiction $P^k[1 \cdots n]$ ciblant une même variable intermédiaire durant n exécutions. Ensuite, ces prédictions de données sont transformées en prédictions de consommations de courant en utilisant un modèle de fuite M (*e.g.*, poids de Hamming de la variable intermédiaire). Finalement, pour chaque hypothèse de sous-clef k, un cœfficient de corrélation de Pearson $\rho^k[t]$ est calculé entre les prédictions de consommation et les traces pour chaque échantillon t, comme défini par l'équation 4.5.

$$\begin{aligned}\rho^k[t] &= \mathsf{Corr}\left(\mathsf{M}\left(P^k[1 \cdots n]\right),\ \left[T^1[t], \cdots, T^n[t]\right]\right) \\ &= \frac{\frac{1}{n}\sum_{i=1}^{n}\left(\mathsf{M}\left(\varphi(k, b^i)\right) - \frac{1}{n}\sum_{j=1}^{n}\mathsf{M}\left(\varphi(k, b^j)\right)\right)\left(T^i[t] - \frac{1}{n}\sum_{j=1}^{n} T^j[t]\right)}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\mathsf{M}\left(\varphi(k, b^i)\right) - \frac{1}{n}\sum_{j=1}^{n}\mathsf{M}\left(\varphi(k, b^j)\right)\right)^2}\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(T^i[t] - \frac{1}{n}\sum_{j=1}^{n} T^j[t]\right)^2}}\end{aligned} \tag{4.5}$$

En théorie, si le nombre de traces est suffisamment élevé, le cœfficient de corrélation maximal est défini par $\rho^{k^*}[x]$ où x désigne un échantillon qui correspond à une fuite liée à la variable intermédiaire ciblée. La méthode d'attaque CPA univariée est résumée par l'algorithme 4.1.

Contrairement à la DPA, l'analyse des fluctuations de consommation liées à la manipuation d'une variable intermédiaire ne repose pas sur un partionnement des traces selon un critère spécifique (*i.e.*, la valeur d'un de ses bits), mais sur la pertinence d'un modèle de fuite appliqué à l'intégralité de la variable intermédiaire prédite. Le fait de tenir compte de tous les bits du mot machine réduit considérablement le risque de pics fantômes, mais également le nombre de traces nécessaires pour que le biais statistique de la bonne hypothèse de sous-clef se distingue significativement des autres.

Ainsi, si une fonction de sélection doit être calculatoirement efficace, il faut également qu'elle corresponde autant que possible à l'architecture de la plateforme et à l'implémentation ciblée. La fonction définie par l'équation 4.1 est particulièrement adaptée aux

Algorithme 4.1 Attaque par analyse de corrélation CPA(φ, M, $T^{1\cdots n}$, $[a,b]$, $B^{1\cdots n}$)

Entrée(s):
Fonction de sélection φ
Modèle de fuite M
Traces $T^{1\cdots n}$
Fenêtre d'échantillons à considérer $[a,b]$
Données en entrée $B^{1\cdots n}$

Sortie(s):
Sous-clef k^*

1: **pour** k de 0 à $|\mathcal{K}|-1$ **faire** ▷ Pour chaque hypothèse de sous-clef
2: **pour** i de 1 à n **faire** ▷ Pour chaque trace
3: $P^k[i] \leftarrow \varphi(k, M^i)$ ▷ Prédiction d'une variable intermédiaire
4: **fin pour**
5: **fin pour**
6: **pour** t de a à b **faire** ▷ Pour chaque échantillon à considérer
7: **pour** k de 0 à $|\mathcal{K}|-1$ **faire** ▷ Pour chaque hypothèse de sous-clef
8: $\rho^k[t] \leftarrow \mathsf{Corr}\left(\mathsf{M}\left(P^k[1\cdots n]\right), \left[T^1[t], \cdots, T^n[t]\right]\right)$
9: **fin pour**
10: **fin pour**
11: **retourner** k^* tel que $\forall k \in \mathcal{K}, \max\left(\rho^{k^*}\right) \geq \max\left(\rho^k\right)$ ▷ Sous-clef la plus probable

plateformes 8-bit dans la mesure où les prédictions désignent des octets qui occupent entièrement les registres et les bus de données. Cependant, elle n'est pas nécessairement optimale pour exploiter les fuites d'un processeur 32-bit qui utilise une implémentation basée sur des tables de précalculs (*cf.* 3.1.1.2). En effet, dans ce cas les prédictions ne constituent qu'un quart (*i.e.*, $\frac{8}{32}$) des variables manipulées et ne sont donc que partiellement corrélées aux fuites enregistrées à un instant précis. Néanmoins, il a été démontré que la prédiction partielle d'une donnée manipulée peut être utilisée pour mener à bien une attaque de type CPA [BCO04, THM+07]. La valeur maximal du coefficient de corrélation est alors

$$\sqrt{\frac{l}{m}} \tag{4.6}$$

où l désigne le nombre de bits prédits et m le nombre de bits total de la variable manipulée.

4.2.4.3 Information mutuelle

L'information mutuelle est un outil de la théorie de l'information qui mesure la dépendance statistique entre deux variables aléatoires. Il est défini par l'équation 4.7

$$I(X,Y) = H(X) - H(X|Y) \tag{4.7}$$

avec

$$H(X) = -\sum_x P(X=x)\log_2 P(X=x) \tag{4.8}$$

l'entropie de X qui décrit la quantité d'information en bits nécessaire pour connaître le comportement de X et

$$H(X|Y) = -\sum_y P(Y=y)\left(\sum_x P(X=x|Y=y)\log_2 P(X=x|Y=y)\right) \tag{4.9}$$

l'entropie conditionnelle qui mesure l'entropie restante sur la variable X sachant Y. Ainsi, $I(X,Y)$ quantifie l'information sur X apportée par l'observation de Y et si $I(X,Y)=0$, alors les deux variables sont indépendantes.

L'utilisation de l'information mutuelle comme distingueur pour les attaques par canaux auxiliaires a été introduite par Gierlichs *et al.* en 2008 [GBTP08] sous le nom d'analyse par information mutuelle (*Mutual Information Analysis*). Le principe est de calculer $I\big(\varphi(k,x),T\big) = H(T) - H\big(T|\varphi(k,x)\big)$ pour chaque hypothèse de sous-clef k. La valeur $k = k^\star$ devrait retourner l'information mutuelle la plus grande, ou en d'autres termes, l'entropie conditionnelle $H\big(T|\varphi(k,x)\big)$ la plus petite. Ainsi, contrairement au cœfficient de corrélation, ce distingueur ne nécessite pas d'hypothèse forte sur le modèle de consommation et permet de mettre en avant des relations non linéaires entre les traces et les variables intermédiaires. Cependant, le calcul de l'entropie conditionnelle nécessite d'estimer la fonction de densité de probabilité de $P\big(T|\varphi(k,x)\big)$ (*e.g.*, à l'aide de B-splines [Ven10]) ce qui est plus coûteux que le calcul du cœfficient de corrélation ou de la différence des moyennes. De plus, pour un modèle de fuite linéaire, il a été démontré que l'analyse par corrélation est plus efficace en pratique [PR09, MMPS09]

4.3 Contre-mesures

Les contre-mesures pour se prémunir des attaques sur les données manipulées peuvent être divisées en deux catégories principales : les contre-mesures de type *hiding*, qui cherchent à dissimuler les fuites, et celles de type *masking*, qui cherchent à rendre les valeurs intermédiaires de l'algorithme indépendantes des fuites observables.

4.3.1 Indépendance entre les fuites et les variables intermédiaires

4.3.1.1 Masquage

Principe La contremesure de masquage, inspirée des techniques de partage de secret [Bla79, Sha79], fut initialement introduite par Chari *et al.* [CJRR99]. Un schéma de masquage à l'ordre d consiste à "masquer" une valeur de l bits en scindant cette dernière en $d+1$ parts aléatoires de l bits, de manière à ce que la totalité de ces $d+1$ parts soit nécessaire pour retrouver la valeur initiale. Pour ce faire, il est nécessaire de générer d masques aléatoires $m_1, \cdots, m_d$. Ainsi, une valeur x est décomposée en $d+1$ parts comme suit

$$x = r_0 \perp \cdots \perp r_d \tag{4.10}$$

avec $r_0 = x * m_1 * \cdots * m_d$ et $r_i = m_i$ pour $1 \leq i \leq d$, où $\perp$ et $*$ désignent deux opérateurs tels que $(a * b) \perp b = a$. La nature d'un schéma de masquage dépend directement de l'opérateur utilisé pour appliquer les masques, et doit être déterminée selon les opérations exécutées par l'algorithme que l'on cherche à protéger. En effet, dans le cas d'une fonction linéaire par rapport à la nature du masquage (*i.e.*, $f(a * b) = f(a) * f(b)$), il suffit d'appliquer cette fonction à chaque part r_i indépendamment (*i.e.*, $f(x) = f(r_0) \perp \cdots \perp f(r_d)$). Cependant, afin de respecter la propriété de confusion énoncée par Shannon, une primitive cryptographique symétrique contient nécessairement une composante non-linéaire. Ces fonctions sont beaucoup plus complexes à implémenter dans le cadre d'un schéma de masquage puisque le traitement indépendant de chaque part ne convient pas.

Opérations non-linéaires sur des valeurs masquées Dans le cadre d'un chiffrement basé sur un réseau de substitution-permutation (*e.g.*, AES), la composante non-linéaire désigne généralement une boîte-S. Pour les implémentations qui utilisent une table de précalcul S pour la couche de substitution, une manière couramment employée est de recalculer une table S' à l'aide d'un masquage booléen (*i.e.*, $\perp = * = \oplus$) telle que

$$S'(x \oplus m_1 \oplus \cdots \oplus m_d) = S(x) \oplus m'_1 \oplus \cdots \oplus m'_d\,. \tag{4.11}$$

La première contre-mesure de ce type fut proposée pour un masquage à l'ordre 1 et consiste à calculer S' pour chaque couple de masques (m_1, m'_1) considéré [CJRR99]. Pour des raisons pratiques, il fut suggéré de fixer le couple (m_1, m'_1) à chaque exécution afin de calculer S' une unique fois au début de l'algorithme [Mes00]. Une adaptation à l'ordre supérieur (*i.e.*, $d > 1$) directement inspirée de cette technique ainsi qu'une de ses variantes [TDSG03] fut proposée pour l'AES [SP06] mais reste générique dans le sens où elle peut s'appliquer à n'importe quelle boîte-S. Cependant, il a été démontré qu'elle est uniquement efficace pour $d \leq 2$ et peut être mise en échec par une attaque multivariée d'ordre 3 [CPR07]. Plus récemment, Coron proposa une nouvelle technique pour calculer une table masquée à n'importe quel ordre d [Cor14]. Selon la structure algébrique de la boîte-S que l'on cherche à protéger, des techniques spécifiques peuvent être proposées comme cela a été largement fait pour l'AES [AG01, OMPR05, RP10]. Cela souligne le fait que le masquage peut en réalité être implémenté à plusieurs niveaux, aussi bien au niveau algorithmique, qu'au niveau des portes logiques élémentaires.

Si les schémas de masquage au niveau algorithmique permettent d'aboutir à des optimisations en termes de temps d'exécution ou de taille d'implémentation, les schémas de masquage qui s'appliquent au niveau des portes logiques constituent une solution générique puisqu'ils peuvent être utilisés pour n'importe quelle implémentation bitslicée ou n'importe quel circuit matériel. Les schémas de masquage au niveau des portes logiques sont naturellement booléens, par conséquent, les difficultés d'implémentation concernent les portes non-linéaires. Seules les portes `AND` sont discutées ci-dessous étant donné que, outre les portes `NOT` pour lesquelles l'application à une seule part suffit (*i.e.*, $\neg(r_0 \oplus \cdots \oplus r_d) = \neg r_0 \oplus \cdots \oplus r_d$), ce sont les seules portes non-linéaires qui seront utilisées dans la suite de ce manuscrit. La première manière de calculer une porte `AND` sur des bits masqués à l'ordre 1 a été introduite par Messerges [Mes00] et est décrite par l'algorithme 4.2.

Algorithme 4.2 Porte `AND` masquée à l'ordre 1 proposée par Messerges [Mes00]

Entrée(s): $(\tilde{x} = x \oplus m_x, m_x)$; $(\tilde{y} = y \oplus m_y, m_y)$
Sortie(s): $(\tilde{z}, m_z)$ tel que $\tilde{z} \oplus m_z = x \wedge y$

1: $\tilde{z} \leftarrow \tilde{x} \wedge \tilde{y}$
2: $m_z \leftarrow (m_x \wedge m_y) \oplus (m_x \wedge \tilde{y}) \oplus (m_y \wedge \tilde{x})$

Dans une publication ultérieure [Tri03], Trichina souligna que cette méthode n'est pas sécurisée car le masque en sortie n'est pas uniformément distribué (*i.e.*, $\mathbf{Pr}(m_z = 0) = \frac{5}{8} \neq \frac{1}{2}$) et proposa une variante qui utilise un bit (pseudo-)aléatoire en guise de nouveau masque.

Algorithme 4.3 Porte `AND` masquée à l'ordre 1 proposée par Trichina [Tri03]

Entrée(s): $(\tilde{x} = x \oplus m_x, m_x)$; $(\tilde{y} = y \oplus m_y, m_y)$; bit (pseudo-)aléatoire r
Sortie(s): $(\tilde{z}, m_z)$ tel que $\tilde{z} \oplus m_z = x \wedge y$

1: $\tilde{z} \leftarrow \big((\tilde{x} \wedge \tilde{y}) \oplus r\big) \oplus (m_x \wedge m_y) \oplus (m_x \wedge \tilde{y}) \oplus (m_y \wedge \tilde{x})$
2: $m_z \leftarrow r$

La même année, Ishai *et al.* publièrent un schéma de masquage pour les portes `AND` qui généralise la méthode de Trichina à n'importe quel ordre [ISW03]. Récemment, Biryukov *et al.* proposèrent une optimisation pour le masquage des portes `AND` à l'ordre 1 [BDLCU18]. En plus d'économiser une opération élémentaire par rapport à l'algorithme 4.3 (*i.e.*, 7 au lieu de 8), cette méthode s'affranchit de la génération de (pseudo-)aléa.

Algorithme 4.4 Porte AND masquée à l'ordre 1 proposée par Biryukov *et al.* [BDLCU18]
Entrée(s): $(\tilde{x} = x \oplus m_x, m_x)$; $(\tilde{y} = y \oplus m_z, m_y)$
Sortie(s): $(\tilde{z}, m_z)$ tel que $\tilde{z} \oplus m_z = x \wedge y$
1: $\tilde{z} \leftarrow (\tilde{x} \wedge \tilde{y}) \oplus (\tilde{x} \vee \neg m_y)$
2: $m_z \leftarrow (m_x \wedge \tilde{y}) \oplus (m_x \vee \neg m_y)$

Le cas particulier des structures ARX Les schémas de masquage pour les structures ARX sont particulièrement complexes étant donné que ces dernières reposent à la fois sur des opérations bit à bit et des additions modulaires. L'utilisation de masques arithmétiques (*i.e.*, $\tilde{x} = x \boxplus m_x$) rend le calcul des additions modulaires immédiat puisqu'il suffit de répéter l'opération sur chaque part. En revanche, il n'existe pas de technique à ce jour pour traiter les opérations bit à bit avec de tels masques. Afin de pallier ce problème, Messerges fut le premier à proposer une technique pour convertir les masques arithmétiques en masques booléens et *vice versa* [Mes00], qui fut rapidement démontrée inefficace [CG00]. Une nouvelle technique de conversion à l'ordre 1 fut ensuite proposée par Goubin [Gou01]. S'en suivirent de nombreux travaux pour proposer de nouvelles techniques plus performantes [CT03, Deb12, CGTV15, WH17b] ou des schémas d'ordre supérieur [HT16, Cor17, BCZ18]. Ainsi, il est nécessaire d'effectuer autant de conversions que le nombre de fois où l'algorithme cryptographique alterne entre opérations booléennes et arithmétiques.

Une autre approche, intialement introduite par Karroumi *et al.* [KRJ14], est d'avoir recours à un algorithme pour calculer l'addition modulaire sur des masques booléens. En ayant recours à un tel mécanisme, appelé "additionneur sécurisé", il n'est plus question de conversions et le schéma de masquage est booléen tout le long de l'exécution de l'algorithme. La méthode proposée par Karroumi *et al.* est directement dérivée de celle de Goubin, et par conséquent, hérite de la même complexité algorithmique, soit $O(n)$ où n où n désigne la taille en bits des opérandes. Par la suite, deux nouvelles méthodes furent proposées, toutes deux bénéficiant d'une complexité logarithmique. Si la méthode de Dinu *et al.* [DGLC17] s'avère plus performante que celle de Coron *et al.* [CGTV15], elle présente l'inconvénient de ne pas s'exécuter en temps constant et de causer des fuites d'information sur la valeur des opérandes via le temps de calcul. Pour cette raison, nous avons préféré utiliser la méthode la moins efficace mais la plus sécurisée pour nos études sur ChaCha.

Si les structures ARX sont particulièrement efficaces en logiciel, l'intégration d'un schéma de masquage pour les prémunir des attaques par observation change radicalement la donne. À titre d'exemple, il a été observé qu'en utilisant un additionneur sécurisé, le temps d'exécution de l'algorithme SPECK était augmenté d'un facteur 23 [CGTV15]. Ainsi, le nombre d'additions modulaires influe directement sur l'impact, en termes de performances, de l'intégration d'un schéma de masquage à une structure ARX.

Limites pratiques Si la contre-mesure de masquage constitue une protection contre les attaques sur les données manipulées, elle doit être implémentée avec précaution afin d'assurer le niveau de sécurité théoriquement annoncé. Cependant, indépendamment de la qualité d'implémentation, les schémas de masquage peuvent être mis en échec par des phénomènes physiques tels que les délais de propagation (*glitches*) dans le cas des implémentations matérielles. Le principe est que si tous les opérandes n'arrivent pas au même moment en entrée d'une porte logique, alors la sortie de cette dernière va subir plusieurs transitions jusqu'à ce que toutes les entrées soient à disposition. Ces transitions peuvent révéler des informations sur les valeurs intermédiaires malgré les masques [MPG05, Bil15] qui peuvent être exploitées dans le but de mettre en échec

le schéma de masquage comme cela a été démontré en pratique à l'encontre d'une implémentation matérielle de l'AES [MPO05]. De nouvelles méthodes d'aléatoirisation des valeurs intermédiaires, appelées "implémentations à seuil" (*threshold implementations*) [NRR06], furent proposer afin d'assurer le niveau de sécurité théorique, et ce malgré de tels phénomènes physiques.

Bien que les implémentations à seuil étaient initialement destinées aux implémentations matérielles afin de se prémunir des transitions dûes aux délais de propagation, elles ont été récemment appliquées aux implémentations logicielles [SBM18, JPS18]. En effet, divers travaux mirent en évidence qu'un schéma de masquage implémenté en logiciel n'assure pas nécessairement le niveau de sécurité désiré, à cause d'optimisations du compilateur [BGRV15] ou de fuites "de transitions" [CGP+12] dûes à la manipulation de la donnée masquée et du masque dans un même bus/registre [BGG+15, dGPdLP+17, PV17].

Ainsi, les implémentations à seuil sont très appréciées car elles constituent une solution générique qui fait gage de sécurité malgré les différents phénomènes énoncés ci-dessus, et ce quelle que soit la plateforme d'exécution. Cependant, elles entraînent des surcharges plus importantes que les schémas de masquage, notamment en termes de temps d'exécution et de quantité de (pseudo-)aléa à générer. Pour ces raisons, les contre-mesures de masquage restent largement utilisées pour les implémentations logicielles, en particulier lorsque l'on cherche à sécuriser une implémentation pour une plateforme donnée, où il est possible d'étudier de caractériser les fuites de celle-ci pour répondre aux attentes sécuritaires.

4.3.1.2 Attaques multivariées

L'ordre d'un schéma de masquage définit directement son niveau de sécurité. Si un schéma de masquage d'ordre $d = 1$ permet théoriquement de se protéger des attaques univariées qui considèrent un seul point de mesure de chaque trace simultanément, elles sont vulnérables aux attaques multivariées d'ordre $d \geq 2$. Ces attaques intègrent généralement une étape de prétraitement des traces qui a pour but de combiner plusieurs points de mesure afin de ramener le problème à une attaque univariée. Par exemple, une fonction de combinaison couramment utilisée pour les attaques d'ordre 2 est la valeur absolue des différences, car pour $a, b \in \{0,1\}$ on a $\mathsf{HW}(a \oplus b) = |\mathsf{HW}(a) - \mathsf{HW}(b)|$. Le principe est donc de calculer de nouvelles traces T' qui combinent tous les échantillons de T et d'appliquer une attaque univariée sur ces traces prétraitées. Cependant, la complexité de l'attaque est beaucoup plus élevée car si les traces T contiennent l échantillons, alors les traces T' en contiennent $\frac{l(l-1)}{2}$. Plus généralement, il a été démontré que la complexité d'une attaque multivariée augmente de manière exponentielle selon l'ordre d [CJRR99].

Ainsi, une implémentation cryptographique ne peut garantir une sécurité absolue face aux attaques physiques. Un expert avec d'importantes ressources à disposition sera tôt ou tard capable d'extraire le secret manipulé. Par conséquent, un ingénieur se doit d'intégrer un schéma de masquage d'un ordre assez élevé pour rendre ces attaques inaccessibles au plus grand nombre, tout en minimisant les impacts d'implémentation. Cependant, même pour des ordres relativement petits (*e.g.*, $d \leq 3$), ces impacts peuvent être démesurés en pratique comme l'illustre [SP06] où l'intégration de schémas de masquage pour $d = 2$ à l'AES entraîne un temps d'exécution 40 à 50 fois supérieur à une version non protégée de cet algorithme. En pratique, les implémentations qui offrent un haut niveau de sécurité combinent généralement des contre-mesures qui cherchent à rendre aléatoires les valeurs manipulées par l'unité de calcul (*e.g.*, masquage) avec des contre-mesures qui cherchent à dissimuler le fuites [HOM06].

4.3.2 Dissimulation des fuites

Contrairement aux techniques de masquages, les contre-mesures de type *hiding* n'influent pas sur les valeurs intermédiaires manipulées par l'unité de calcul. Leur principe est de dissimuler les fuites au sein des traces d'acquisition en diminuant le rapport signal sur bruit. Pour ce faire, il est courant d'utiliser des *techniques de désynchronisation*, qui cherchent à dissimuler les fuites dans le temps. Dans le cas d'une attaque de type CPA, l'attaquant doit calculer, pour chaque vecteur de prédiction (*i.e.*, chaque hypothèse de sous-clef), un cœfficient de corrélation pour chaque point de mesure. Un pic de corrélation pourra alors être observé pour la bonne sous-clef, à l'instant t où la fuite liée à la variable intermédiaire a été enregistrée. Cependant, si la fuite ciblée apparaît à différents instants pour chaque trace, alors l'attaque sera infructueuse. Ainsi, les techniques de désynchronisation visent à exploiter le caractère temporel des attaques par observation, afin de compliquer leur mise en pratique.

Les techniques de désynchronisation peuvent être implémentées aussi bien en matériel, à l'aide d'une horloge instable (*clock jitter*) dont la fréquence varie aléatoirement, qu'en logiciel en exécutant des instructions factices à intevalles irréguliers [CK09]. Ainsi, il est nécessaire d'appliquer un prétraitement aux traces en utilisant des algorithmes de resynchronisation, tels que ceux utilisés par les systèmes de reconnaissance vocale [vWWB11].

Une autre technique de désynchronisation est la contre-mesure "d'ordonnancement aléatoire" (*shuffling*) [VCMKS12]. Elle consiste à exécuter dans un ordre aléatoire des opérations indépendantes les unes des autres (*e.g.*, application de la boîte-S de l'AES à chaque octet de l'état interne). Ainsi, la fuite d'information sur une variable intermédiaire X s'étalera sur n différents échantillons $T[1], \cdots, T[n]$ au fil des exécutions, où n désigne le nombre d'opérations indépendantes considérées (*e.g.*, $n = 16$ dans le cas de la boîte-S de l'AES). Si la distribution est uniforme, alors la probabilité que l'échantillon $T[i]$ contiennent de l'information sur X est de $\frac{1}{n}$ ce qui implique une réduction du rapport signal sur bruit d'un facteur n. Il a été démontré que cette contre-mesure augmente le nombre de traces nécessaires pour mener une attaque d'un facteur n^2 [Man04]. Si le nombre n est limité et dépend directement de l'algorithme et de l'implémentation considérés, il peut s'accroître à l'aide d'instructions factices voire un schéma de masquage, qui augmenteront les possibilités d'ordonnancement [RPD09].

4.4 Synthèse

Les attaques par observation sur les données manipulées sont, de par leur caractère générique, les plus étudiées dans la littérature. Le niveau de sécurité d'une implémentation face à ce type d'attaques n'est jamais absolu et dépend des moyens à disposition de l'attaquant. Ainsi, on peut se poser la question "jusqu'à quel point une implémentation cryptographique embarquée dans un object connecté devrait être protégée des attaques physiques ?". Étant donné la multitude des cas d'usage de l'IdO, il n'existe évidemment pas de réponse unique à cette question. En effet, si certains objets peu sensibles (*e.g.*, balance connectée) pourront se contenter d'utiliser de simples techniques de désynchronisation, des cas d'usage plus critiques (*e.g.*, système d'éclairage public) devront répondre à des éxigences de sécurité plus pointues, comme par exemple, l'obtention d'une certification reconnue (*e.g.*, critères communs [ISO09]) nécessitant une certaine résilience à des attaques avancées. Afin d'étudier des solutions flexibles qui pourraient s'adapter au plus grand nombre de cas d'usage, nos travaux se concentrent sur des attaques univariées et des schémas de masquage à l'ordre 1. Si le niveau de sécurité d'un tel schéma de masquage peut laisser à désirer dans certains cas à fortes contraintes sécuritaires, il apporte

néanmoins un niveau de sécurité satisfaisant pour un bon nombre de cas d'usage de l'IdO [TH08], tout en limitant leur retombée sur les aspects d'implémentation.

Les attaques par observation sur les données manipulées présentées dans la suite de ce manuscrit consistent principalement en la définition de fonctions de sélection pour les algorithmes ACORN, Ascon et ChaCha et à leur évaluation à l'aide de simulations et d'expérimentations pratiques, en utilisant le cœfficient de corrélation de Pearson comme distingueur. Les attaques étudiées sont exclusivement univariées puisque le but de notre démarche est, dans un premier temps, d'évaluer la vulnérabilité des algorithmes susmentionnés face aux attaques sur les données manipulées. Ainsi, nous nous intéressons à des implémentations non protégées pour tenter d'en extraire les secrets et à l'intégration d'un schéma de masquage et de techniques de désynchronisation afin de répondre aux attaques proposées.

Deuxième partie

Implémentations, attaques et contre-mesures

Chapitre 5

L'attaque du poseur de briques

« *Les pierres font partie du chemin.* »

Proverbe roumain

Sommaire

5.1 Attaque par observation à l'encontre de ChaCha

5.1.1 Motivations

ChaCha ayant été conçu pour être particulièrement efficace sur des plateformes embarquées démunies d'accélérateur matériel pour l'AES, il est naturel de s'intéresser à l'application d'attaques physiques à son égard. Lorsque nous avons commencé à nous intéresser à cet algorithme, la seule attaque physique publiée à l'encontre de ChaCha20 était une attaque par injection de fautes [KPB+17]. Nous avons alors décidé d'étudier comment mettre cet algorithme en échec à l'aide d'attaques par observation. En effet, si ChaCha jouit d'une résistance intrinsèque face aux attaques par temps de calcul grâce à sa structure ARX, cela n'apporte aucune garantie de sécurité face aux attaques par analyse de consommation de courant ou de rayonnements électromagnétiques.

5.1.2 Méthodologie

Les attaques présentées dans ce chapitre ont été exécutées en suivant une même méthodologie, qui se décompose en trois grande étapes.

- **Définition d'une fonction de sélection**. Cette étape consiste à analyser la définition mathématique de l'algorithme ciblé afin d'identifier le chemin d'attaque théorique le plus trivial. Cependant, la pertinence d'une fonction de sélection dépend de plusieurs facteurs en pratique, et par conséquent, la meilleure fonction théorique n'est pas nécessairement adaptée à l'implémentation ciblée. Pour cette raison, l'implémentation est également prise en compte pour la définition du chemin d'attaque.
- **Phase de simulation**. Une fois la fonction de sélection définie, son efficacité est évaluée à l'aide de techniques de simulations décrites ci-dessous.
- **Phase d'expérimation pratique**. Une fois l'attaque validée en théorie, cette dernière fait l'objet d'une réalisation expérimentale concrète afin d'aboutir à une preuve de concept et une validation en pratique.

5.1.2.1 Simulations

L'étape de simulation a pour but d'identifier les éventuelles complications que la fonction de sélection pourrait soulever en pratique. Elle consiste à simuler une fuite liée à l'état interne défini par la fonction de sélection, et à exécuter une attaque sur les traces simulées. Pour une fonction de sélection $\varphi(x,k) = y$, on fixe une sous-clef $k^\star$ et un nombre arbitraire d'entrées $x^1, \cdots, x^n$ à considérer afin de générer les états internes cibles $y^i = \varphi(x^i, k^\star)$. Afin de construire les traces correspondantes, il est nécessaire de simuler les fuites liées à ces états internes. Pour ce faire, nous utilisons dans ce manuscrit la fonction $\mathscr{L}$ définie ci-dessous

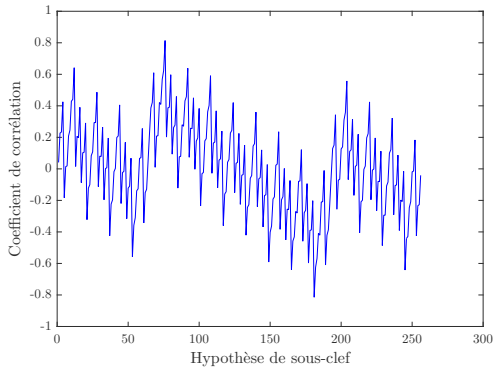
$$\mathscr{L}(y^i) = \mathsf{HW}(y^i) + \epsilon \tag{5.1}$$

où ϵ, qui suit une loi normale standard (*i.e.*, $\epsilon \sim \mathscr{N}(0,\sigma)$), simule le bruit lors des acquisitions. Le choix du poids de Hamming comme modèle de fuite est directement lié à la plateforme utilisée pour nos expérimentations, présentée ci-après. Toutes les simulations présentées par la suite sont effectuées avec $\sigma = 1$.

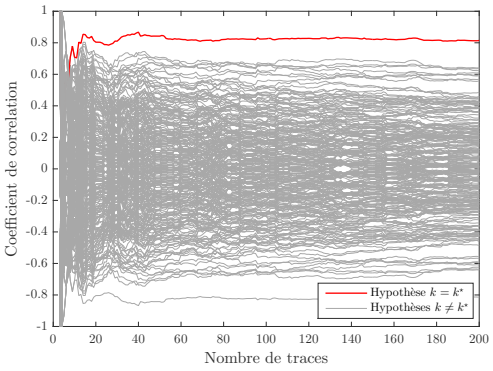
Les principaux avantages qu'apportent une simulation sont l'évaluation de la distinguabilité de la bonne hypothèse de sous-clef, notée δ_φ et définie en équation 5.2, ainsi que l'estimation du nombre de traces nécessaires pour atteindre un critère de distinguabilité maximal.

$$\delta_\varphi = \min_{\forall k \neq k^\star} (\rho^{k^\star} - \rho^k) \tag{5.2}$$

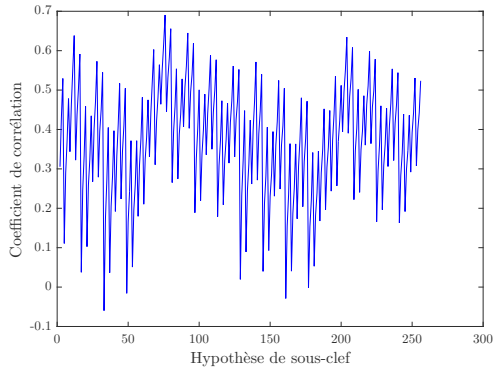
À titre d'exemple, la figure 5.1 illustre des résultats de simulation pour deux fonctions de sélection différentes, toutes deux opérants sur des octets : les opérateurs bit à bit `XOR` et `AND`. Pour ce faire, les entrées x^i ont été choisies aléatoirement. Les figures 5.1a et 5.1c illustrent le coefficient de corrélation correspondant à chaque hypothèse de sous-clef tandis que les figures 5.1b et 5.1d illustrent l'évolution de ces derniers selon le nombre de traces considérées.



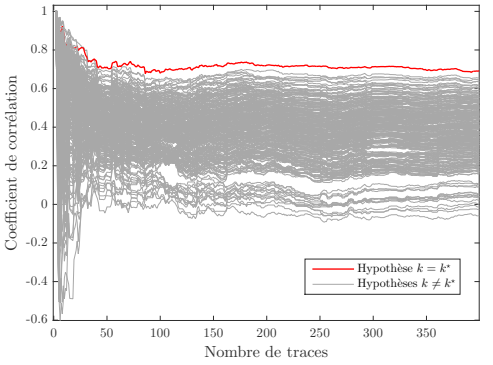
(a) Coefficient de corrélation pour chaque hypothèse de sous-clef $\varphi(x,k) = x \oplus k$



(b) Évolution du coefficient de corrélation $\varphi(x,k) = x \oplus k$



(c) Coefficient de corrélation pour chaque hypothèse de sous-clef $\varphi(x,k) = x \wedge k$



(d) Évolution du coefficient de corrélation $\varphi(x,k) = x \wedge k$

FIGURE 5.1 – Simulations pour les opérateurs bit à bit `XOR` et `AND` avec 400 exécutions

Il est clair que les deux fonctions n'exhibent pas les mêmes propriétés. Par exemple, l'opérateur `XOR` jouit d'un meilleur critère de distinguabilité, qui de plus, nécessite un nombre inférieur de traces pour se démarquer. Par ailleurs, la figure 5.1a met en évidence une symétrie. Cette dernière s'explique par le fait que le coefficient de corrélation d'une sous-clef est égal à l'opposé du coefficient de la négation de cette sous-clef (*i.e.*, $\rho^k = -\rho^{\neg k}$). La généralisation de cette remarque est énoncée par la proposition 1 dont la preuve est donnée en annexe B.1.

Proposition 1. *Soit X une variable aléatoire quelconque. Soient $Y_i = (y_{i,1}, y_{i,2}, ..., y_{i,n})$ et $Y_j = (y_{j,1}, y_{j,2}, ..., y_{j,n})$ deux variables telles que $\forall n,\ y_{i,n} = z - y_{j,n}$ avec $z \in \mathbb{R}$. Alors,* $\mathsf{Corr}(X, Y_i) = -\mathsf{Corr}(X, Y_j)$.

Le corollaire 1.1 découle directement de cette proposition.

Corollaire 1.1. *Pour mener une attaque CPA ciblant le résultat d'un* `XOR` *sur n bits avec $n > 1$, il suffit de considérer des sous-clefs définies sur $n-1$ bits.*

En effet, si $k^\star$ tel que $\max(\rho^{k^\star}) \geq \max_{\forall k}(\rho^k)$ vérifie également $\max(\rho^{k^\star}) \geq \max_{\forall k}(|\rho^k|)$ alors $k^\star$ est la sous-clef la plus probable. Dans le cas contraire, la sous-clef la plus probable est $\neg k^\star$.

Ainsi, en plus d'identifier en amont les difficultés qui peuvent être rencontrées en pratique, une étape de simulation peut permettre de déceler des optimisations selon les propriétés de la fonction de sélection.

5.1.2.2 Matériel d'expérimentation

Plateforme d'exécution La cible utilisée pour nos expérimentations pratiques est un microcontrôleur 32 bits portant la référence STM32F100 [STM16]. Ce dernier est équipé d'un processeur ARM Cortex-M3 cadencé à 24MHz et embarque 128Ko de mémoire non-volatile et 8Ko de RAM. Il est à noter qu'il n'embarque aucune contre-mesure spécifique aux attaques physiques. Ce type de microcontrôleur est un sérieux candidat pour les applications de l'IdO, comme le témoigne le module LoRa CMWX1ZZABZ-091 [Mur18] qui est basé sur un microcontrôleur STM32L072 [STM15] et un émetteur-récepteur LoRa SX1276 [Sem16]. Les requêtes sont effectuées depuis un ordinateur relié au microcontrôleur via un adaptateur USB vers UART, comme illustré par la figure 5.2.

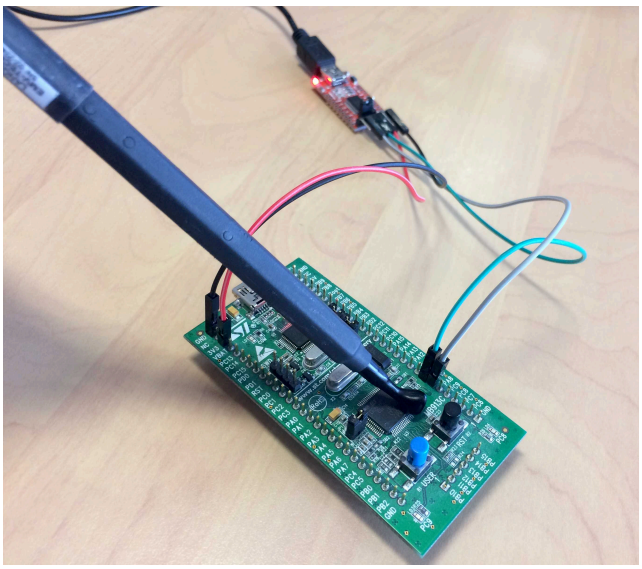
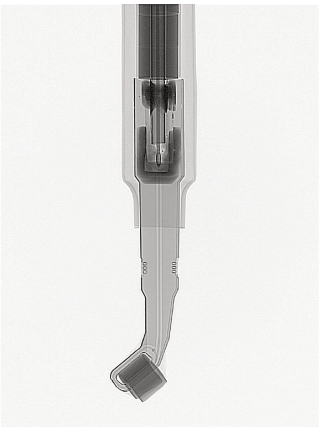
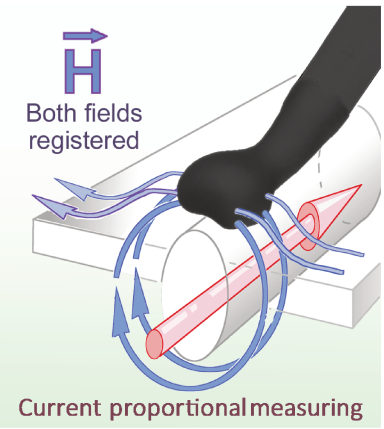


FIGURE 5.2 – Plateforme cible reliée à un ordinateur de contrôle via un adaptateur USB vers UART. Une sonde électromagnétique est positionnée au dessus du processeur.

Outils de mesure Pour ses avantages pratiques énoncés en section 1.3.2.3, nous avons privilégié les émanations électromagnétiques comme canal auxiliaire. La sonde électromagnétique utilisée est une sonde Langer EMV-Technik RF-U 5-2 [Tecb], détaillée par la figure 5.3. Cette dernière est couplée à un amplificateur Langer PA 303 BNC [Teca] qui offre un gain de 30dB. Cet amplificateur est relié à un oscilloscope Lecroy WaveRunner 640Zi [Lec17] qui enregistre les signaux reçus après amplification. Finalement, l'oscilloscope est directement connecté à l'ordinateur de contrôle via un périphérique ethernet afin d'enregistrer les acquisitions au fur et à mesure.



(a) Imagerie par rayons X



(b) Champs électromagnétiques capturés (tiré de [Tecb])

FIGURE 5.3 – Caractéristiques de la sonde Langer EMV-Technik RF-U 5-2

L'acquisition des traces au niveau de l'oscilloscope est paramétrée par un signal de déclenchement (*trigger*) qui permet de se concentrer sur un sous-ensemble de l'algorithme afin de considérer un plus petit nombre d'échantillons. Afin d'avoir un signal de déclenchement stable, une broche de connexion est montée à l'état haut au début de la phase de calcul, puis passée à l'état bas à la fin. Cela nécessite de connecter la dite broche à l'oscilloscope en utilisant une sonde adéquate et surtout l'ajout de lignes de code pour changer l'état de la broche durant l'exécution de l'algorithme. Si cette technique est irréaliste pour un scénario d'attaque en pratique, elle est couramment utilisée dans le cadre des attaques par observation afin d'évaluer ce qu'il est possible de réaliser avec des traces parfaitement alignées. En d'autres termes, il s'agit de se mettre dans des conditions idéales pour considérer les scénarios d'attaques les plus critiques.

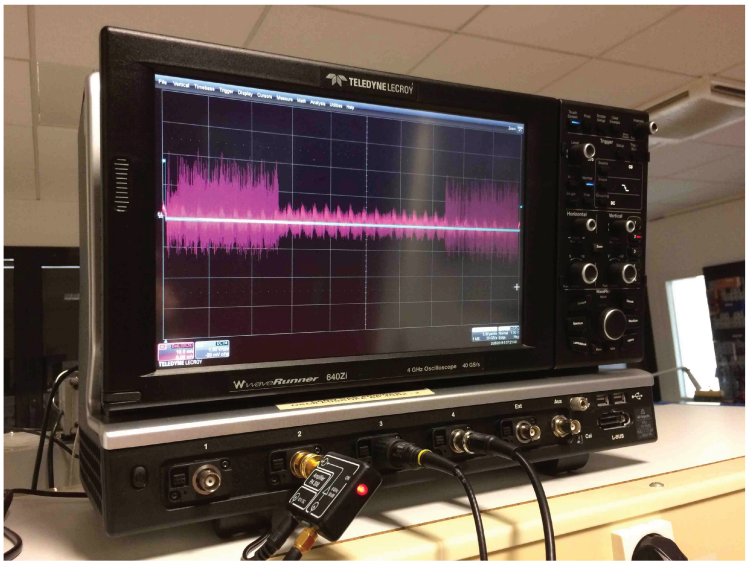
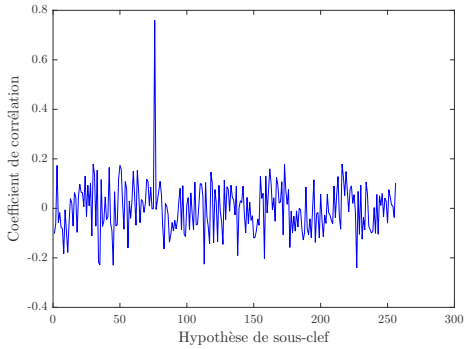


FIGURE 5.4 – Oscilloscope utilisé pour nos expérimentations, relié à l'amplificateur, la sonde électromagnétique et la broche pour le signal de déclenchement

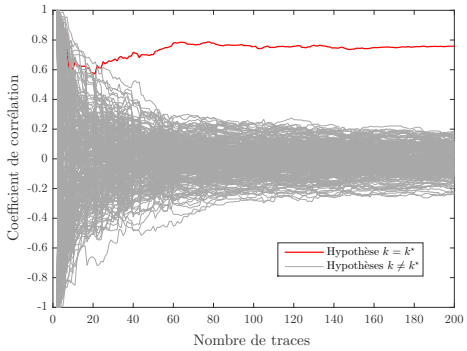
5.1.3 Définition d'un chemin d'attaque

5.1.3.1 Particularités de ChaCha

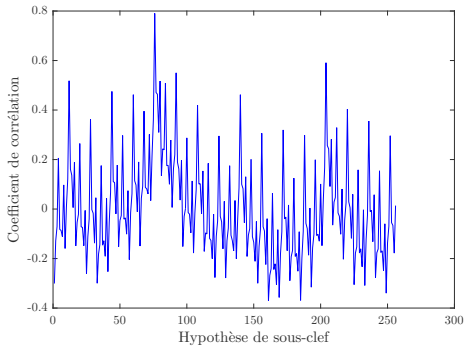
Si les attaques de type CPA ciblent généralement une valeur intermédiaire en sortie de boîte-S pour tirer profit des propriétés de non-linéarité de cette dernière [BCO04], cette astuce ne s'applique pas aux structures ARX où la propriété de confusion est uniquement assurée par les additions modulaires. La non-linéarité de cette opération dans $\mathbb{F}_{2^n}$ est due à la propagation de la retenue. Bien que cela ne constitue pas un candidat aussi pertinent qu'une valeur en sortie d'une boîte-S en termes de distinguabilité comme illustré par la figure 5.5, cibler le résultat d'un addition modulaire est assez efficace pour être utilisé en pratique [LSP04].



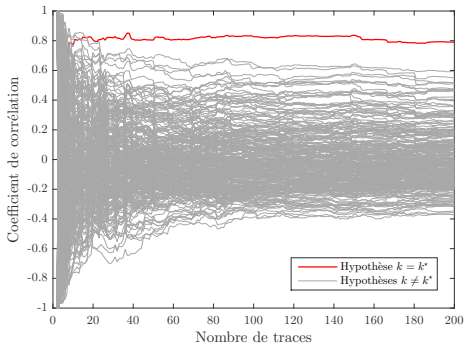
(a) Coefficient de corrélation pour chaque hypothèse de sous-clef $\varphi(x,k) = S(x \oplus k)$



(b) Évolution du coefficient de corrélation $\varphi(x,k) = S(x \oplus k)$



(c) Coefficient de corrélation pour chaque hypothèse de sous-clef $\varphi(x,k) = x \boxplus k$



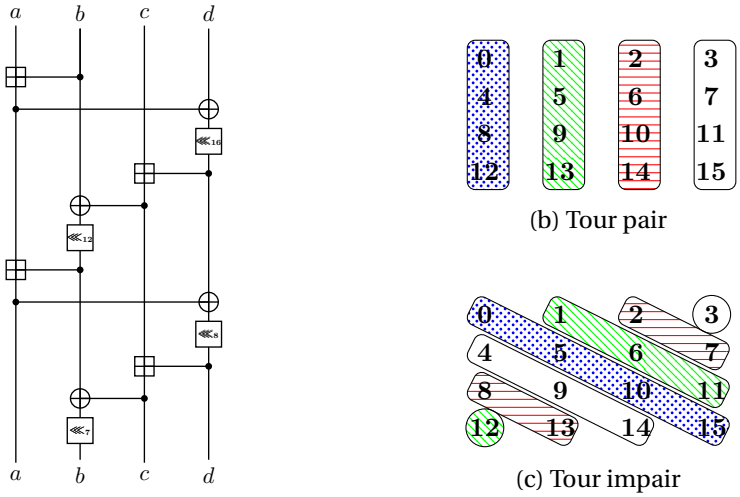
(d) Évolution du coefficient de corrélation $\varphi(x,k) = x \boxplus_8 k$

FIGURE 5.5 – Simulations pour la boîte-S de l'AES et l'addition modulo 2^8 (200 exécutions)

Reste à déterminer une valeur intérmédiaire qui dépend d'une sous-clef et d'une donnée publique afin de définir la fonction de sélection. Pour les chiffrements par bloc, la donnée publique considérée est généralement soit le texte clair, soit le texte chiffré. Dans le cas d'un chiffrement à flot, c'est rarement le cas étant donné que le chiffré est obtenu au travers d'un `XOR` entre le clair et avec le pseudo-aléa de chiffrement. Par conséquent, la seule donnée publique qui interragit directement avec la clef est l'IV.

5.1.3.2 Tirer profit des fuites en début de chiffrement

Pour une meilleure lisibilité, nous rappelons ci-dessous la définition de la fonction de quart de tour et ses applications, ainsi que la structure du bloc initial.



(a) $Q(a,b,c,d)$

(b) Tour pair

(c) Tour impair

La fonction quart de tour et ses applications

'expa'	'nd 3'	'2-by'	'te k'
$\mathtt{k}_0$	$\mathtt{k}_1$	$\mathtt{k}_2$	$\mathtt{k}_3$
$\mathtt{k}_4$	$\mathtt{k}_5$	$\mathtt{k}_6$	$\mathtt{k}_7$
$\mathtt{iv}_0$	$\mathtt{iv}_1$	$\mathtt{iv}_2$	$\mathtt{iv}_3$

Initialisation de l'état interne de ChaCha

Ainsi, durant le premier tour de ChaCha, le $i^{\text{ème}}$ quart de tour ($0 \leq i \leq 3$) opère sur quatre mots de 32 bits :

- une constante $\mathtt{const}_i$
- deux-sous clefs $\mathtt{k}_i$ et $\mathtt{k}_{i+4}$
- un mot d'IV $\mathtt{iv}_i$.

Durant ces quarts de tour, la première (et donc la plus simple au sens calculatoire) valeur intermédiaire qui dépend des deux sous-clefs est définie par

$$\varphi_0^{\mathsf{ChaCha}}\left(\mathtt{iv}_i, \hat{\mathtt{k}}_i \,\|\, \mathtt{k}_{i+4}\right) = \left(\left(\mathtt{iv}_i \oplus \hat{\mathtt{k}}_i\right) \lll 16\right) \boxplus \mathtt{k}_{i+4} \tag{5.3}$$

où $\tilde{\mathtt{k}}_i = k_i \boxplus \mathtt{const}_i$. Cependant, cette fonction de sélection nécessite de cibler deux sous-clefs de 32 bits simultanément, soit 2^{64} hypothèses au total. Puisque pour chacune de ces hypothèses, il est nécessaire de prédire la valeur intermédiaire définie par $\varphi_0^{\mathsf{ChaCha}}$ pour chaque trace, afin de calculer un coefficient de corrélation pour chaque point, il est évident que cela nécessite des moyens calculatoires colossaux. Une alternative est de traiter chaque sous-clef séparément, en ciblant au préalable $\mathtt{iv}_i \oplus \hat{\mathtt{k}}_i$, pour s'attaquer ensuite à $\varphi_0^{\mathsf{ChaCha}}\left(\mathtt{iv}_i, \hat{\mathtt{k}}_i \,\|\, \mathtt{k}_{i+4}\right)$ en connaissance de $\hat{\mathtt{k}}_i$. Ainsi, chaque étape nécessite de traiter 2^{32} hypothèses. S'il est possible de mettre en œuvre une telle attaque en pratique, il est bien plus efficace d'adapter une approche diviser-pour-régner en découpant la variable intermédiaire ciblée en mots de plus petites tailles. C'est exactement la technique qui a été adoptée par les auteurs de la première attaque CPA à l'encontre de ChaCha [JB17], en ciblant des octets pour chacune des deux étapes. Il est à noter que cette attaque a été publiée en parallèle des travaux présentés dans ce chapitre.

L'inconvénient de ce chemin d'attaque est que le XOR, qui a une moins bonne distinguabilité que l'addition modulaire, est utilisé comme fonction de sélection pour la moitié des sous-clefs. Une alternative serait d'étudier un chemin d'attaque visant une valeur intermédiaire en fin d'algorithme. Pour ce faire, il est nécessaire d'avoir connaissance du pseudo-aléa de chiffrement, ou en d'autres termes, du texte clair et du texte chiffré, chose que nous admettons dans la suite de ce chapitre.

5.1.3.3 Tirer profit des fuites en fin de chiffrement

Un attaquant en possession de couples clair/chiffré est capable de calculer le pseudo-aléa de chiffrement correspondant en sommant ces deux valeurs par XOR. Puisque ChaCha$X$ ajoute, via l'addition modulo 2^{32}, l'état interne initial à celui obtenu après l'application des X tours, un attaquant peut cibler la valeur des mots avant l'addition afin de mener une attaque. Ainsi, la fonction de sélection est la même pour chaque sous-clef i et est définie par

$$\varphi_1^{\mathsf{ChaCha}}(x_{i+4}, \mathtt{k}_i) = x_{i+4} \boxminus \mathtt{k}_i \tag{5.4}$$

où x_{i+4} désigne le $(i+4)^{\text{ème}}$ mot du bloc de pseudo-aléa et $\boxminus$ désigne la soustraction modulo 2^{32}.

Les propriétés de la soustraction modulaires d'un point de vue fonction de sélection sont les mêmes que celles présentées en figure 5.5 pour l'addition, étant donné que d'un point de vue binaire, la seule différence est la manière dont la retenue est propagée.

5.1.4 Application pratique

Cette section détaille la mise en pratique d'une CPA à l'encontre de ChaCha en utilisant $\varphi_1^{\mathsf{ChaCha}}$.

5.1.4.1 Fuites exploitables

Si une fonction de sélection dépend en premier lieu de l'algorithme qu'elle cible, elle doit également être en accord avec l'implémentation attaquée. En effet, les fuites de certaines valeurs intermédiaires peuvent être plus faciles à exploiter que d'autres. Considérons le cas des implémentations logicielles sur les architectures de type *load-store*. Pour réaliser des opérations sur des données, ces architectures transfèrent ces données de la mémoire (RAM ou Flash) vers des registres du processeur, effectuent des opérations sur ces registres, pour finalement transférer à nouveau le résultat en mémoire. Les instructions qui servent à transférer les données sont appelées *instructions mémoires* tandis que les instructions opérant sur les registres sont appelées *instructions arithmétiques et logiques*. Plusieurs travaux ont mis en avant que les transferts sur les bus de données fuitent plus d'informations que les opérations de l'UAL [CZP14, MEO15]. Notamment, des expérimentations pratiques sur microcontrôleur 8-bit de type AVR [BDG16] ont démontré que des attaques CPA sur la consommation de courant des opérations de l'UAL n'ont abouti à aucun résultat, contrairement à l'analyse de la consommation de courant des instructions mémoires.

Nous avons voulu vérifier ces phénomènes à l'aide de notre matériel d'expérimentation afin que notre étude soit la plus pertinente possible. Pour ce faire, nous avons considéré deux implémentations différentes de ChaCha : l'une en langage C, tirée de la librairie cryptographique à code source ouvert OpenSSL (version `1.1.0f`), et l'autre écrite en Assembleur ARM par nos soins, qui minimise les accès mémoires.

Le code C tiré d'OpenSSL de la fonction de quart de tour de ChaCha est donné en annexe A.1. Une fois le code intégré à notre microcontrôleur, nous avons compilé ce dernier à l'aide du compilateur GNU ARM C 5.06 avec différentes options d'optimisation (`-O0` et `-O3`) afin d'analyser le code Assembleur généré. Peu importe le niveau d'optimisation spécifié, chaque addition et chaque rotation durant le quart de tour est suivie d'une instruction mémoire `STR` afin de sauvegarder le résultat en RAM comme détaillé dans l'annexe A.2. Par conséquent, il ne devrait pas y avoir de difficultés à appliquer les attaques décrites ci-dessus à cette implémentation.

Afin d'éviter les transferts de données inutiles, notre implémentation Assembleur minimise les instructions mémoires. Pour n'avoir recours à aucune instruction de ce type durant tout le processus de chiffrement, il faudrait charger l'intégralité de l'état interne dans les registres. Bien que ChaCha ne nécessite aucune variable intermédiaire, son état interne est constitué de 16 mots de 32-bits, ce qui nécessiterait 16 registres généraux à disposition. Si le processeur ARM Cortex-M3 embarque bel et bien 16 registres, 3 d'entre eux sont des registres spécialisés (*stack pointer*, *link register* et *program counter*), ce qui laisse seulement 13 registres généraux à disposition. Par conséquent, les transferts de données sont inévitables sur cette plateforme. Vient ensuite la question "jusqu'à quel point peut-on se passer d'accès mémoires?". Comme illustré par la figure 3.4, chaque tour sur les colonnes est suivi d'un tour sur les diagonales et réciproquement. Ainsi, il est indispensable que tous les quarts de tour au sein d'un même tour aient été entièrement exécutés avant de passer au tour suivant. Par conséquent, la principale composante de notre implémentation est un code Assembleur de la fonction de quart de tour qui s'affranchit de tout accès mémoire durant son exécution, comme détaillé en annexe A.2. Bien que ce code ne transfère pas sur le bus de données les valeurs intermédiaires définies par $\varphi_0^{\mathsf{ChaCha}}$, c'est le cas pour les valeurs définies par $\varphi_1^{\mathsf{ChaCha}}$ étant donné que l'état initial est ajouté à l'état courant après le dernier tour. Pour remédier à cela, nous utilisons un code Assembleur différent pour les quarts de tour du dernier tour. En plus de charger la diagonale de l'état interne qui doit être mise à jour par Q, nous chargeons la diagonale correspondante de l'état initial afin de calculer l'addition finale avant d'exécuter l'accès mémoire. En d'autres termes, le code correspond à la fonction Q' définie par

$$Q'(a,b,c,d,x,y,z,t) = Q(a,b,c,d) \boxplus (x,y,z,t) \tag{5.5}$$

où a,b,c,d désignent les mots de l'état interne courant et x,y,z,t les mots de l'état interne initial. Cette méthode nécessite 8 registres généraux, ce qui ne représente aucune difficulté sur notre plateforme de test.

5.1.4.2 Conditions d'attaque

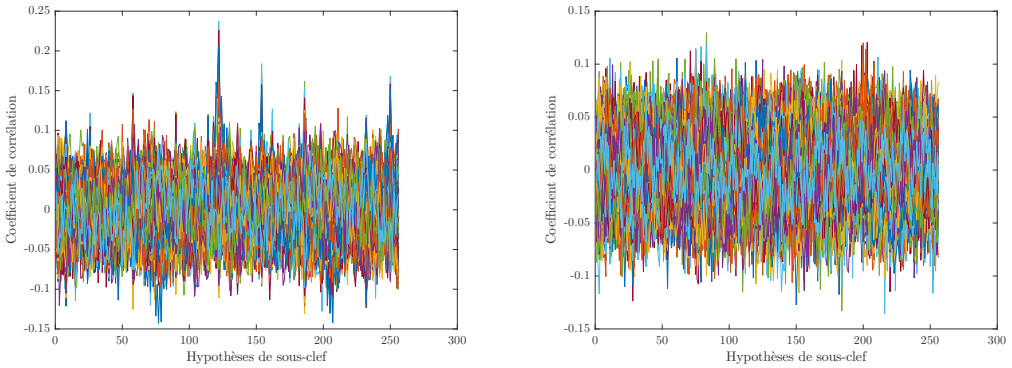
Ces deux implémentations ont été évaluées en pratique dans les conditions décrites comme suit. Le bloc initial ci-dessous a été codé en dur pour les expérimentations. La phase d'acquisition a duré une dizaine de minutes et a consisté à chiffrer 1 000 textes en incrémentant le nonce $\mathtt{iv}_0 = \mathtt{0x00000000}$ à chaque chiffrement. Nous avons ciblé la sous-clef $\mathtt{k}_0 = \mathtt{0xad057876}$ à l'aide de $\varphi_1^{\mathsf{ChaCha}}$, et plus précisément l'octet de poids faible 76.

```
0x61707865   0x3320646e   0x79622d32   0x6b206574
0xad057876   0xe962fc0a   0x42ffc031   0x75018bee
0xb7ae69dc   0xf1490ca8   0x89ac12fd   0xbe8466d3
0x00000000   0xf1d69cbf   0x8e34191d   0x7024af3b
```

FIGURE 5.7 – Bloc initial de ChaCha20 utilisé lors des expérimentations pratiques

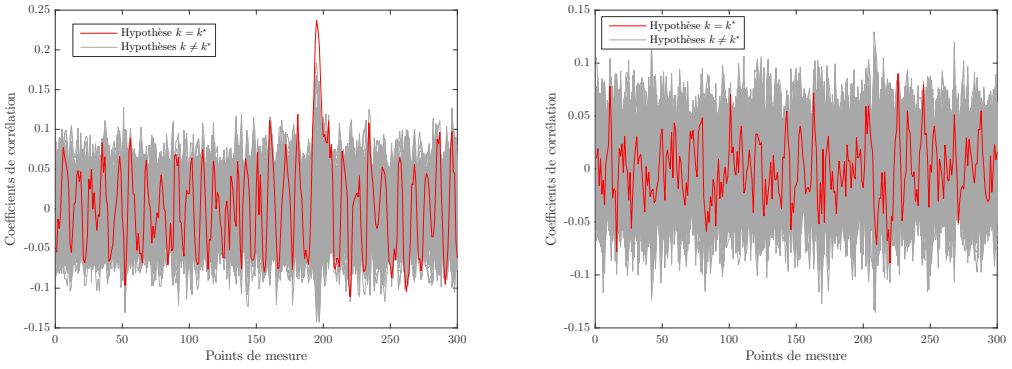
5.1.4.3 Résultats

Concernant le code C compilé, 200 traces suffisent à obtenir un net pic de corrélation pour le code C compilé, tandis que l'attaque est infructueuse pour l'implémentation Assembleur comme illustré par la figure 5.8, et ce même avec 1 000 traces. Sur la figure 5.9 qui illustre les fuites dans le temps, on distingue clairement une fuite d'information pour le code C contrairement au code Assembleur. Ainsi, nous n'avons pas été capables d'extraire la clef pour le code Assembleur, qui s'affranchit d'instructions mémoires manipulant des valeurs intermédiaires qui se prêtent bien aux attaques de type CPA. Dans le doute, nous avons également réitérer l'attaque en considérant un modèle de fuite en distance de Hamming, en vain. Ces résultats viennent conforter l'idée qu'en pratique, les données manipulées par les instructions mémoires sont plus faciles à exploiter que celles manipulées par les opérations de l'UAL.



(a) Code C compilé en `-O3` (b) Code Assembleur ARM

FIGURE 5.8 – Résultats d'expérimentations pour l'exploitation des fuites mémoires : coefficients de corrélation



(a) Code C compilé en `-O3` (b) Code Assembleur ARM

FIGURE 5.9 – Résultats d'expérimentations pour l'exploitation des fuites mémoires : fuites dans le temps

La suite de ce chapitre est consacrée à la définition d'un chemin d'attaque plus complexe afin de mettre en échec l'implémentation Assembleur, qui n'embarque aucune autre contre-mesure que la minimisation astucieuse des transferts sur le bus de données.

5.2 Définition d'un chemin d'attaque avancé

Tout au long de cette section, pour une meilleure lisibilité, on considère que tous les opérateurs sont associatifs à gauche de telle sorte que

$$a \boxplus b \oplus c \lll d \iff (((a \boxplus b) \oplus c) \lll d)\,.$$

5.2.1 Cibler un mot en sortie de quart de tour

Le but de cette section est d'étudier la pertinence d'une fonction de sélection qui définit une valeur en sortie de la fonction de quart de tour.

5.2.1.1 Définition de la fonction de sélection

Lors du quart de tour, chaque mot en entrée est impliqué dans la mise à jour de chaque mot en sortie. De plus, les opérations sont séquentielles dans le sens où le résultat de chacune d'elle impacte toutes les opérations qui suivent. Ainsi, la valeur intermédiaire la plus simple à cibler est le premier mot entièrement mis à jour comme illustré par la figure 5.10.

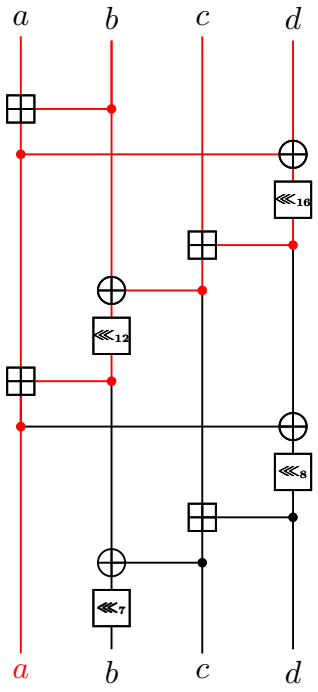


FIGURE 5.10 – Mot ciblé en sortie du quart de tour $Q(a,b,c,d)$

Il est défini par la fonction de sélection

$$\varphi_2^{\mathsf{ChaCha}}(\mathtt{iv}_i, \mathtt{k}_i \parallel \mathtt{k}_{i+4}) = \mathtt{iv}_i \oplus \hat{\mathtt{k}}_i \lll 16 \boxplus \mathtt{k}_{i+4} \oplus k_i \lll 12 \boxplus \hat{\mathtt{k}}_i\,. \tag{5.6}$$

Parce qu'une attaque ciblant l'intégralité du mot défini par φ_2 nécessite de considérer 2^{64} hypothèses de sous-clef, nous avons étudié une approche diviser-pour-régner pour cette fonction de sélection. Étant donné que deux sous-clefs sont impliquées dans cette fonction, on pourrait penser que cibler n bits parmi les 32 nécessiterait de traiter 2^{2n} hypothèses de clefs. En réalité, la complexité est plus élevée à cause des rotations bit à bit. Cela est dû au fait que la sous-clef $\mathtt{k}_i$ agit en tant qu'opérande pour 3 opérations de l'équation 5.6, et qu'une rotation intervient entre chacune de ces opérations. Par conséquent, dans le cas d'une approche diviser-pour-régner, les fenêtres ciblées diffèrent

ce qui nécessite de faire des hypothèses pour chaque opérande définie par $\mathtt{k}_i$. À titre d'exemple, la figure 5.11 illustre l'implication des différentes sous-clef au fil du calcul du quart de tour, lorsque la fenêtre Z de $y = \varphi_2^{\mathsf{ChaCha}}(\mathtt{iv}_i, \mathtt{k}_i \,\|\, \mathtt{k}_{i+4})$, notée y^Z, est ciblée.

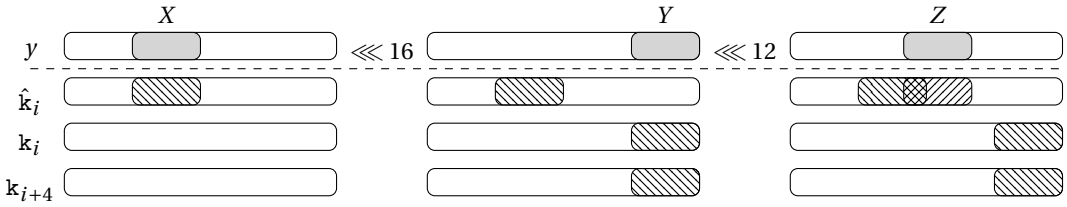


FIGURE 5.11 – Exemple de l'approche diviser-pour-régner sur le quart de tour, $n = 8$

Cette figure illustre le fait que selon la taille de la fenêtre ciblée, il est possible que les hypothèses $\hat{\mathtt{k}}_i^X$ et $\hat{\mathtt{k}}_i^Z$ partagent certains bits. Ainsi, la complexité de l'attaque dépend directement du nombre de bits n ciblés parmi les 32 bits de y^Z, et est définie par

$$|\mathscr{K}| = \begin{cases} 2^{4n}, & \text{si } n \leq 4 \\ 2^{3n+4}, & \text{si } 4 \leq n \leq 12 \\ 2^{2n+16}, & \text{si } 13 \leq n \leq 16 \\ 2^{n+32}, & \text{sinon.} \end{cases} \tag{5.7}$$

En réécrivant la fonction de sélection telle qu'elle est calculée pour chaque hypothèse de sous-clef lors d'une approche diviser-pour-regner, c'est à dire en définissant clairement chaque hypothèse en entrée et en enlevant les rotations, on obtient

$$\varphi_{2,n}^{\mathsf{ChaCha}}\left(\mathtt{iv}_i, \hat{\mathtt{k}}_i^X \,\|\, \mathtt{k}_i^Y \,\|\, \mathtt{k}_{i+4}^Y \,\|\, \hat{\mathtt{k}}_i^Z\right) = \mathtt{iv}_i^X \oplus \hat{\mathtt{k}}_i^X \boxplus \mathtt{k}_{i+4}^Y \oplus \mathtt{k}_i^Y \boxplus \hat{\mathtt{k}}_i^Z . \tag{5.8}$$

5.2.1.2 Phase de simulation

Afin d'étudier la pertinence de $\varphi_{2,n}^{\mathsf{ChaCha}}$, nous avons effectué une phase de simulation telle que décrite en section 5.1.2.1, avec 1 000 `iv` choisis aléatoirement. La simulation a été effectuée avec $n = 2$ afin de traiter un espace de recherche de sous-clef de taille raisonnable, soit 256.

Comme illustré en figure 5.12, le coefficient maximal n'est pas uniquement atteint pour $k^\star$ mais également pour de nombreuses sous-clefs.

Définition 1 (Collision)**.** *Soit $\varphi(x,k)$ une fonction de sélection et $k^\star$ la bonne sous-clef. Une collision est une hypothèse k telle que $\varphi(x,k) = \varphi(x,k^\star)$ pour tout x.*

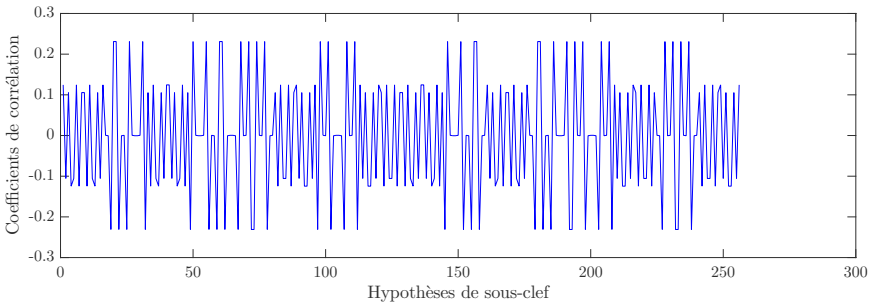


FIGURE 5.12 – Simulation de $\varphi_{2,2}^{\mathsf{ChaCha}}$ avec 1 000 exécutions

En l'occurence, on observe 32 collisions. Plus généralement, la proposition 2 donne une borne supérieure sur le nombre de collisions retournées par $\varphi_{2,2}^{\mathsf{ChaCha}}$, dont la preuve est énoncée en annexe 2.

Proposition 2. *Une attaque à l'aide de $\varphi_{2,n}^{ChaCha}$ retourne jusqu'à $n \cdot 2^{n+2}$ collisions.*

Une difficulté supplémentaire qui n'a pas été mentionnée jusqu'ici est la gêne occasionée par la propagation de la retenue lors des additions modulaires. Excepté lorsque ce sont les bits de poids faibles qui sont ciblés, il est impossible de savoir si une hypothèse de sous-clef est impactée ou non par une retenue. Dans le cas où une retenue ne serait pas prise en compte lors du calcul des prédictions, alors l'attaque retournerait une hypothèse de clef erronée. Par conséquent, la position des fenêtres des hypothèses de sous-clef est primordiale pour la simplicité d'une attaque. Par exemple, la position des fenêtres dans la figure 5.11 a été choisie telle que $\hat{\mathrm{k}}_i^Z$ soit la seule hypothèse qui pourrait être erronée à cause d'une propagation de retenue.

De toute évidence, bien que $\varphi_{2,2}^{\mathsf{ChaCha}}$ permettent d'obtenir des informations sur la clef, une attaque s'appuyant sur cette fonction de sélection serait fastidieuse. Pour cette raison, nous avons étudié l'existence d'un chemin d'attaque plus efficace à partir du pseudo-aléa de chiffrement.

5.2.2 Cibler un mot en entrée de quart de tour

En partant du pseudo-aléa de chiffrement, l'idée est de soustraire les hypothèses de sous-clef afin de calculer les valeurs en entrée du dernier quart de tour (respectivement, en sortie de l'avant dernier). En effet ces valeurs intermédiaires sont manipulées par des accès mémoires et devraient être exploitables.

5.2.2.1 Les avantages du quart de tour inverse

Le quart de tour de ChaCha est facilement inversible, et on note la fonction inverse Q^{-1} dans la suite de ce chapitre. Une propriété intéressante de Q^{-1} est que, contrairement à Q, chaque mot en entrée n'est pas impliqué dans la mise à jour de chaque mot en sortie. En effet, comme illustré par la figure 5.13, a n'impacte pas la mise à jour de b.

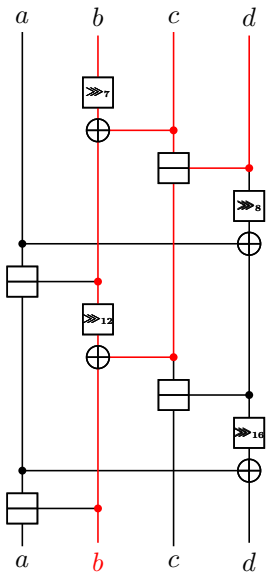


FIGURE 5.13 – Mot ciblé en sortie du quart de tour inverse $Q^{-1}(a,b,c,d)$

Par conséquent, cela constitue la valeur en sortie de Q^{-1} la plus simple à cibler et cette dernière est définie par

$$\varphi_3^{\mathsf{ChaCha}}(b \,\|\, c \,\|\, d \,\|\, \mathtt{iv}_i, \mathtt{k}_b \,\|\, \mathtt{k}_c) = \left(b \boxplus \mathtt{k}_b \ggg 7\right) \oplus \left(c \boxplus \mathtt{k}_c \ggg 12\right) \oplus (c \boxplus \mathtt{k}_c \boxplus d \boxplus \mathtt{iv}_i) \quad (5.9)$$

où b, c, d désignent les mots de pseudo-aléa correspondant à la diagonale du quart de tour attaqué, tandis que $\mathtt{k}_b$ et $\mathtt{k}_c$ désignent les sous-clefs ajoutés à b et c lors de l'addition finale.

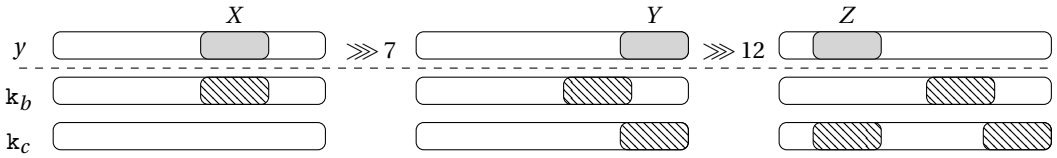


FIGURE 5.14 – Exemple de l'approche diviser-pour-régner sur la quart de tour inverse, $n = 8$

La sous-clef $\mathtt{k}_c$ intervient deux fois en tant qu'opérande sur deux fenêtres différentes. Comme la rotation est sur 12 bits, les hypothèses respectives ne se chevauchent que si la taille de la fenêtre ciblée n est telle que $n > 12$. Par conséquent, l'espace de recherche de clef dépend toujours du paramètre n

$$|\mathcal{K}| = \begin{cases} 2^{3n}, & \text{si } n \leq 12 \\ 2^{2n+12}, & \text{si } 12 \leq n \leq 20 \\ 2^{n+32}, & \text{sinon} \end{cases} \quad (5.10)$$

et la fonction de sélection utilisée pour l'approche diviser-pour-régner est définie par

$$\varphi_{3,n}^{\mathsf{ChaCha}}\left(b \,\|\, c \,\|\, d \,\|\, \mathtt{iv}_i, \mathtt{k}_b^X \,\|\, \mathtt{k}_c^Y \,\|\, \mathtt{k}_c^Z\right) = \left(b^X \boxplus \mathtt{k}_b^X\right) \oplus \left(c^Y \boxplus \mathtt{k}_c^Y\right) \oplus \left(c^Z \boxplus \mathtt{k}_c^Z \boxplus d^Z \boxplus \mathtt{iv}_i^Z\right). \quad (5.11)$$

Puisque chaque hypothèse de sous-clef de $\varphi_{3,n}^{\mathsf{ChaCha}}$ représente un diminuteur, la problématique de propagation des retenues persiste. Cependant, notre scénario d'attaque suppose la connaisance de l'intégralité des diminuendes (*i.e.*, les mots de pseudo-aléa). Ainsi, selon la valeur de chaque diminuende, il est possible de calculer la probabilité qu'une propagation de retenue affecte une fenêtre spécifique. Par exemple, lorsque $\mathtt{k}_b^{[x,x+n[}$ est soustrait à $b^{[0,x[}$, alors la probabilité qu'il faille considérer une retenue pour ce calcul est

$$p = \mathbb{P}\left(\mathtt{k}_b^{[0,x[} > b^{[0,x[}\right) = \frac{2^x - \left(b^{[0,x[} + 1\right)}{2^x}. \quad (5.12)$$

5.2.2.2 Phase de simulation

Nous évaluons dans un premier temps la pertinence de la fonction de sélection à l'aide d'une phase de simulation avec 1 000 exécutions où les mots de pseudo-aléa b, c, d et `iv` sont générés aléatoirement. L'attaque cible les mêmes fenêtres que celles illustrées en figure 5.14, avec $n = 4$, afin que l'hypothèse $\mathtt{k}_c^Y$ ne nécessite pas de calcul de probabilité. Pour $\mathtt{k}_b^X$ et $\mathtt{k}_c^Z$, une retenue est prise en compte si et seulement si $p > \frac{3}{4}$.

Les résultats présentés en figure 5.15 montrent que cette fonction de sélection est moins sujette aux collisions que la précédente, comme énoncé par la proposition 3.

Proposition 3. *Une attaque à l'aide de $\varphi_{3,n}^{ChaCha}$ retourne 4 collisions.*

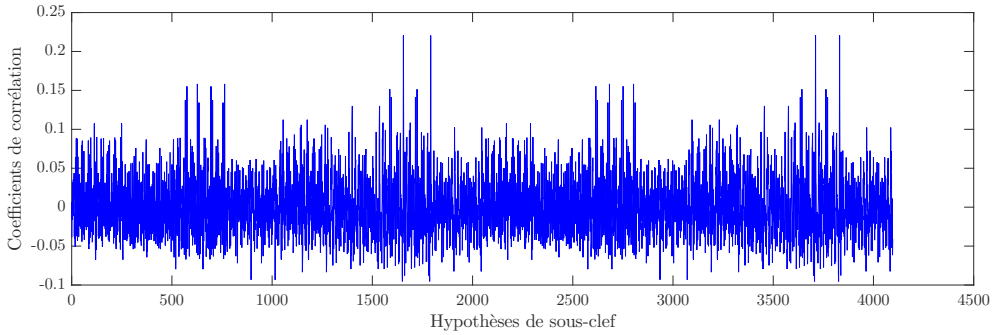


FIGURE 5.15 – Simulation de $\varphi_{3,4}^{\mathsf{ChaCha}}$ avec 1 000 exécutions

Démonstration. La négation du bit de poids fort du diminuende ou du diminuteur entraîne la négation du bit de poids fort de la différence modulaire. Dans le cas de $\varphi_{3,n}^{\mathsf{ChaCha}}$, la négation du bits de poids fort de deux hypothèses de clef simultanément donne le même résultat. On en déduit que le nombre de collisions est $1 + \binom{3}{2} = 4$. □

Comme énoncé par la preuve ci-dessus il est possible de réduire l'espace de recherche de sous-clefs de moitié (*i.e.*, $|\mathcal{K}| = 2^{3n-1}$ pour $n \leq 12$) en ne considérant pas le bit de poids fort d'une hypothèse de sous-clef. En effet, on voit clairement que les coefficient de corrélation se répètent à partir de l'hypothèse de clef $\mathtt{k}_c^Z \parallel \mathtt{k}_c^Y \parallel \mathtt{k}_b^X = (100000000000)_2 = 2\,048$.

En somme, ces expérimentations démontrent qu'inverser les derniers quarts de tour en soustrayant aux mots de pseudo-aléa les hypothèses de sous-clef est beaucoup plus efficace que de cibler les sorties des premiers quarts de tour en début de chiffrement.

5.2.2.3 L'attaque du poseur de briques

Les fonctions de sélection étudiées ci-dessus sont destinées à être employées avec une approche diviser-pour-régner étant donné leur complexité calculatoire. Par conséquent, il est possible de diviser le mot cible en plusieurs fenêtres, d'exécuter des attaques en parallèle sur chacune d'elles, et de finalement faire une recherche exhaustive de la clef en combinant toutes les collisions obtenues.

Plutôt qu'adopter une approche parallèle, nous proposons de favoriser une approche séquentielle où chaque attaque tire parti des résultats de la précédente. Par exemple, après une première attaque à l'aide de $\varphi_{3,n}^{\mathsf{ChaCha}}$ qui a retourné 4 collisions, il est possible d'attaquer $\varphi_{3,m}^{\mathsf{ChaCha}}$ avec $m > n$ pour un espace de recherche de sous-clef $|\mathcal{K}| = 2^{3(m-n)+1}$.

Au delà du fait que la valeur maximale du coefficient de corrélation augmente au fil de l'attaque $\left(m > n \Rightarrow \sqrt{\frac{m}{32}} > \sqrt{\frac{n}{32}}\right)$ et assure une meilleure distinguabilité [THM+07], l'approche séquentielle apporte des avantages supplémentaires. Premièrement, l'estimation de la propagation de retenue est uniquement nécessaire pour la première fenêtre ciblée étant donné que les bits précédents seront pris en compte. Cette propriété est particulièrement intéressante dans le cas d'un scénario d'attaque où la connaissance de couples clairs/chiffrés est irréaliste et donc que $\varphi_{2,n}^{\mathsf{ChaCha}}$ constitue le meilleur chemin d'attaque, étant donné que cette fonction ne bénéficie pas d'une méthode d'estimation de la retenue contrairement à $\varphi_{3,n}^{\mathsf{ChaCha}}$. Notons que les bénéfices d'une approche séquentielle par rapport à la propagation de retenues dans le cadre d'opérations arithmétiques ont dores et déjà été mentionnés dans la littérature [LSP04].

Cependant, cette méthode apporte un avantage supplémentaire pour notre cas d'étude : chaque attaque à l'étape i annule certaines collisions obtenues à l'étape $i-1$. En effet, dans le cas de $\varphi_{3,n}^{\mathsf{ChaCha}}$ où les collisions dépendent uniquement du bit de poids fort des sous-clefs, à l'étape i, les collisions obtenues à l'étape $i-1$ constitueront les bits de poids faibles des hypothèses de sous-clef, permettant à la bonne sous-clef $k^\star$ de se distinguer des collisions. Concernant $\varphi_{2,n}^{\mathsf{ChaCha}}$, la disqualification des mauvaises sous-clefs est plus nuancée étant donné que les collisions sont à l'origine de nombreux cas de figure comme expliqué en preuve 2. Ainsi, l'approche séquentielle ne distingue pas la bonne sous-clef de toutes les collisions mais permet néanmoins de se débarrasser d'une partie d'entre elles. Le nombre de collisions disqualifiées dans ce cas varie selon la valeur des hypothèses de sous-clef et la taille des fenêtres.

Pour profiter de tous les avantages énoncés ci-dessus, nous proposons d'adopter l'approche séquentielle à une attaque CPA ciblant les mots en entrées/sorties de quart de tour et nommons cette méthode "l'attaque du poseur de briques". Ce nom a été inspiré de la figure 5.16, qui illustre un exemple d'application à l'aide de $\varphi_{3,n}^{\mathsf{ChaCha}}$. Il est à noter qu'à l'étape $i=3$, deux hypothèses de sous-clefs au lieu de trois sont considérées, étant donné que la troisième sous-clef à considérer a déjà été ciblée par les précédentes attaques, à cause des rotations. Finallement, lorsque le nombre de bits restant est suffisamment petit, l'attaque est finalisée en ciblant l'intégralité du mot de 32 bits défini par $\varphi_3^{\mathsf{ChaCha}}$.

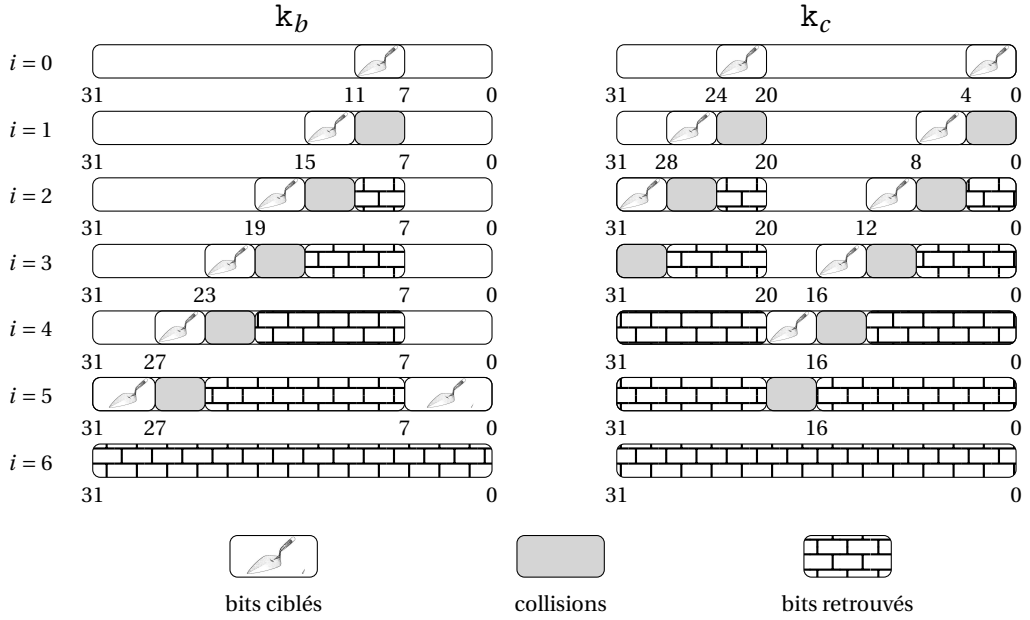


FIGURE 5.16 – Illustration d'une instance de l'attaque du poseur de brique avec $\varphi_{3,n}^{\mathsf{ChaCha}}$. Les indices délimitent les fenêtres attaquées à chaque étape i.

5.2.3 Expérimentation pratique

Afin de valider expérimentalement l'analyse par canaux auxiliaires de l'intégralité du quart de tour de ChaCha, nous avons mis en pratique de l'attaque du poseur de briques illustrée en figure 5.16. L'implémentation testée est le code Assembleur minimisant les accès mémoires, pour lequel l'attaque ciblant les valeurs intermédiaires avant l'addition finale de la clef présentée en section 5.1.4.3 n'avait pas fonctionné. La nouvelle attaque a été réalisée dans les mêmes conditions, mais avec plus de textes étant donné que les fenêtres sont plus petites et que par conséquent, plus de traces sont nécessaires pour que

les collisions se distinguent des autres sous-clefs. Ainsi, nous avons capturé les émanations électromagnétiques pour 4 000 exécutions avec le même bloc d'initialisation (*c.f.* figure 5.7), et toujours en incrémentant le nonce $\mathtt{iv}_0 = \mathtt{0x00000000}$ à chaque chiffrement.

La fonction de sélection utilisée étant $\varphi_{3,n}^{\mathsf{ChaCha}}$, nous avons ciblé un quart de tour diagonal, plus précisément celui hachuré en figure 5.17 qui définit les sous-clefs cibles $\mathtt{k}_2 = \mathtt{42ffc031}$ et $\mathtt{k}_7 = \mathtt{be8466d3}$. Les résultats expérimentaux sont présentés en figures 5.18 et 5.19. Nous détaillons ci-dessous la démarche suivie pour construire les hypothèses de sous-clef et par conséquent, comment interpréter ces résultats.

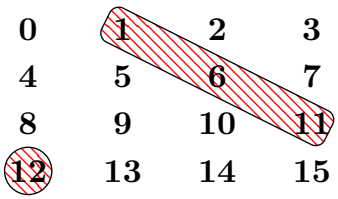


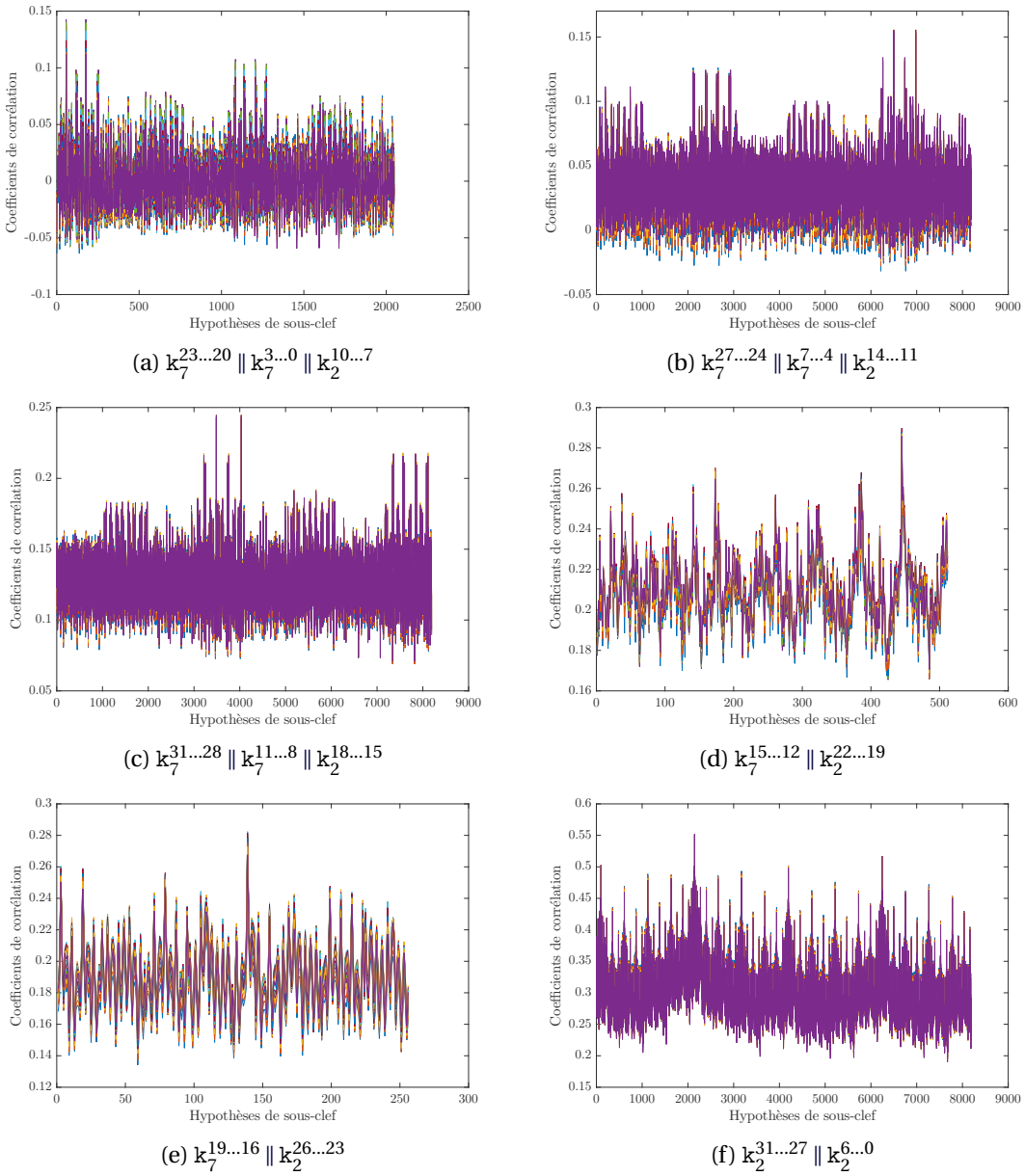
FIGURE 5.17 – Le quart de tour ciblé par notre expérimentation pratique, $\mathtt{k}_b = \mathtt{k}_2$ et $\mathtt{k}_c = \mathtt{k}_7$

Toutes les attaques à l'aide de $\varphi_{3,n}^{\mathsf{ChaCha}}$ ont tiré parti du fait qu'il est possible de réduire de moitié la complexité de l'attaque. Par conséquent, seule la moitié des collisions apparaissent graphiquement. Néanmoins, il suffit de calculer $\gamma \oplus 2^{3n-1}$, avec n la taille de la fenêtre utilisée pour l'attaque, pour chaque collision γ obtenue afin d'obtenir les collisions manquantes. Pour l'étape suivante, chaque hypothèse de sous-clef $k \in \mathscr{K}$ est couplée avec chaque collision $\gamma_j \in \Gamma$ et le résultat est placé dans le vecteur de prédiction à l'indice $k \times |\Gamma| + j$. De cette manière, la bonne sous-clef de l'étape précédente se distinguera des collisions en calculant $(k \times |\Gamma| + j) \bmod |\Gamma|$.

Illustrons cela en détaillant les résultats des deux premières étapes de l'attaque illustrées en figures 5.18a et 5.18b. À l'issue de l'étape $i = 0$, on distingue deux collisions qui sont $\gamma_1^0 = 56$ et $\gamma_2^0 = 176$. On en déduit les quatre collisions qui sont définies par $\Gamma^0 = \{\gamma_1^0, \gamma_2^0, \gamma_3^0, \gamma_4^0\} = \{56, 176, 56 \oplus 2^{11}, 176 \oplus 2^{11}\} = \{56, 176, 2096, 2232\}$. L'attaque à l'étape $i = 1$ prend donc ces quatre collisions en considération. Plus précisément, pour chaque hypothèse $k = \mathtt{k}_7^{27\ldots24} \parallel \mathtt{k}_7^{7\ldots4} \parallel \mathtt{k}_2^{14\ldots11}$ on calcule $k' = \mathtt{k}_7^{27\ldots20} \parallel \mathtt{k}_7^{7\ldots0} \parallel \mathtt{k}_2^{14\ldots7}$ pour chaque $\gamma_j^0 \in \Gamma^0$ comme suit.

$$
\begin{aligned}
\mathtt{k}_7^{27\ldots20} &= \mathtt{k}_7^{27\ldots24} \parallel (\gamma_j^0 \gg 8) \\
\mathtt{k}_7^{7\ldots0} &= \mathtt{k}_7^{7\ldots4} \parallel ((\gamma_j^0 \gg 4) \wedge 15) \\
\mathtt{k}_2^{14\ldots7} &= \mathtt{k}_2^{14\ldots11} \parallel (\gamma_j^0 \wedge 15)
\end{aligned}
$$

Les résultats obtenus pour l'hypothèse k' ne sont placés à l'indice $k \cdot |\Gamma| + j$ dans la matrice de prédiction. À l'issue de l'étape $i = 1$, on obtient deux collisions aux indices 6499 et 6979. Étant donné qu'ils sont tous deux congrus à 3 modulo 4, alors on a la certitude que $\mathtt{k}_7^{23\ldots20} \parallel \mathtt{k}_7^{3\ldots0} \parallel \mathtt{k}_2^{10\ldots7} = \gamma_3^0$. Par conséquent on en déduit que les collisions correspondant à $k = \mathtt{k}_7^{27\ldots24} \parallel \mathtt{k}_7^{7\ldots4} \parallel \mathtt{k}_2^{14\ldots11}$ sont $\gamma_1^1 = 6499 - 34 = 1624$ et $\gamma_2^1 = 6979 - 34 = 1744$. On peut alors recalculer les quatre prédictions définies par Γ^1 et réitérer la procédure décrite ci-dessus pour les fenêtres suivantes.



(a) $k_7^{23...20} \parallel k_7^{3...0} \parallel k_2^{10...7}$

(b) $k_7^{27...24} \parallel k_7^{7...4} \parallel k_2^{14...11}$

(c) $k_7^{31...28} \parallel k_7^{11...8} \parallel k_2^{18...15}$

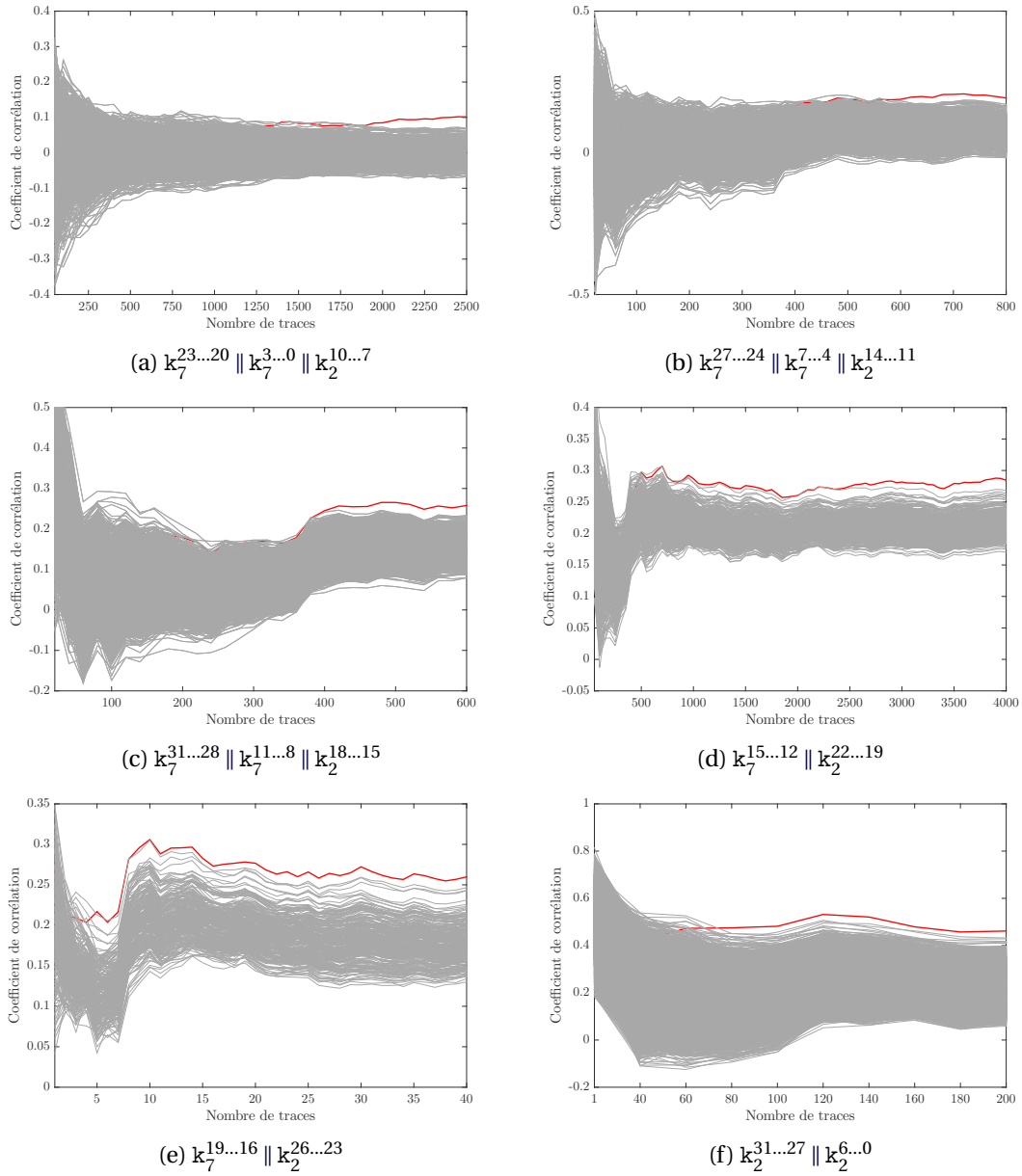
(d) $k_7^{15...12} \parallel k_2^{22...19}$

(e) $k_7^{19...16} \parallel k_2^{26...23}$

(f) $k_2^{31...27} \parallel k_2^{6...0}$

FIGURE 5.18 – Coefficients de corrélation retournés par l'attaque du poseur de briques à l'encontre de k_2 et k_7

Ainsi, nous avons été en mesure de retrouver les sous-clefs k_2 et k_7, et ce malgré le fait que les fuites liées aux valeurs intermédiaires qui ne transitent pas sur les bus de données ne soient pas exploitables avec notre matériel d'expérimentation. D'après la figure 5.19, environ 2000 traces auraient suffi pour mener à bien toutes les étapes de l'attaque. L'attaque du poseur de briques peut être réitérer sur les trois autres quarts de tour afin de retrouver l'intégralité de la clef de chiffrement. La suite de ce chapitre analyse brièvement la complexité que soulève l'intégration de contre-mesures à une implémentation de ChaCha afin de se protéger contre ce type d'attaque.



(a) $k_7^{23...20} \| k_7^{3...0} \| k_2^{10...7}$

(b) $k_7^{27...24} \| k_7^{7...4} \| k_2^{14...11}$

(c) $k_7^{31...28} \| k_7^{11...8} \| k_2^{18...15}$

(d) $k_7^{15...12} \| k_2^{22...19}$

(e) $k_7^{19...16} \| k_2^{26...23}$

(f) $k_2^{31...27} \| k_2^{6...0}$

FIGURE 5.19 – Nombre de traces nécessaire à l'attaque du poseur de briques à l'encontre de k_2 et k_7

5.3 Étude de contre-mesures

Cette section se limite à trois types de contre-mesures : un schéma de masquage, de l'ordonnancement aléatoire, et des contre-mesures protocolaires.

5.3.1 Masquage

La contre-mesure de masquage est générique et s'applique à n'importe quel algorithme. Cependant, le coût de l'intégration d'un schéma de masquage dépend directement de l'algorithme que l'on cherche à protéger. Comme énoncé en section 4.10, les structures ARX se prêtent mal à cette contre-mesure étant donné qu'elles alternent entre opérations arithmétiques et booléennes. Nous avons cependant tenu à évaluer la surcharge, en termes de performances, induite par un schéma de masquage à l'ordre 1 appliqué à ChaCha. Pour ce faire, nous utilisons l'additionneur sécurisé défini dans [CGTV15] qui

bénéficie d'une complexité logarithmique en la taille des opérandes et d'une exécution en temps constant. Ce dernier nécessite $28 \cdot \log_2 n + 4$ opérations où n désigne taille des opérandes en bits, soit 144 pour $n = 32$. De plus, 6 variables intermédiaires sont requises. Par conséquent, il n'est pas possible d'exécuter un quart de tour masqué sans aucun accès RAM, le nombre de données à maintenir dans les registres généraux étant trop importants. Même s'il est fortement conseillé d' implémenter un schéma de masquage au plus bas niveau pour des raisons sécuritaires, notre but est uniquement d'étudier l'impact d'une telle contre-mesure pour le cas spécifique de ChaCha. Par souci de commodité, nous avons décidé d'appliquer le schéma de masquage à l'implémentation C tiré d'OpenSSL évaluée précédemment. Le tableau 5.1 résume les résultats obtenus après compilation en `-O3`. À noter que ces temps d'exécution sont uniquement à titre indactif puisqu'ils ne prennent pas en compte la génération d'aléa, qui n'est pas disponible sur ce microcontrôleur.

TABLEAU 5.1 – Temps d'exécution en cycles d'horloge pour chiffrer un bloc de 512-bit avec ChaCha20 sur ARM Cortex-M3

	Temps d'exécution	Facteur de pénalité
ChaCha20 non masqué	4380	1
ChaCha20 entièrement masqué à l'ordre 1	93993	22
ChaCha20 partiellement masqué à l'ordre 1	21273	5

Pour un schéma de masquage tout du long de l'algorithme, on observe un facteur de dégradation des peformances d'environ 22, et ce sans la génération d'aléa. À titre comparatif, dans le cadre de l'AES, le passage d'une impleméntation naïve à une implémentation régulière et masquée à l'ordre 1 peut être effectué au détriment d'un facteur 13 sur les performances [SS17]. Ainsi, les avantages de ChaCha par rapport à l'AES sont discutables dès lors qu'une contre-mesure de type masquage est nécessaire. Afin de limiter l'impact sur les performances, nous avons étudié si seule la protection des deux derniers et premiers tours constiturait un compromis intéressant. Même en appliquant le masquage à si peu de tours, on obtient un facteur de dégradation avoisinnant 5. De plus, la quantité d'aléa à générer est supérieure étant donné que l'état interne est quatre fois plus grand que celui de l'AES.

En réponse à ces résultats, une récente publication [JPS18] rapporte que les implémentations à seuil constituent une alternative prometteuse pour sécuriser les opérations arithmétiques avec des masques booléens. Si le nombre d'opérations requis par leur méthode est théoriquement 50% inférieur à l'additionneur sécurisé utilisé dans le cadre de cette étude, son application en pratique provoque une dégradation des performances d'un facteur 35 par rapport à leur implémentation de référence, qui est bien supérieur au facteur 22 observé ci-dessus. Les auteurs supposent que cela est dû au fait que leur implémentation de référence est plus efficace que le code *C* compilé en `-O3` sur notre microcontrôleur.

De toute évidence, les contre-mesures qui consistent à rendre les fuites indépendantes des variables manipulées remettent en question la pertinence des structures ARX dans les scénarios où la sécurité physique est souhaitée. Nous nous sommes alors naturellement penchés sur les techniques d'ordonnancement aléatoire qui sont généralement moins coûteuses en termes d'aléà et de temps d'exécution.

5.3.2 Ordonnancement aléatoire

Les techniques d'ordonnancement aléatoire s'appliquent à des fonctions indépendantes les unes des autres. Dans le cadre d'un tour de ChaCha, il est possible d'exécuter les quarts de tour dans n'importe quel ordre. Cependant cela ne constitue pas une contre-mesures très efficace étant donné que pour une valeur intermédiaire v étant normalement manipulée à un instant t, alors la probabilité qu'elle soit manipulée à cet instant sera de $\frac{1}{4}$. D'après les résultats théoriques énoncés en section 4.3.2, multiplier le nombres de traces par $4^2 = 16$ atténuerait suffisamment le bruit généré par l'ordonnancement aléatoire pour mener à bien une attaque. Les opérations au sein d'un quart de tour ne peuvent pas être exécutées aléatoirement étant donné que la modification de l'ordre des opérations affecte le résultat.

Une approche étudiée dans [JB17] est de découper aléatoirement chaque quart de tour en 12 sous-fonctions $f_1^i, \cdots, f_{12}^i$, pour $i = 0, \cdots, 3$, où chaque sous-fonction exécute un sous-ensemble des opérations d'un quart de tour (*i.e.*, $f_{12}^i \circ \cdots \circ f_1^i = Q$). Ainsi, il est possible d'exécuter dans un ordre aléatoire les f_1^i puis les f_2^i et ainsi de suite. Si cette technique permet d'augmenter l'efficacité de la contre-mesure, la dépendance des opérateurs au sein d'un quart de tour biaise la distribution des opérations. En effet, les opérations en début/fin de quart de tour ont une forte probabilité d'être exécutées en début/fin de quart de tour. Par conséquent, les valeurs intermédiaires manipulées par ces opérations sont moins affectées par l'ordonnancement aléatoire et par conséquent, plus facilement attaquables. Cette observation souligne le fait que l'attaque du poseur de briques peut être d'intérêt même lorsque des fuites liées aux valeurs temporaires au sein d'un quart de tour sont accessibles. De plus, puisque les opérandes considérés peuvent différer à chaque opération, cette technique augmente drastiquement les transferts sur les bus de données. Dans ce cas, les fuites de certaines valeurs intermédiaires initiallement inexploitables pourraient alors être ciblées grâce à l'intégration de cette contre-mesure.

Cette première analyse ne laissant pas entrevoir de perspectives particulièrement prometteuses concernant l'application d'une contre-mesure d'ordonnancement aléatoire à ChaCha, nous n'avons pas mener plus d'investigations à ce sujet.

5.3.3 Contre-mesures protocolaires

Dans le cas où les contre-mesures au niveau algorithmique sont trop coûteuses, une alternative est l'intégration de contre-mesures au niveau protocolaire. Nous en donnons ci-après un bref aperçu, bien que la mise en pratique d'une telle approche soit étroitement liée à la flexbilité du protocole, qui est souvent limitée en pratique.

Il est à noter que les deux chemins d'attaque que nous avons étudiés nécessitent la connaissance de l'IV. Ainsi, assurer qu'un attaquant ne peut avoir connaissance de l'IV constituerait une contre-mesure protocolaire. Bien que cette donnée soit considérée comme publique, il existe plusieurs protocoles pour lesquels l'IV est géré de manière confidentielle, comme c'est le cas dans TLS. Depuis la version 1.3, l'IV est généré en interne à partir du secret partagé établi durant l'étape de négociation de clef (*handshake*). Ainsi, à chaque début de session, le client et le serveur ont secrètement négocié une clef symétrique et un IV grâce à leurs paires de clefs asymétriques. Pour garantir l'unicité de l'IV tout au long de la session, le numéro de séquence de chaque message, qui est transmis dans les métadonnées, est ajouté à l'IV via l'opérateur XOR. De cette manière, un attaquant qui espionnerait le trafic d'une session n'aurait pas connaissance de l'IV utilisé. Les concepteurs de la version 1.3 du protocole TLS justifient ce choix par la prévention d'attaques multi-utilisateurs [BT16, HTT18].

Cette contre-mesure permettrait de contrecarrer les attaques de type CPA à l'encontre de ChaCha. Cependant, en théorie, cela ne protège pas des attaques à l'aveugle qui cibleraient les valeurs intermédiaires définies par les fonctions de sélection définies précédemment. En revanche, en combinant cette contre-mesure protocolaire avec un simple ordonnancement aléatoire des quarts de tour, on aboutirait alors à une implémentation résistante à toutes les attaques publiées à l'encontre de ChaCha à ce jour.

5.4 Synthèse

Bien que ChaCha bénéficie d'une résilience intrinsèque aux attaques par temps de calcul, il reste vulnérable par l'analyse d'autres canaux auxilaires, tels que la consommation de courant et le rayonnement électromagnétique. Nous avons démontré cette affirmation avec des expérimentations pratiques. Ces dernières ont mis en évidence le fait que toutes les instructions d'un processeur ne fuitent pas nécessairement de la même manière. En l'occurence, notre banc d'attaque ne nous a pas permi d'exploiter les instructions logiques et arithmétiques s'exécutant sur les registres, mais uniquement les transferts sur les bus de données dus aux instructions mémoires. Ainsi, nous avons été en mesure de développer une implémentation sur ARM Cortex-M3 sur laquelle les attaques publiées à ce jour dans la littérature ne s'appliquent pas, du moins avec le banc d'attaque en question. Suite à ces résultats, des questions subsistent quant à la vulnérabilité des différents types instructions face aux attaques physiques. Par exemple, quelles conditions expérimentales seraient nécessaires pour exploiter les fuites liées aux valeurs intermédiaires manipulées par les instructions de l'UAL ? Faudrait-il établir un modèle de fuite spécifique à ces instructions ? Des attaques profilées seraient-elles plus efficaces ?

Afin d'analyser la possibilité de mettre en échec l'implémentation minimisant les accès mémoires, nous avons dû analyser l'intégralité de la fonction de quart de tour. Étant donné que chacune des entrées affecte chacune des sorties du quart de tour, la fonction de sélection qui en découle pâtit d'une complexité calculatoire considérable. Ainsi, l'approche diviser-pour-régner est indispensable pour exécuter une attaque en pratique. Cependant, notre étude démontre que les rotations bit à bit des structures ARX compliquent ce type d'analyse, étant donné que plusieurs fenêtres d'un même mot de clef peuvent interragir entre elles. Il en résulte des collisions, c'est-à-dire plusieurs sous-clef pour lesquelles le coefficient de corrélation maximal est atteint. Par conséquent, nous avons étudié l'alternative d'attaquer à partir du pseudo-aléa de chiffrement en espérant aboutir à une fonction de sélection plus efficace. Cette variante consiste à remonter les quarts de tour en tirant parti du fait que ces derniers sont inversibles. L'avantage de ce chemin d'attaque est qu'il existe une sortie de Q^{-1} qui ne dépend pas de toutes les entrées. Cette propriété nous a permi de définir une fonction de sélection d'une complexité calculatoire inférieure et moins sujette aux collisions. Finalement, nous proposons une approche globale pour mener l'attaque, nommée "l'attaque du poseur de briques". Elle consiste principalement à mener l'attaque sur différentes fenêtres de manière séquentielle afin d'annuler les collisions en tirant parti des résultats précédents. Nous démontrons finalement la validité de cette attaque en pratique à l'aide d'une expérimentation à l'encontre de l'implémentation minimisant les accès mémoires.

En dernier lieu, nous appliquons à ChaCha une méthode de masquage destinée aux structures ARX afin d'évaluer l'impact d'une telle contre-mesure sur cet algorithme. Les résultats ne sont pas convaincants, étant donné que les performances sont dégradées d'un facteur 22, sans considérer la génération d'aléa, qui peut être un réel défi pour un objet à ressources contraintes. Il semble aussi compliqué d'appliquer un ordonnancement aléatoire efficace. Cependant, l'étude de ce point a souligné que cibler un mot en sortie de

quart de tour est pertinent à l'encontre de cette contre-mesure dans le cadre de ChaCha. Par conséquent, cela laisse entrevoir des applications de l'attaque du poseur de briques à des fins spécifiques, et non uniquement dans le cas où les fuites des valeurs intermédiaires au sein des quarts de tour sont inexploitables.

Ces résultats soulignent le fait que ChaCha n'est probablement pas le candidat idéal pour les plateformes à faibles ressources qui nécessitent une certaine résilience aux attaques physiques. Bien que son implémentation en temps constant soit une propriété intéressante, de nombreux algorithmes de cryptographie légère vérifient cette propriété par construction. C'est pour cette raison que nos travaux de recherche se sont ensuite concentrés sur les deux finalistes de la compétition CAESAR : ACORN et ASCON.

Ces travaux ont donné lieu à la publication *"Bricklayer Attack : A Side-Channel Analysis on the ChaCha Quarter Round"* [AFM17] qui a été présentée à la conférence internationale IndoCrypt 2017.

Chapitre 6

Analyse de ACORN et ASCON

« *Un athlète ne peut arriver en compétition très motivé s'il n'a jamais été mis à l'épreuve.* »

Sénèque

Sommaire

6.1 Introduction

6.1.1 Contexte

En mars 2018, le comité d'évaluation de la compétition annonce ACORN et ASCON comme les chiffrements authentifiés finalistes pour les applications à faibles ressources. Pour rappel, les critères de sélection pour cette catégorie sont les suivants.

- Bénéficier d'une implémentation compacte aussi bien en logiciel qu'en matériel.
- Atteindre un débit compétitif, et en particulier être efficace pour une faible quantité de données à traiter (*e.g.*, paquet de 16 octets).
- Capacité d'intégrer efficacement des contre-mesures contre les attaques par canaux auxiliaires.

Concernant la compacité et l'efficacité des implémentations des différents candidats, de nombreuses études ont été publiées tout au long de la compétition aussi bien d'un point de vue logiciel [AA16] que matériel [KHYK+17, TDSK18]. De plus, des résultats officiels de comparaisons, en termes de performances logicielles, entre les candidats sur différentes plateformes sont disponibles [eBA]. Concernant les implémentations matérielles, différentes études s'accordent sur le fait que ACORN présente des meilleures caractéristiques, en termes de compacité et d'efficacité [KHYK+17, TDSK18]. D'un point de vue logiciel, les résultats sont plus nuancés. En effet, ils varient selon les plateformes et la quantité de données à traiter. Par exemple, [AA16] rapporte que pour des données additionnelles de 5 octets et un message à chiffrer de 1 500 octets, ACORN est plus performant que ASCON sur un processeur basé sur la micro-architecture SandyBridge [Kan15] alors que le résultat est inversé en considérant une micro-architecture SkyLake [Meg16]. D'autre part, les comparatifs officiels rapportent que ASCON est globalement plus efficace que ACORN pour 64 octets de données à chiffrer, mais que la tendance s'inverse lorsque l'on considère 64 octets de données additionnelles supplémentaires.

Plusieurs études ont également été menées au sujet des attaques physiques. Concernant ASCON, une attaque par observations de type DPA a été publiée [SD17], ainsi qu'une implémentation à seuil pour s'en prémunir [GWDE17]. Pour ce qui est des attaques par perturbation, cet algorithme est, par construction, vulnérable aux attaques en fautes statistiques [DEK+16]. Quant à ACORN, si de nombreuses attaques par injection de fautes ont été démontrées à son égard [DRA16, SSMC17, ZFL18], aucune attaque par observations n'a été présentée à ce jour. Le coût d'une protection face à ces attaques a cependant été étudié dans le cadre d'une implémentation matérielle. En effet, [DAF+18] compare plusieurs candidats de la compétition CAESAR (dont ACORN et ASCON) sur un FPGA Spartan-6 dans le cas d'une implémentation non protégée puis pour une implémentation à seuil à l'ordre 1, le but étant d'étudier la surcharge induite pour chaque algorithme. Pour justifier le besoin d'une implémentation à seuil, les auteurs appliquent une méthode de détection de fuites [SM16] à l'implémentation non protégée de chaque algorithme. Il en résulte que seulement ACORN présente les meilleures caractéristiques dans les deux configurations, mais qu'en plus la détection de fuites est plus faible comparé aux autres candidats.

6.1.2 Motivations

À ce jour, le coût de l'intégration de contre-mesures à ACORN et ASCON a été uniquement étudié dans le cadre des implémentations matérielles. Concernant ACORN, le seul résultat qui justifie le besoin de contre-mesures est l'application d'une méthode de détection de fuites à une implémentation matérielle sur Spartan-6. Si un tel test statistique permet de se faire une idée sur la sécurité d'une implémentation, il ne permet pas de

retrouver le secret et n'apporte aucune information sur la complexité d'une attaque en pratique [Sta17]. Pour répondre à ce manque, nous avons décidé de développer la première attaque par observations a l'encontre d'ACORN. Compte tenu des résultats obtenus, nous proposons des schémas de masquage à bas coût pour chacun des deux finalistes et comparons les performances logicielles sur un STM32 muni d'un ARM Cortex-M3 afin d'obtenir des résultats sur des processeurs plus contraints que ceux utilisés pour les précédentes comparaisons. De plus, nous effectuons des mesures, avec et sans contre-mesures, pour de faibles quantités de données afin d'être conforme avec les critères de sélection.

6.2 Vulnérabilités d'ACORN face aux attaques par observations

6.2.1 Identification d'un chemin d'attaque

6.2.1.1 Vue d'ensemble

ACORN repose sur un état interne basé sur des registres à décalage à rétroaction. Ainsi, contrairement à ChaCha qui opère par bloc de données de 64 octets et qui manipule la clef pour chacun de ces blocs, ACORN peut traiter jusqu'à 2^{64} bits de données pour un même IV et ce en utilisant uniquement la clef et l'IV dans le but d'initialiser l'état interne. Dans le cadre d'une attaque par observations contre de telles structures, il n'y a généralement pas d'autre option que de cibler cette phase d'initialisation [GBC+08, Str10]. En effet, la spécification d'ACORN mentionne que la clef ne peut-être retrouvée à partir de l'état interne une fois la phase d'initialisation terminée, cette dernière étant non réversible et suffisamment complexe (*"the secret key of ACORN cannot be recovered easily from the state since the initialization is now non-invertible"* [Wu16]). Cette phase d'initilisation étant exécutée à chaque fois qu'un paquet doit être chiffré et/ou authentifié, elle peut être exploitée par une attaque de type CPA.

6.2.1.2 Zoom sur la phase d'initialisation

En synthétisant les informations contenues dans le tableau 3.2, on peut réécrire la phase d'initialisation en pseudo-code comme suit, où `iter` fait référence à la fonction d'itération d'ACORN définie par l'algorithme 3.2.

Algorithme 6.1 Phase d'initialisation de l'état interne d'ACORN

Entrée(s):
Clef K
Vecteur d'initialisation IV
Sortie(s):
État interne initialisé S_{1792}

1: $(S_{0,0}, \cdots, S_{0,292}) \leftarrow (0, \cdots, 0)$ ▷ Initialisation de l'état interne à 0
2: **pour** i de 0 à 127 **faire**
3: $S_{i+1} \leftarrow \texttt{iter}(S_i, K_i, 1, 1)$ ▷ Mise à jour de l'état interne avec K_i en entrée
4: **fin pour**
5: **pour** i de 128 à 255 **faire**
6: $S_{i+1} \leftarrow \texttt{iter}(S_i, IV_{i-128}, 1, 1)$ ▷ Mise à jour de l'état interne avec IV_{i-128} en entrée
7: **fin pour**
8: $S_{256} \leftarrow \texttt{iter}(S_{255}, K_0, 1, 1)$
9: **pour** i de 257 à 1791 **faire**
10: $S_{i+1} \leftarrow \texttt{iter}(S_i, K_{i \bmod 128}, 1, 1)$
11: **fin pour**

Les 128 premières mises à jour de l'état interne ne peuvent pas être la cible d'une attaque CPA puisque leurs exécutions sont constantes au fil des exécutions. En revanche, les itérations prenant en entrée les bits IV_i manipulent des valeurs qui dépendent à la fois de la clef et de l'IV. La figure 6.1 illustre l'évolution de l'état interne durant les 256 premières itérations. Au sein de cette fonction d'itération décrite par l'algorithme 3.2, le chiffrement du bit d'entrée (*i.e.*, $c_i \leftarrow ks_i \oplus m_i$) ainsi que l'injection du bit de rétroaction (*i.e.*, $S_{i+1,292} \leftarrow f_i \oplus m_i$) constituent les deux valeurs qui peuvent théoriquement être la cible d'une attaque. Cependant, le chiffrement est en réalité inutile durant la phase d'initialisation et peut donc être ignoré, laissant l'injection du bit de rétroaction comme la seule opération d'intérêt à cibler.

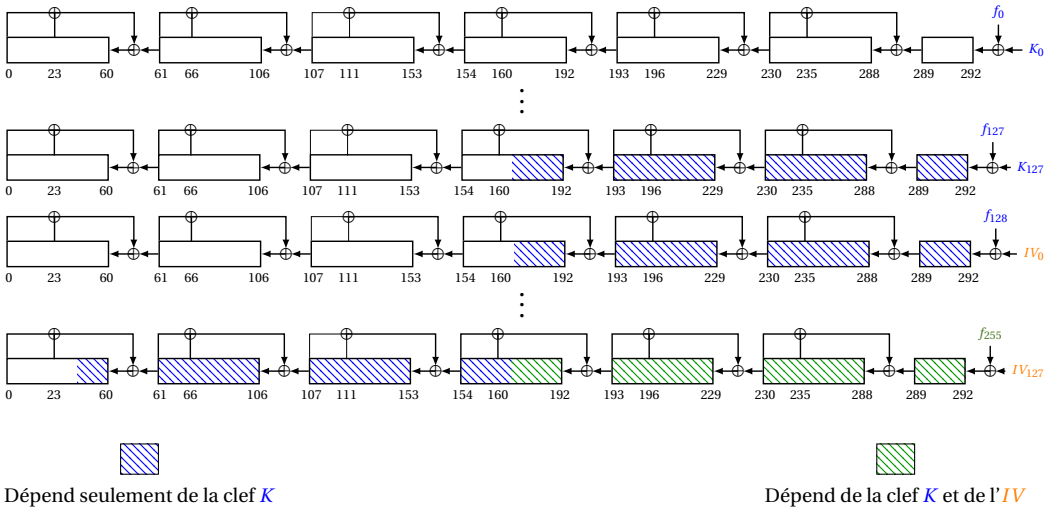


FIGURE 6.1 – Interaction du bit de rétroaction non-linéaire avec les bits d'entrée durant les 256 premières étapes de l'initialisation d'ACORN

6.2.2 Étude théorique de l'exploitation des fuites liées à l'injection du bit de rétroaction

6.2.2.1 Identification des difficultés

Si une attaque CPA ciblant le calcul de $S_{i+1,292} \leftarrow f_i \oplus m_i$ avec $i = 128, \cdots, 255$ et $m_i = IV_{i-128}$ est théoriquement possible, elle ne permet pas de directement retrouver la clef K. En effet, puisque les bits ks_i et $S_{i,196}$ sont utilisés pour calculer f_i durant la phase d'initialisation (*i.e.*, $ca = cb = 1$), chaque f_i est en réalité composé de plusieurs bits de K. Ainsi, une attaque par observations à l'encontre d'ACORN est inévitablement algébrique. À titre d'illustration, après les 128 premières itérations de l'algorithme, l'état interne est

$$\begin{aligned}
(S_{128,0}, \cdots, S_{128,164}) &= (0, \cdots, 0) \\
(S_{128,165}, \cdots, S_{128,198}) &= (\neg K_0, \cdots, \neg K_{33}) \\
(S_{128,199}, \cdots, S_{128,201}) &= (K_{34} \oplus K_0, \cdots, K_{36} \oplus K_2) \\
(S_{128,202}, \cdots, S_{128,218}) &= (\neg K_{37} \oplus K_3 \oplus K_0, \cdots, \neg K_{53} \oplus K_{19} \oplus K_{16}) \\
(S_{128,219}, \cdots, S_{128,223}) &= (K_{54} \oplus K_{20} \oplus K_{17} \oplus K_0, \cdots, K_{58} \oplus K_{24} \oplus K_{21} \oplus K_4) \\
(S_{128,224}, \cdots, S_{128,229}) &= (\neg K_{59} \oplus K_{25} \oplus K_{22} \oplus K_5 \oplus K_0, \cdots, \neg K_{64} \oplus K_{30} \oplus K_{27} \oplus K_{10} \oplus K_5) \\
(S_{128,230}, \cdots, S_{128,261}) &= (\neg K_{65} \oplus K_{11} \oplus K_6, \cdots, \neg K_{96} \oplus K_{42} \oplus K_{37}) \\
(S_{128,262}, \cdots, S_{128,272}) &= (K_{97} \oplus K_{43} \oplus K_{38} \oplus f_{97}, \cdots, K_{107} \oplus K_{53} \oplus K_{48} \oplus f_{107}) \\
(S_{128,273}, \cdots, S_{128,288}) &= (\neg K_{108} \oplus K_{54} \oplus K_{49} \oplus K_0 \oplus f_{108}, \cdots, \neg K_{123} \oplus K_{69} \oplus K_{64} \oplus K_{15} \oplus f_{123}) \\
(S_{128,289}, \cdots, S_{128,292}) &= (\neg K_{124} \oplus f_{124}, \cdots, \neg K_{127} \oplus f_{127})
\end{aligned} \tag{6.1}$$

où les 128 premiers bits de rétroaction non-linéaires sont définis par

$$f_i = \begin{cases} 1 & \text{si } 0 \le i \le 96 \\ K_{i-97} & \text{si } 97 \le i \le 99 \\ (\neg K_{i-58}) \wedge (\neg K_{i-100}) \oplus K_{i-97} & \text{if } 100 \le i \le 111 \\ (K_{i-58} \oplus K_{i-112}) \wedge (\neg K_{i-100}) \oplus K_{i-97} & \text{si } 112 \le i \le 116 \\ \neg(K_{i-58} \oplus K_{i-112} \oplus K_{i-117}) \wedge (\neg K_{i-100}) \oplus K_{i-97} & \text{si } 117 \le i \le 127. \end{cases} \tag{6.2}$$

Une difficulté supplémentaire est que f_i n'est plus constant pour une certaine clef lorsque $i \ge 177$. En effet, le bit $f_{128} \oplus IV_0$ injecté à l'état interne à l'étape $i = 128$ se retrouve en position $S_{177,244}$ et intervient donc dans le calcul de f_{177} d'après la définition de Φ_i en équation 3.3. L'attaque décrite ci-dessus sera par conséquent inefficace pour $i \ge 177$ étant donné que la valeur de f_i variera au fil des exécutions. Reste à savoir quelles informations il est possible d'obtenir à partir des f_i pour $128 \le i \le 176$. Afin d'établir quelle combinaison de bits de clef définit chaque f_i, nous avons programmé une version "texte" d'ACORN qui opère sur des valeurs textuelles au lieu de valeurs numériques (*i.e.*, $(\neg$`'a'` $\oplus$ `'b'`$) \wedge$ `'c'` $=$ `'(!(a) ^ b)& c'`). Nous avons pu alors exprimer le système d'équations booléennes $\mathscr{F}_{[128,176]}$ composé des f_i pour $128 \le i \le 176$.

$$\mathscr{F}_{[128,176]} = \begin{cases}
(\neg K_{70} \oplus K_{11} \oplus K_{16}) \wedge \neg K_{28} \oplus K_{31} = f_{128} \\
(\neg K_{71} \oplus K_{12} \oplus K_{17}) \wedge \neg K_{29} \oplus K_{32} = f_{129} \\
(\neg K_{72} \oplus K_{13} \oplus K_{18}) \wedge \neg K_{30} \oplus K_{33} = f_{130} \\
(\neg K_{73} \oplus K_{14} \oplus K_{19}) \wedge \neg K_{31} \oplus K_{34} \oplus \neg K_0 = f_{131} \\
(\neg K_{74} \oplus K_{15} \oplus K_{20}) \wedge \neg K_{32} \oplus K_{35} \oplus \neg K_1 = f_{132} \\
(\neg K_{84} \oplus K_{25} \oplus K_{30}) \wedge \neg K_0 \oplus (\neg K_{75} \oplus K_{16} \oplus K_{21}) \wedge \neg K_{33} \oplus K_{36} \oplus \neg K_2 = f_{133} \\
(\neg K_{85} \oplus K_{26} \oplus K_{31}) \wedge \neg K_1 \oplus (\neg K_{76} \oplus K_{17} \oplus K_{22}) \wedge (K_{34} \oplus K_0) \oplus K_{37} \oplus K_3 \oplus K_0 = f_{134} \\
(\neg K_{86} \oplus K_{27} \oplus K_{32}) \wedge \neg K_2 \oplus (\neg K_{77} \oplus K_{18} \oplus K_{23}) \wedge (K_{35} \oplus K_1) \oplus K_{38} \oplus K_4 \oplus K_1 = f_{135} \\
(\neg K_{87} \oplus K_{28} \oplus K_{33}) \wedge \neg K_3 \oplus (\neg K_{78} \oplus K_{19} \oplus K_{24}) \wedge (K_{36} \oplus K_2) \oplus K_{39} \oplus K_5 \oplus K_2 = f_{136} \\
(\neg K_{88} \oplus K_{29} \oplus K_{34}) \wedge \neg K_4 \oplus (\neg K_{79} \oplus K_{20} \oplus K_{25}) \wedge (K_{37} \oplus K_3 \oplus \neg K_0) \oplus K_{40} \oplus K_6 \oplus K_3 = f_{137} \\
(\neg K_{89} \oplus K_{30} \oplus K_{35}) \wedge \neg K_5 \oplus (\neg K_{80} \oplus K_{21} \oplus K_{26}) \wedge (K_{38} \oplus K_4 \oplus \neg K_1) \oplus K_{41} \oplus K_7 \oplus K_4 = f_{138} \\
(\neg K_{90} \oplus K_{31} \oplus K_{36}) \wedge \neg K_6 \oplus (\neg K_{81} \oplus K_{22} \oplus K_{27}) \wedge (K_{39} \oplus K_5 \oplus \neg K_2) \oplus K_{42} \oplus K_8 \oplus K_5 = f_{139} \\
(\neg K_{91} \oplus K_{32} \oplus K_{37}) \wedge \neg K_7 \oplus (\neg K_{82} \oplus K_{23} \oplus K_{28}) \wedge (K_{40} \oplus K_6 \oplus \neg K_3) \oplus K_{43} \oplus K_9 \oplus K_6 = f_{140} \\
(\neg K_{92} \oplus K_{33} \oplus K_{38}) \wedge \neg K_8 \oplus (\neg K_{83} \oplus K_{24} \oplus K_{29}) \wedge (K_{41} \oplus K_7 \oplus \neg K_4) \oplus K_{44} \oplus K_{10} \oplus K_7 = f_{141} \\
(\neg K_{93} \oplus K_{34} \oplus K_{39}) \wedge \neg K_9 \oplus (\neg K_{84} \oplus K_{25} \oplus K_{30}) \wedge (K_{42} \oplus K_8 \oplus \neg K_5) \oplus K_{45} \oplus K_{11} \oplus K_8 = f_{142} \\
(\neg K_{94} \oplus K_{35} \oplus K_{40}) \wedge \neg K_{10} \oplus (\neg K_{85} \oplus K_{26} \oplus K_{31}) \wedge (K_{43} \oplus K_9 \oplus \neg K_6) \oplus K_{46} \oplus K_{12} \oplus K_9 = f_{143} \\
(\neg K_{95} \oplus K_{36} \oplus K_{41}) \wedge \neg K_{11} \oplus (\neg K_{86} \oplus K_{27} \oplus K_{32}) \wedge (K_{44} \oplus K_{10} \oplus \neg K_7) \oplus K_{47} \oplus K_{13} \oplus K_{10} = f_{144} \\
(\neg K_{96} \oplus K_{37} \oplus K_{42}) \wedge \neg K_{12} \oplus (\neg K_{87} \oplus K_{28} \oplus K_{33}) \wedge (K_{45} \oplus K_{11} \oplus \neg K_8) \oplus K_{48} \oplus K_{14} \oplus K_{11} = f_{145} \\
(K_0 \oplus K_{97} \oplus K_{38} \oplus K_{43}) \wedge \neg K_{13} \oplus (\neg K_{88} \oplus K_{29} \oplus K_{34}) \wedge (K_{46} \oplus K_{12} \oplus \neg K_9) \oplus K_{49} \oplus K_{15} \oplus K_{12} = f_{146} \\
(K_1 \oplus K_{98} \oplus K_{39} \oplus K_{44}) \wedge \neg K_{14} \oplus (\neg K_{89} \oplus K_{30} \oplus K_{35}) \wedge (K_{47} \oplus K_{13} \oplus \neg K_{10}) \oplus K_{50} \oplus K_{16} \oplus K_{13} = f_{147} \\
(K_2 \oplus K_{99} \oplus K_{40} \oplus K_{45}) \wedge \neg K_{15} \oplus (\neg K_{90} \oplus K_{31} \oplus K_{36}) \wedge (K_{48} \oplus K_{14} \oplus \neg K_{11}) \oplus K_{51} \oplus K_{17} \oplus K_{14} = f_{148} \\
(f_{100} \oplus K_{100} \oplus K_{41} \oplus K_{46}) \wedge \neg K_{16} \oplus (\neg K_{91} \oplus K_{32} \oplus K_{37}) \wedge (K_{49} \oplus K_{15} \oplus \neg K_{12}) \oplus K_{52} \oplus K_{18} \oplus K_{15} = f_{149} \\
(f_{101} \oplus K_{101} \oplus K_{42} \oplus K_{47}) \wedge \neg K_{17} \oplus (\neg K_{92} \oplus K_{33} \oplus K_{38}) \wedge (K_{50} \oplus K_{16} \oplus \neg K_{13}) \oplus K_{53} \oplus K_{19} \oplus K_{16} = f_{150} \\
(f_{102} \oplus K_{102} \oplus K_{43} \oplus K_{48}) \wedge \neg K_{18} \oplus (\neg K_{93} \oplus K_{34} \oplus K_{39}) \wedge (K_{51} \oplus K_{17} \oplus \neg K_{14}) \oplus K_{54} \oplus K_{20} \oplus K_{17} \oplus \neg K_0 = f_{151} \\
(f_{103} \oplus K_{103} \oplus K_{44} \oplus K_{49}) \wedge \neg K_{19} \oplus (\neg K_{94} \oplus K_{35} \oplus K_{40}) \wedge (K_{52} \oplus K_{18} \oplus \neg K_{15}) \oplus K_{55} \oplus K_{21} \oplus K_{18} \oplus \neg K_1 = f_{152} \\
(f_{104} \oplus K_{104} \oplus K_{45} \oplus K_{50}) \wedge \neg K_{20} \oplus (\neg K_{95} \oplus K_{36} \oplus K_{41}) \wedge (K_{53} \oplus K_{19} \oplus \neg K_{16}) \oplus K_{56} \oplus K_{22} \oplus K_{19} \oplus \neg K_2 = f_{153} \\
(f_{105} \oplus K_{105} \oplus K_{46} \oplus K_{51}) \wedge \neg K_{21} \oplus (\neg K_{96} \oplus K_{37} \oplus K_{42}) \wedge (K_{54} \oplus K_{20} \oplus K_{17} \oplus K_0) \oplus K_{57} \oplus K_{23} \oplus K_{20} \oplus \neg K_3 = f_{154} \\
(f_{106} \oplus K_{106} \oplus K_{47} \oplus K_{52}) \wedge \neg K_{22} \oplus (K_{97} \oplus K_{38} \oplus K_{43} \oplus K_0) \wedge (K_{55} \oplus K_{21} \oplus K_{18} \oplus K_1) \oplus K_{58} \oplus K_{24} \oplus K_{21} \oplus \neg K_4 = f_{155} \\
(f_{107} \oplus K_{107} \oplus K_{48} \oplus K_{53}) \wedge \neg K_{23} \oplus (\neg K_{98} \oplus K_{39} \oplus K_{44} \oplus K_1) \wedge (K_{56} \oplus K_{22} \oplus K_{19} \oplus K_2) \oplus K_{59} \oplus K_{25} \oplus K_{22} \oplus K_5 \oplus K_0 = f_{156} \\
(f_{108} \oplus K_{108} \oplus K_{49} \oplus K_{54} \oplus \neg K_0) \wedge \neg K_{24} \oplus (K_{99} \oplus K_{40} \oplus K_{45} \oplus K_2) \wedge (K_{57} \oplus K_{23} \oplus K_{20} \oplus K_3) \oplus K_{60} \oplus K_{26} \oplus K_{23} \oplus K_6 \oplus K_1 = f_{157} \\
(f_{109} \oplus K_{109} \oplus K_{50} \oplus K_{55} \oplus \neg K_1) \wedge \neg K_{25} \oplus (f_{100} \oplus K_{100} \oplus K_{41} \oplus K_{46}) \wedge (K_{58} \oplus K_{24} \oplus K_{21} \oplus K_4) \oplus K_{61} \oplus K_{27} \oplus K_{24} \oplus K_7 \oplus K_2 = f_{158} \\
(f_{110} \oplus K_{110} \oplus K_{51} \oplus K_{56} \oplus \neg K_2) \wedge \neg K_{26} \oplus (f_{101} \oplus K_{101} \oplus K_{42} \oplus K_{47}) \wedge (K_{59} \oplus K_{25} \oplus K_{22} \oplus K_5 \oplus \neg K_0) \oplus K_{62} \oplus K_{28} \oplus K_{25} \oplus K_8 \oplus K_3 = f_{159} \\
(f_{111} \oplus K_{111} \oplus K_{52} \oplus K_{57} \oplus \neg K_3) \wedge \neg K_{27} \oplus (f_{102} \oplus K_{102} \oplus K_{43} \oplus K_{48}) \wedge (K_{60} \oplus K_{26} \oplus K_{23} \oplus K_6 \oplus \neg K_1) \oplus K_{63} \oplus K_{29} \oplus K_{26} \oplus K_9 \oplus K_4 = f_{160} \\
(f_{112} \oplus K_{112} \oplus K_{53} \oplus K_{58} \oplus \neg K_4) \wedge \neg K_{28} \oplus (f_{103} \oplus K_{103} \oplus K_{44} \oplus K_{49}) \wedge (K_{61} \oplus K_{27} \oplus K_{24} \oplus K_7 \oplus \neg K_2) \oplus K_{64} \oplus K_{30} \oplus K_{27} \oplus K_{10} \oplus K_5 = f_{161} \\
(f_{113} \oplus K_{113} \oplus K_{54} \oplus K_{59} \oplus \neg K_5) \wedge \neg K_{29} \oplus (f_{104} \oplus K_{104} \oplus K_{45} \oplus K_{50}) \wedge (K_{62} \oplus K_{28} \oplus K_{25} \oplus K_8 \oplus \neg K_3) \oplus K_{65} \oplus K_{31} \oplus K_{28} \oplus K_{11} \oplus K_6 = f_{162} \\
(f_{114} \oplus K_{114} \oplus K_{55} \oplus K_{60} \oplus \neg K_6) \wedge \neg K_{30} \oplus (f_{105} \oplus K_{105} \oplus K_{46} \oplus K_{51}) \wedge (K_{63} \oplus K_{29} \oplus K_{26} \oplus K_9 \oplus \neg K_4) \oplus K_{66} \oplus K_{32} \oplus K_{29} \oplus K_{12} \oplus K_7 = f_{163} \\
(f_{115} \oplus K_{115} \oplus K_{56} \oplus K_{61} \oplus \neg K_7) \wedge \neg K_{31} \oplus (f_{106} \oplus K_{106} \oplus K_{47} \oplus K_{52}) \wedge (K_{64} \oplus K_{30} \oplus K_{27} \oplus K_{10} \oplus \neg K_5) \oplus K_{67} \oplus K_{33} \oplus K_{30} \oplus K_{13} \oplus K_8 = f_{164} \\
(f_{116} \oplus K_{116} \oplus K_{57} \oplus K_{62} \oplus \neg K_8) \wedge \neg K_{32} \oplus (f_{107} \oplus K_{107} \oplus K_{48} \oplus K_{53}) \wedge (K_{65} \oplus K_{31} \oplus K_{28} \oplus K_{11} \oplus \neg K_6) \\
\quad \oplus K_{68} \oplus K_{34} \oplus K_{31} \oplus K_{14} \oplus K_9 \oplus \neg K_0 = f_{165} \\
(f_{117} \oplus K_{117} \oplus K_{58} \oplus K_{63} \oplus \neg K_9) \wedge (K_{33} \oplus K_0) \oplus (f_{108} \oplus K_{108} \oplus K_{49} \oplus K_{54} \oplus \neg K_0) \wedge (K_{66} \oplus K_{32} \oplus K_{29} \oplus K_{12} \oplus \neg K_7) \\
\quad \oplus K_{69} \oplus K_{35} \oplus K_{32} \oplus K_{15} \oplus K_{10} \oplus \neg K_1 = f_{166} \\
(f_{118} \oplus K_{118} \oplus K_{59} \oplus K_{64} \oplus K_{10} \oplus K_0) \wedge (K_{34} \oplus K_1 \oplus \neg K_0) \oplus (f_{109} \oplus K_{109} \oplus K_{50} \oplus K_{55} \oplus \neg K_1) \\
\quad \wedge (K_{67} \oplus K_{33} \oplus K_{30} \oplus K_{13} \oplus \neg K_8) \oplus K_{70} \oplus K_{36} \oplus K_{33} \oplus K_{16} \oplus K_{11} \oplus \neg K_2 = f_{167} \\
(f_{119} \oplus K_{119} \oplus K_{60} \oplus K_{65} \oplus K_{11} \oplus K_1) \wedge (K_{35} \oplus K_2 \oplus \neg K_1) \oplus (f_{110} \oplus K_{110} \oplus K_{51} \oplus K_{56} \oplus \neg K_2) \\
\quad \wedge (K_{68} \oplus K_{34} \oplus K_{31} \oplus K_{14} \oplus K_9 \oplus K_0) \oplus K_{71} \oplus K_{37} \oplus K_{34} \oplus K_{17} \oplus K_{12} \oplus \neg K_3 = f_{168} \\
(f_{120} \oplus K_{120} \oplus K_{61} \oplus K_{66} \oplus K_{12} \oplus K_2) \wedge (K_{36} \oplus \oplus K_3 \oplus \neg K_2) \oplus (f_{111} \oplus K_{111} \oplus K_{52} \oplus K_{57} \oplus \neg K_3) \\
\quad \wedge (K_{69} \oplus K_{35} \oplus K_{32} \oplus K_{15} \oplus K_{10} \oplus K_1) \oplus K_{72} \oplus K_{38} \oplus K_{35} \oplus K_{18} \oplus K_{13} \oplus \neg K_4 = f_{169} \\
(f_{121} \oplus K_{121} \oplus K_{62} \oplus K_{67} \oplus K_{13} \oplus K_3) \wedge (K_{37} \oplus K_4 \oplus K_3 \oplus K_0) \oplus (f_{112} \oplus K_{112} \oplus K_{53} \oplus K_{58} \oplus \neg K_4) \\
\quad \wedge (K_{70} \oplus K_{36} \oplus K_{33} \oplus K_{16} \oplus K_{11} \oplus K_2) \oplus K_{73} \oplus K_{39} \oplus K_{36} \oplus K_{19} \oplus K_{14} \oplus \neg K_5 = f_{170} \\
(f_{122} \oplus K_{122} \oplus K_{63} \oplus K_{68} \oplus K_{14} \oplus K_4) \wedge (K_{38} \oplus K_5 \oplus K_4 \oplus K_1) \oplus (f_{113} \oplus K_{113} \oplus K_{54} \oplus K_{59} \oplus \neg K_5) \\
\quad \wedge (K_{71} \oplus K_{37} \oplus K_{34} \oplus K_{17} \oplus K_{12} \oplus K_3) \oplus K_{74} \oplus K_{40} \oplus K_{37} \oplus K_{20} \oplus K_{15} \oplus K_6 \oplus K_0 = f_{171} \\
(f_{123} \oplus K_{123} \oplus K_{64} \oplus K_{69} \oplus K_{15} \oplus K_5) \wedge (K_{39} \oplus K_6 \oplus K_5 \oplus K_2 \oplus \neg K_0) \oplus (f_{114} \oplus K_{114} \oplus K_{55} \oplus K_{60} \oplus \neg K_6) \\
\quad \wedge (K_{72} \oplus K_{38} \oplus K_{35} \oplus K_{18} \oplus K_{13} \oplus K_4) \oplus K_{75} \oplus K_{41} \oplus K_{38} \oplus K_{21} \oplus K_{16} \oplus K_7 \oplus K_1 = f_{172} \\
(f_{124} \oplus K_{124} \oplus K_{65} \oplus K_{70} \oplus K_{16} \oplus K_6) \wedge (K_{40} \oplus K_7 \oplus K_6 \oplus K_3 \oplus \neg K_1) \oplus (f_{115} \oplus K_{115} \oplus K_{56} \oplus K_{61} \oplus \neg K_7) \\
\quad \wedge (K_{73} \oplus K_{39} \oplus K_{36} \oplus K_{19} \oplus K_{14} \oplus K_5) \oplus K_{76} \oplus K_{42} \oplus K_{39} \oplus K_{22} \oplus K_{17} \oplus K_8 \oplus K_2 = f_{173} \\
(f_{125} \oplus K_{125} \oplus K_{66} \oplus K_{71} \oplus K_{17} \oplus K_7) \wedge (K_{41} \oplus K_8 \oplus K_7 \oplus K_4 \oplus \neg K_2) \oplus (f_{116} \oplus K_{116} \oplus K_{57} \oplus K_{62} \oplus \neg K_8) \\
\quad \wedge (K_{74} \oplus K_{40} \oplus K_{37} \oplus K_{20} \oplus K_{15} \oplus K_6 \oplus \neg K_0) \oplus K_{77} \oplus K_{43} \oplus K_{40} \oplus K_{23} \oplus K_{18} \oplus K_9 \oplus K_3 = f_{174} \\
(f_{126} \oplus K_{126} \oplus K_{67} \oplus K_{72} \oplus K_{18} \oplus K_8) \wedge (K_{42} \oplus K_9 \oplus K_8 \oplus K_5 \oplus \neg K_3) \oplus (f_{117} \oplus K_{117} \oplus K_{58} \oplus K_{63} \oplus \neg K_9) \\
\quad \wedge (K_{75} \oplus K_{41} \oplus K_{38} \oplus K_{21} \oplus K_{16} \oplus K_7 \oplus \neg K_1) \oplus K_{78} \oplus K_{44} \oplus K_{41} \oplus K_{24} \oplus K_{19} \oplus K_{10} \oplus K_4 = f_{175} \\
(f_{127} \oplus K_{127} \oplus K_{68} \oplus K_{73} \oplus K_{19} \oplus K_9) \wedge (\neg K_{43} \oplus K_{10} \oplus K_9 \oplus K_6 \oplus K_4) \oplus (f_{118} \oplus K_{118} \oplus K_{59} \oplus K_{64} \oplus K_{10} \oplus K_0) \\
\quad \wedge (K_{76} \oplus K_{42} \oplus K_{39} \oplus K_{22} \oplus K_{17} \oplus K_8 \oplus \neg K_2) \oplus \neg K_{79} \oplus K_{45} \oplus K_{42} \oplus K_{25} \oplus K_{20} \oplus K_{11} \oplus K_5 = f_{176}
\end{cases} \tag{6.3}$$

On remarque que tous les bits de K interviennent dans la définition de $\mathscr{F}_{[128,176]}$. Cependant, le nombre de solutions est probablement inexploitable puisque ce système est composé de 49 équations à 128 inconnues. Afin de le vérifier, nous avons calculé les valeurs de chaque f_i avec $128 \leq i \leq 176$ pour une clef arbtiraire dans le but de résoudre le système d'équations booléennes.

La résolution de $\mathscr{F}_{[128,176]}$ peut être réduite à une variante du problème de satisfaisabilité booléenne (SAT), qui étant donné une formule de logique propositionnelle, détermine s'il existe une assignation des variables propositionnelles telle que la formule est vraie. Le système $\mathscr{F}_{[128,176]}$ peut être traduit en une formule de logique propositionnelle en utilisant sa représentation en forme normale conjonctive (CNF). Pour ce faire, nous avons utilisé le programme `bc2cnf` [JN00]. Il suffit de créer un fichier `.txt` débutant par `BC1.1`, et rajouter pour chaque f_i une ligne `ASSIGN!(f_i);` si sa valeur numérique est 0 et `ASSIGN(f_i);` dans le cas contraire, où `f_i` désigne la valeur textuelle de f_i. À titre d'exemple, les premières lignes du fichier correspondant à $\mathscr{F}_{[128,176]}$ pour une clef arbitraire sont données ci-dessous.

```
BC1.1
ASSIGN !(!((((((!k70) ^ (!k11)) ^(!k16)) &(!k28))) ^(!k31)));
ASSIGN !((((((!k71) ^(!k12)) ^(!k17)) &(!k29))) ^(!k32));
ASSIGN !(!((((((!k72) ^(!k13)) ^!k18)) &(!k30))) ^(!k33)));
                        ⋮
```

Une fois le fichier `.txt` entièrement rempli, l'outil `bc2cnf` permet d'obtenir un fichier `.cnf` qui utilise le format textuel standard DIMACS CNF généralement requis par les solveurs SAT. Nous avons opté pour le solveur SAT `CryptoMiniSat` [1] qui est particulièrement efficace pour traiter les formules de logique propositionnelle contenant de nombreux opérateurs `XOR` [SNC09]. Dans notre cas, nous savons qu'il existe au moins une assignation telle que la formule logique propositionnelle qui découle de $\mathscr{F}_{[128,176]}$ est vraie, qui équivaut à K. Cependant, il est possible que d'autres assignations existent. Par conséquent, nous attendons du solveur SAT qu'il nous retourne toutes ces assignations afin d'exécuter une attaque par force brute pour déterminer celle qui correspond à la clef. C'est une variante du problème SAT appellée All-SAT. `CryptoMiniSat` peut être exécuté avec l'option `-maxsol n` qui tentera de retourner jusqu'à n assignations. En fixant $n = 4 \cdot 10^9$, deux semaines de calcul sur un PC muni d'un processeur i5-6200U n'ont pas suffi pour énumérer toutes les assignations. De plus, rien n'indique que la borne n choisie était assez élevée. Ces résultats nous ont amené à investiguer comment exploiter les f_i pour $i \geq 177$ afin d'augmenter le nombre d'équations booléennes.

6.2.2.2 Exploitation des fuites liées aux bits de rétroaction non constants

Comme mentionné ci-dessus, une attaque CPA à l'aide de $\varphi(k,x) = k \oplus m$ serait inefficace à l'encontre des f_i pour $i \geq 177$ étant donné que leur valeur varie au fil des exécutions. Néanmoins, grâce à la distributivité de l'opérateur `AND` par rapport à l'opérateur `XOR`, il est possible d'isoler la partie constante de la partie variable, comme illustré ci-dessous pour f_{177}.

1. https://github.com/msoos/cryptominisat/releases

$$\begin{aligned} f_{177} &= \neg(f_{128} \oplus \boldsymbol{IV_0} \oplus K_{69} \oplus K_{74} \oplus K_{20} \oplus K_{10}) \wedge S_{177,160} \oplus (f_{119} \oplus K_{119} \oplus K_{60} \oplus K_{65} \oplus K_{11} \oplus K_1) \\ &\quad \wedge \neg(K_{77} \oplus K_{43} \oplus K_{40} \oplus K_{23} \oplus K_{18} \oplus K_9 \oplus K_3 \oplus K_{38} \oplus K_1 \oplus K_4 \oplus K_5 \oplus S_{177,160}) \\ &\quad \oplus \neg(K_{38} \oplus K_1 \oplus K_4 \oplus K_5 \oplus K_{80} \oplus K_{21} \oplus K_{26} \oplus K_{43} \oplus K_6 \oplus K_{46} \oplus K_{12}) \\ &= \neg(f_{128} \oplus K_{69} \oplus K_{74} \oplus K_{20} \oplus K_{10}) \wedge S_{177,160} \oplus (f_{119} \oplus K_{119} \oplus K_{60} \oplus K_{65} \oplus K_{11} \oplus K_1) \\ &\quad \wedge \neg(K_{77} \oplus K_{43} \oplus K_{40} \oplus K_{23} \oplus K_{18} \oplus K_9 \oplus K_3 \oplus K_{38} \oplus K_1 \oplus K_4 \oplus K_5 \oplus S_{177,160}) \\ &\quad \oplus \neg(K_{38} \oplus K_1 \oplus K_4 \oplus K_5 \oplus K_{80} \oplus K_{21} \oplus K_{26} \oplus K_{43} \oplus K_6 \oplus K_{46} \oplus K_{12}) \oplus \boldsymbol{IV_0} \wedge S_{177,160} \\ &= f'_{177} \oplus \boldsymbol{IV_0} \wedge S_{177,160} \end{aligned} \tag{6.4}$$

Ainsi, il est possible de mener une attaque en faisant des hypothèses à la fois sur f'_{177} et sur $S_{177,160}$ en utilisant la fonction de sélection $\varphi(f'_{177} \parallel S_{177,160}, IV) = f'_{177} \oplus (S_{177,160} \wedge IV_0) \oplus IV_{49}$. Cependant, le nombre de bits d'IV qui s'immiscent dans le bit de rétroaction augmente au fil de l'initilisation. S'il est toujours possible de tirer avantage de la distributivité du `AND` sur le `XOR`, cela augmente exponentiellement la complexité de l'attaque. Le tableau 6.1 résume le nombre de bits d'IV qui perturbent l'attaque initiale ciblant $f_i \oplus IV_{i-128}$ pour $i \in I$.

TABLEAU 6.1 – Nombre de bits d'IV x qui perturbent une attaque contre $f_i \oplus IV_{i-128}$ pour $i \in I$

I	$[128,176]$	$[177,185]$	$[186,190]$	$[191,224]$	$\cdots$	$[254,255]$
x	0	1	2	3	$\cdots$	33

On remarque que la complexité d'une attaque à l'encontre des derniers bits de rétroaction est trop importante pour être applicable en pratique (*e.g.*, 2^{33+1} hypothèses pour f_{254} et f_{255}). Cependant, il n'est probablement pas nécessaire de cibler f_i pour chaque $i \in [128,255]$ puisque le système de 128 inconnues est rapidement constitué d'au moins 128 équations. En effet, en tirant avantage de la distributivité du `AND` sur le `XOR`, une attaque contre $f_i \oplus IV_{i-128}$ retourne $x+1$ équations où x est donné par le tableau 6.1 selon la valeur de i. À l'instar de l'équation 6.4 on a

$$f_{186} = f'_{186} \oplus \boldsymbol{IV_9} \wedge S_{186,160} \oplus \boldsymbol{IV_0} \wedge S_{186,193} \tag{6.5}$$

et

$$f_{191} = f'_{191} \oplus \boldsymbol{IV_{14}} \wedge S_{191,160} \oplus \boldsymbol{IV_5} \wedge S_{191,193} \oplus \boldsymbol{IV_0} \wedge S_{191,111}\,. \tag{6.6}$$

ce qui nous permet de poser

$$\mathscr{F}_{[128,i]} = \mathscr{F}_{[128,176]} \cup \mathscr{F}'_{[177,i]} \cup \mathscr{S}_{[177,i],160} \cup \mathscr{S}_{[186,i],193} \cup \mathscr{S}_{[191,i],111} \qquad \text{pour } i \geq 191\,. \tag{6.7}$$

Ainsi, le nombre d'équations incluses dans $\mathscr{F}_{[128,i]}$ est $i-127+(i-176)+(i-185)+(i-190)$. Puisque $i-127+(i-176)+(i-185)+(i-190) \geq 128 \Leftrightarrow i \geq 202$, alors $\mathscr{F}_{[128,i]}$ contient au moins 128 équations si et seulement si $i \geq 202$. Avec la même méthodologie que celle décrite ci-dessus pour $\mathscr{F}_{[128,176]}$, nous obtenons 329 assignations pour $\mathscr{F}_{[128,202]}$. En incrémentant i progressivement, nous constatons que $\mathscr{F}_{[128,i]}$ retourne une unique assignation (*i.e.*, la clef K) si et seulement si $i \geq 209$. Plus précisément, les fonctions de sélection à considérer sont définies par l'équation 6.8 tandis que les systèmes booléens additionnels sont brièvement décrits ci-dessous.

$$\varphi_i^{\text{ACORN}} : \begin{cases} (x) \mapsto x \oplus IV_{i-128} & \text{si } 128 \leq i \leq 176 \\ (x,y) \mapsto x \oplus (y \wedge IV_{i-177}) \oplus IV_{i-128} & \text{si } 177 \leq i \leq 185 \\ (x,y,z) \mapsto x \oplus (y \wedge IV_{i-177}) \oplus (z \wedge IV_{i-186}) \oplus IV_{i-128} & \text{si } 186 \leq i \leq 190 \\ (x,y,z,t) \mapsto x \oplus (y \wedge IV_{i-177}) \oplus (z \wedge IV_{i-186}) \oplus (t \wedge IV_{i-191}) \oplus IV_{i-128} & \text{si } 191 \leq i \leq 209 \end{cases} \tag{6.8}$$

$$\mathscr{F}'_{[177,209]} = \begin{cases} \neg(f_{128} \oplus K_{69} \oplus K_{10} \oplus K_{74} \oplus K_{20}) \wedge S_{177,160} \oplus (f_{119} \oplus K_{119} \oplus K_{60} \oplus K_1 \oplus K_{65} \oplus K_{11}) \wedge \neg(K_{77} \oplus K_{18} \oplus K_{23} \oplus K_{40} \oplus K_3 \\ \oplus K_{43} \oplus K_9 \oplus K_{38} \oplus K_1 \oplus K_4 \oplus K_5 \oplus S_{177,160}) \oplus \neg(K_{38} \oplus K_1 \oplus K_4 \oplus K_5 \oplus K_{80} \oplus K_{21} \oplus K_{26} \oplus K_{43} \oplus K_6 \oplus K_{46} \oplus K_{12}) & = f'_{177} \\ \neg(f_{129} \oplus K_{70} \oplus K_{11} \oplus K_{75} \oplus K_{21}) \wedge S_{178,160} \oplus (f_{120} \oplus K_{120} \oplus K_{61} \oplus K_2 \oplus K_{66} \oplus K_{12}) \wedge (K_{78} \oplus K_{19} \oplus K_{24} \oplus K_{41} \oplus K_4 \\ \oplus K_{44} \oplus K_{10} \oplus K_{39} \oplus K_2 \oplus K_5 \oplus K_6 \oplus K_0 \oplus S_{178,160}) \oplus K_{39} \oplus K_2 \oplus K_5 \oplus K_6 \oplus K_0 \oplus K_{81} \oplus K_{22} \oplus K_{27} \oplus K_{44} \oplus K_7 \oplus K_{47} \oplus K_{13} & = f'_{178} \\ \neg(f_{130} \oplus K_{71} \oplus K_{12} \oplus K_{76} \oplus K_{22}) \wedge S_{179,160} \oplus (f_{121} \oplus K_{121} \oplus K_{62} \oplus K_3 \oplus K_{67} \oplus K_{13}) \wedge (K_{79} \oplus K_{20} \oplus K_{25} \oplus K_{42} \oplus K_5 \\ \oplus K_{45} \oplus K_{11} \oplus K_{40} \oplus K_3 \oplus K_6 \oplus K_7 \oplus K_1 \oplus S_{179,160}) \oplus K_{40} \oplus K_3 \oplus K_6 \oplus K_7 \oplus K_1 \oplus K_{82} \oplus K_{23} \oplus K_{28} \oplus K_{45} \oplus K_8 \oplus K_{48} \oplus K_{14} & = f'_{179} \\ \neg(f_{131} \oplus K_{72} \oplus K_{13} \oplus K_{77} \oplus K_{23}) \wedge S_{180,160} \oplus (f_{122} \oplus K_{122} \oplus K_{63} \oplus K_4 \oplus K_{68} \oplus K_{14}) \wedge (K_{80} \oplus K_{21} \oplus K_{26} \oplus K_{43} \oplus K_6 \\ \oplus K_{46} \oplus K_{12} \oplus K_{41} \oplus K_4 \oplus K_7 \oplus K_8 \oplus K_2 \oplus S_{180,160}) \oplus K_{41} \oplus K_4 \oplus K_7 \oplus K_8 \oplus K_2 \oplus K_{83} \oplus K_{24} \oplus K_{29} \oplus K_{46} \oplus K_9 \oplus K_{49} \oplus K_{15} & = f'_{180} \\ \neg(f_{132} \oplus K_{73} \oplus K_{14} \oplus K_{78} \oplus K_{24}) \wedge S_{181,160} \oplus (f_{123} \oplus K_{123} \oplus K_{64} \oplus K_5 \oplus K_{69} \oplus K_{15}) \wedge (K_{81} \oplus K_{22} \oplus K_{27} \oplus K_{44} \oplus K_7 \\ \oplus K_{47} \oplus K_{13} \oplus K_{42} \oplus K_5 \oplus K_8 \oplus K_9 \oplus K_3 \oplus S_{181,160}) \oplus K_{42} \oplus K_5 \oplus K_8 \oplus K_9 \oplus K_3 \oplus K_{84} \oplus K_{25} \oplus K_{30} \oplus K_{47} \oplus K_{10} \oplus K_{50} \oplus K_{16} & = f'_{181} \\ \vdots & \vdots \\ (f_{160} \oplus f_{101} \oplus K_{101} \oplus K_{42} \oplus K_{47} \oplus f_{106} \oplus K_{106} \oplus K_{47} \oplus K_{52}) \wedge S_{209,160} \oplus (\neg(f_{151} \oplus K_{92} \oplus K_{33} \oplus K_{38} \oplus f_{97} \oplus K_{97} \oplus K_{38} \oplus K_{43}) \\ \wedge S_{209,193} \oplus (f_{146} \oplus K_{87} \oplus K_{28} \oplus K_{92} \oplus f_{109} \oplus K_{109} \oplus K_{50} \oplus K_{55} \oplus K_{72} \oplus K_{13} \oplus K_{18} \oplus K_{35} \oplus f_{112} \oplus K_{112} \oplus K_{53} \oplus K_{58} \\ \oplus K_1 \oplus K_{78} \oplus K_{19} \oplus K_{24} \oplus K_{41} \oplus K_4 \oplus K_{44} \oplus K_{10}) \wedge \neg K_{27} \oplus \neg(S_{203,160} \oplus K_{27} \oplus f_{112} \oplus K_{112} \oplus K_{53} \oplus K_{58} \oplus K_{75} \oplus K_{16} \oplus K_{21} \\ \oplus K_{38} \oplus K_1 \oplus K_{78} \oplus K_{19} \oplus K_{24} \oplus K_{44} \oplus K_7 \oplus K_{10}) & = f'_{209} \end{cases} \tag{6.9}$$

$$\mathscr{S}_{[177,209],160} = \begin{cases} \neg(K_{44} \oplus K_7 \oplus K_{10} \oplus K_5 \oplus K_{11}) & = S_{177,160} \\ \neg(K_{45} \oplus K_8 \oplus K_{11} \oplus K_6 \oplus K_{12}) & = S_{178,160} \\ \neg(K_{46} \oplus K_9 \oplus K_{12} \oplus K_7 \oplus K_{13}) & = S_{179,160} \\ \neg(K_{47} \oplus K_{10} \oplus K_{13} \oplus K_8 \oplus K_{14}) & = S_{180,160} \\ \neg(K_{48} \oplus K_{11} \oplus K_{14} \oplus K_9 \oplus K_{15}) & = S_{181,160} \\ \vdots & \vdots \\ K_{76} \oplus K_{39} \oplus K_{42} \oplus K_{37} \oplus K_{43} \oplus K_{22} \oplus K_{17} \oplus K_{10} \oplus K_9 \oplus K_8 \oplus K_6 \oplus K_5 \oplus K_4 \oplus K_3 \oplus K_2 \oplus K_0 & = S_{209,160} \end{cases} \tag{6.10}$$

$$\mathscr{S}_{[186,209],193} = \begin{cases} \neg(S_{180,160} \oplus S_{182,160} \oplus S_{185,160} \oplus S_{186,160} \oplus K_{86} \oplus K_{27} \oplus K_{32} \oplus K_{10} \oplus K_{16} \oplus K_{13} \oplus K_{19}) & = S_{186,193} \\ \neg(S_{181,160} \oplus S_{183,160} \oplus S_{186,160} \oplus S_{187,160} \oplus K_{87} \oplus K_{28} \oplus K_{33} \oplus K_{11} \oplus K_{17} \oplus K_{14} \oplus K_{20}) & = S_{187,193} \\ \neg(S_{182,160} \oplus S_{184,160} \oplus S_{187,160} \oplus S_{188,160} \oplus K_{88} \oplus K_{29} \oplus K_{34} \oplus K_{12} \oplus K_{18} \oplus K_{15} \oplus K_{21}) & = S_{188,193} \\ \neg(S_{183,160} \oplus S_{185,160} \oplus S_{188,160} \oplus S_{189,160} \oplus K_{89} \oplus K_{30} \oplus K_{35} \oplus K_{13} \oplus K_{19} \oplus K_{16} \oplus K_{22}) & = S_{189,193} \\ \neg(S_{184,160} \oplus S_{186,160} \oplus S_{189,160} \oplus S_{190,160} \oplus K_{90} \oplus K_{31} \oplus K_{36} \oplus K_{14} \oplus K_{20} \oplus K_{17} \oplus K_{23}) & = S_{190,193} \\ \vdots & \vdots \\ S_{204,193} \oplus S_{206,160} \oplus K_{209,160} \oplus S_{210,160} \oplus K_{110} \oplus K_{51} \oplus K_{56} \oplus K_{34} \oplus K_{40} \oplus K_{37} \oplus K_{43} \oplus f_{110} & = S_{209,193} \end{cases} \tag{6.11}$$

$$\mathscr{S}_{[191,209],111} = \begin{cases} \neg K_9 & = S_{191,111} \\ \neg K_{10} & = S_{192,111} \\ \neg K_{11} & = S_{193,111} \\ \neg K_{12} & = S_{194,111} \\ \neg K_{13} & = S_{195,111} \\ \vdots & \vdots \\ \neg K_{27} & = S_{209,111} \end{cases} \tag{6.12}$$

On remarque que les systèmes $\mathscr{S}$ définis par les variables additionnelles sont moins complexes que $\mathscr{F}$ et $\mathscr{F}'$. Notamment, chaque équation de $\mathscr{S}_{[191,209],111}$ définit simplement la valeur d'un bit de clef. Ces systèmes sont probablement d'une grande aide pour réduire drastiquement le nombre d'assignations.

Cette étude démontre théoriquement que la phase d'initialisation d'ACORN peut être la cible d'attaques par observations. Notons tout de même qu'une telle attaque nécessite inévitablement l'utilisation de techniques algébriques afin de retrouver la clef à partir des équations booléennes. Dans la section suivante, nous tentons d'appliquer l'attaque à une implémentation logicielle microcontrôleur sur microcontrôleur.

6.2.3 Application pratique

6.2.3.1 Implémentation optimisée

Bien que la spécification d'ACORN présente un mécanisme bit à bit, il est en réalité possible de paralléliser les opérations pour opérer sur des mots plutôt que des bits. Dans le cadre de la compétition CAESAR, Hongjun Wu a proposé une implémentation optimisée d'ACORN qui traite jusqu'à 32 bits en parallèle. Cela est possible puisque pour chaque registre, le premier bit de rétroaction linéaire et le bit poids faible sont espacés d'au moins 32 bits. Par conséquent, les fonctions de rétroaction linéaires ainsi que les fonctions Φ et κ définies en section 3.2.3.2 peuvent opérer sur des mots de 32 bits au lieu de

simple bits sans perturber le bon fonctionnement de l'algorithme. La version originale de l'implémentation optimisée, rappelée en annexe A.3, qui a été publiée dans le but de souligner l'efficacité d'ACORN en logiciel sur des processeurs avancés, fait en réalité usage de processeurs 64 bits. En effet, l'intérêt de stocker chaque LFSR dans une variable de 64 bit est de construire toutes les variables intermédiaire de 32 bits en un simple décalage bit à bit. Considérons à titre d'exemple le premier LFSR défini par les $S_{0\cdots60}$. Si ce dernier est stocké dans une variable de 64 bits x, alors les calculs des mots de rétroaction linéaire $S_{0\cdots31}$ et $S_{23\cdots54}$ correspond respectivement à $x \gg 29$ et $(x \gg 6) \wedge (2^{32}-1)$. Il est à noter que si la variable de destination est définie sur 32 bits (*e.g.*, `unsigned int` en langage C), il est possible de s'affranchir du masque de 32 bits. Ainsi, cette méthode constitue un compromis temps-mémoire qui permet d'économiser quelques cycles d'exécution au prix d'une consommation de RAM plus importante (*i.e.*, 448 bits pour l'état interne au lieu de 293 bits théoriques).

Nous avons décidé de suivre cette approche pour implémenter ACORN sur notre microcontrôleur 32-bit étant donné qu'elle permet d'atteindre les meilleures performances. Par conséquent, au lieu de concaténer tous les LFSR les uns à la suite des autres afin de contenir l'intégralité de l'état interne dans 10 variables de 32 bits, chaque LFSR est divisé en deux variables. De cette façon, on économise des opérations pour construire les variables temporaires, bien que cela induise un surcoût par rapport à la version 64-bit. En effet, si on reconsidère le LFSR défini par $S_{0\cdots60}$, il est alors contenu dans deux variables $x_0 = S_{0\cdots31}$ et $x_1 = S_{32\cdots60}$. Si la construction du mot de rétroaction linéaire $S_{0\cdots31}$ est immédiate puisqu'il est défini par x_0, le construction du mot $S_{23\cdots54}$ nécessite de calculer $(x_0 \ll 23) \parallel (x_1 \gg 6)$.

Le code Assembleur ARM qui en découle est donné en annexe A.4, et constitue la première implémentation optimisée d'ACORN publiée sur ce type d'architecture. Il est également à noter que les accès mémoires sont inévitables puisque l'état interne requiert à lui tout seul les 13 registres généraux à disposition, et que plusieurs variables temporaires sont requises au fil de l'exécution de la fonction de mise à jour. Notamment, la valeur intermédiaire ciblée par notre attaque fait nécessairement l'objet d'une instruction mémoire puisqu'elle est injectée dans l'état interne à la fin de la fonction. Par conséquent, les difficultés rencontrées au chapitre précédent n'ont pas lieu d'être sur les implémentations embarquées 32 bits.

6.2.3.2 Première tentative d'attaque

Puisque notre implémentation exécute 32 étapes à la fois, la phase d'initialisation constituée de 1792 étapes ne nécessite en réalité que $\frac{1792}{32} = 56$ appels à la fonction Assembleur. Le signal de déclenchement est activé dès le cinquième appel de fonction (*i.e.*, celui qui prend en entrée $IV_{0\cdots31}$) et est désactivé à la fin de la phase d'initialisation. Pour 5000 exécutions d'ACORN avec des IV aléatoires et la clef codée en dur K = '`Encryption Key K`' = `0x456e6372797074696f6e204b6579204b`, nous enregistrons les fuites liées aux quatre appels de fonction qui nous intéressent (*i.e.*, ceux avec les mots d'IV en entrée). Comme illustré par la figure 6.2, les mises à jour de l'état interne sont clairement distinguables et sont chacune constituées de 10000 échantillons.

Afin d'exécuter l'attaque définie théoriquement ci-dessus, nous avons choisi d'adopter une approche mono-bit étant donné que les fonctions de sélection évoluent selon les indices des bits de rétroaction ciblés. Plus précisément, une attaque CPA à l'encontre de $f_i \oplus IV_{i-128}$ est exécutée par

$$\mathsf{CPA}\left(\varphi_i^{\text{ACORN}}, \mathsf{HW}, T^{1\cdots5000}, \left[10000 \times \left\lceil \frac{i}{32} \right\rceil + 1, 10000 \times \left\lceil \frac{i+32}{32} \right\rceil\right], IV^{1\cdots5000}\right).$$

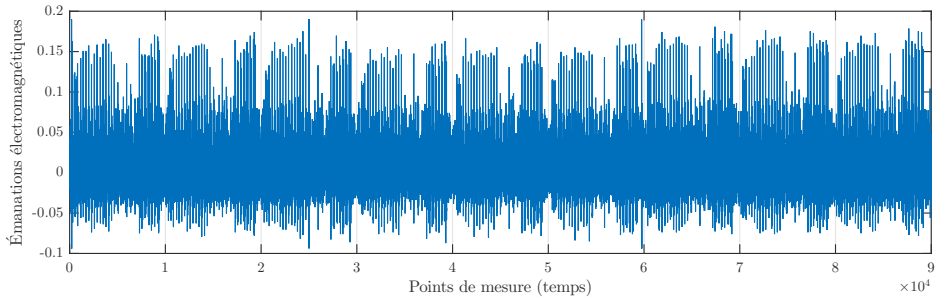


FIGURE 6.2 – Neuf mises à jour de l'état interne d'ACORN (version 32-bit)

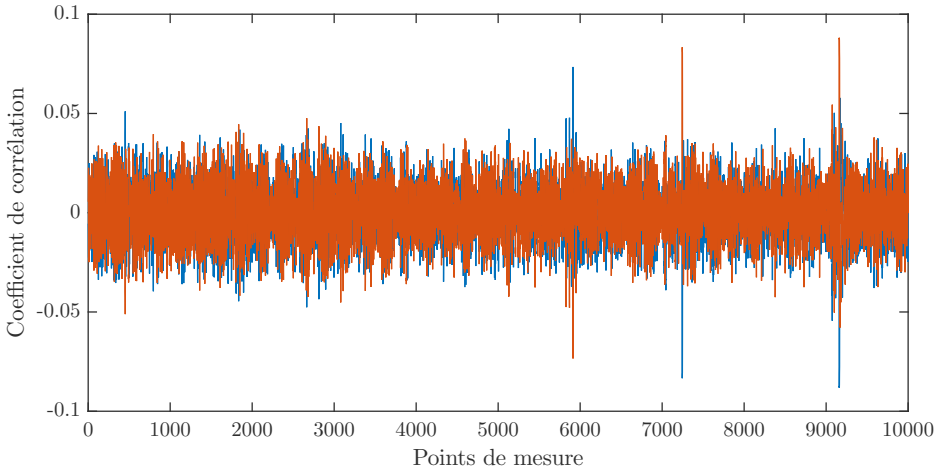
Nous construisons ainsi le système $\mathscr{F}_{[128,209]}$, mais il résulte de la phase d'analyse algébrique qu'aucune assignation ne satisfait la formule de logique propositionnelle. Manifestement, des difficultés pratiques remettent en question l'attaque introduite précédemment. Afin d'identifier ces dernières, nous avons en premier lieu inspecté visuellement les résultats pour plusieurs bits de rétroaction. La figure 6.3 illustre les fuites dans le temps pour quelques uns d'entre eux. La première observation qui peut être faite est que la fuite d'information n'est pas identique pour chaque bit de rétroaction. Par exemple, on distingue trois échantillons (5 913, 7 245 et 9 160) qui révèlent de l'information sur f_{128} en figure 6.3a, tandis que les figures 6.3b et 6.3c ne mettent en avant que deux fuites nettes aux échantillons 5 913 et 9 329. Les raisons qui expliqueraient ces résultats sont les suivantes.

Puisque l'implémentation traite 32 bits à la fois, tous les f_i injectés à l'état interne – excepté les 4 bits de poids faible – ont été affectés par la fonction de rétroaction linéaire du 6ème LFSR (*i.e.*, $f_i \oplus S_{i+4,230} \oplus S_{i+4,235}$). Plus précisément, après la mise à jour de S_{128} avec le mot de 32 bits $IV_{0\cdots31}$, les 32 derniers bits de S_{160} sont comme suit.

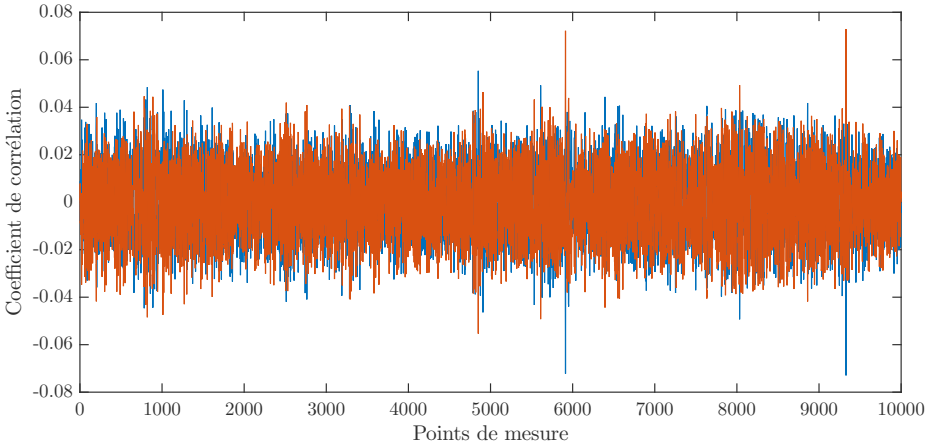
$$\begin{aligned}\left(S_{160,261},\cdots,S_{160,288}\right) &= \left(f_{128} \oplus S_{132,230} \oplus S_{132,235} \oplus IV_0,\cdots,f_{155} \oplus S_{159,230} \oplus S_{159,235} \oplus IV_{27}\right)\\ \left(S_{160,289},\cdots,S_{160,292}\right) &= \left(f_{156} \oplus IV_{28},\cdots,f_{159} \oplus IV_{31}\right)\end{aligned} \tag{6.13}$$

Par conséquent, la fonction de sélection $\varphi^{\text{ACORN}}_{i\in[0,48]}$ cible $f^1_{128} = f_{128} \oplus S_{132,230} \oplus S_{132,235}$ en plus de f_{128} lorsque $i < 28 \bmod 32$. De plus, l'implémentation Assembleur est générique dans le sens où le chiffré est inconditionnellement calculé, et ce même pendant la phase d'initialisation. Cela permet d'utiliser une unique fonction pour l'exécution de tout l'algorithme et donc de minimiser la taille du code. En revanche, cela signifique que ks_i est également la cible de $\varphi^{\text{ACORN}}_{i\in[0,48]}$ étant donné que le chiffré vaut $ks_i \oplus IV_{i-128}$ où le bit de pseudo-aléa ks_i définit une combinaison des bits de clef.

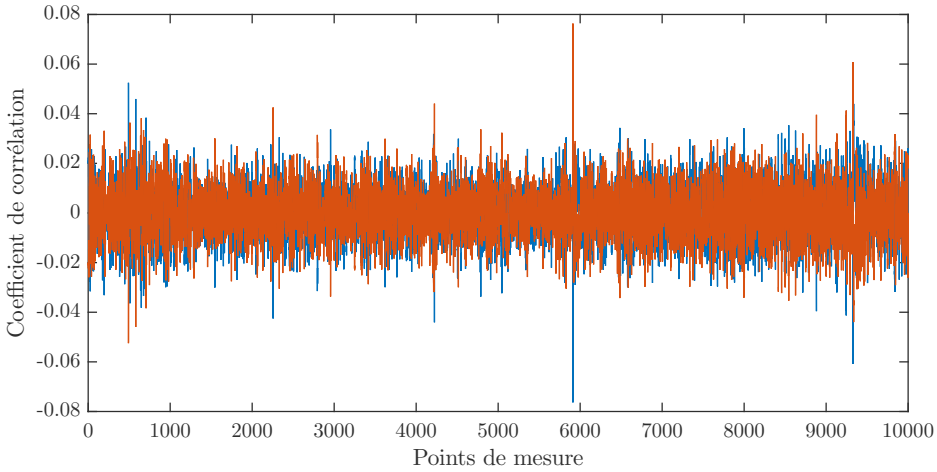
Plus précisément, notre code Assembleur calcule et stocke en RAM $ks_i \oplus IV_{i-128}$ aux lignes 121-122 de l'annexe A.4, tandis que le mot de 32 bits $f_{128\cdots159} \oplus IV_{0\cdots31} \oplus (S_{132\cdots159,230} \oplus S_{132\cdots159,235}) \lll 4$ est calculé et stocké en RAM aux lignes 151-152. Par conséquent, il y a trois instructions mémoires censées exhiber un biais statistique élevé en lien avec $\varphi^{\text{ACORN}}_{i\in[0,48]}$ aux lignes 122, 152 et 196 dans le cas où $i < 28 \bmod 32$, et uniquement deux si $i \geq 28 \bmod 32$ (lignes 122 et 199). Cela corrobore avec les résultats de la figure 6.3. Ainsi, les fuites aux échantillons 5 913, 7 245, 9 160 et 9 329 correspondent respectivement aux instructions aux lignes 122, 152, 196 et 199 du code Assembleur.



(a) $f_{128} \oplus IV_0$



(b) $f_{157} \oplus IV_{29}$



(c) $f_{159} \oplus IV_{31}$

FIGURE 6.3 – Fuites dans le temps pour la première mise à jour de l'état interne. Les courbes bleues et oranges font référence aux bits 0 et 1, respectivement.

Ces observations soulignent les difficultés qui peuvent être rencontrées lorsque l'attaque est exécutée en pratique. En effet les fonctions de sélection utilisées sont en réalité liées à plusieurs combinaisons de bits de clefs (pseudo-aléa de chiffrement, bit de rétroaction et bit de rétroaction mis à jour). De ce fait, un attaquant doit associer chaque fuite à une valeur intermédiaire. En effet, si le coefficient de corrélation maximal est obtenu pour ks_i mais que l'hypothèse est assignée dans $\mathscr{F}_{[128,209]}$ à la valeur de f_i (comme c'est le cas en figure 6.3c), alors l'attaque échouera si $f_i \neq ks_i$.

Puisqu'il n'y a pas d'accès mémoire manipulant $f_i \oplus IV_{i-128}$ pour $i < 28 \bmod 32$, il est nécessaire de modifier certaines fonctions de sélection définies par l'équation 6.8 afin d'exploiter les fuites à disposition. En effet, à partir de $i = 182$, $S_{i+4,235}$ dépend de IV_0. Les fonctions de sélection modifiées sont notées $\varphi_i'^{\text{ACORN}}$ et sont définies ci-dessous.

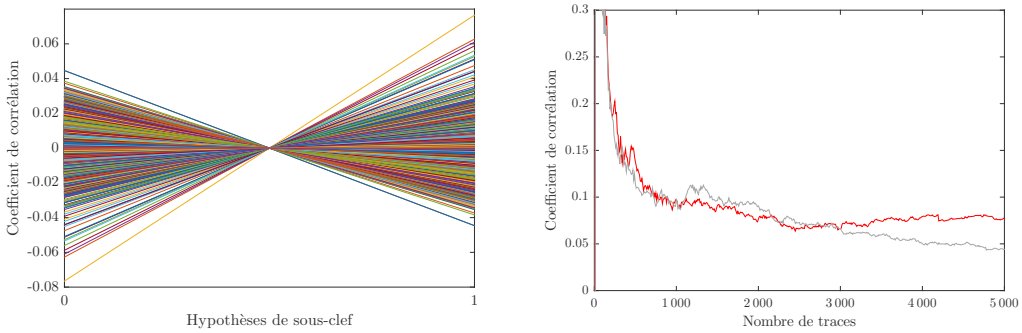
$$\varphi_i^{1\,\text{ACORN}} = \begin{cases} \varphi_i^{\text{ACORN}} & \text{si } i \geq 28 \bmod 32 \text{ ou } i \leq 182 \\ \varphi_{i-54}^{\text{ACORN}} \circ \varphi_i^{\text{ACORN}} & \text{si } 182 \leq i \leq 186 \\ \varphi_{i-59}^{\text{ACORN}} \circ \varphi_{i-54}^{\text{ACORN}} \circ \varphi_i^{\text{ACORN}} & \text{sinon} \end{cases} \tag{6.14}$$

Bien évidemment, les équations booléennes du systèmes doivent être modifiées pour être en accord avec les combinaisons de bits définies par φ_i^1 et nous notons cette variante $\mathscr{F}^1_{[128,i]}$. Une attaque sur les mêmes acquisitions a été de nouveau lancée, sans considérer les fuites liées aux bits pseudo-aléatoires de chiffrement, en exécutant pour $i = 0 \cdots 81$

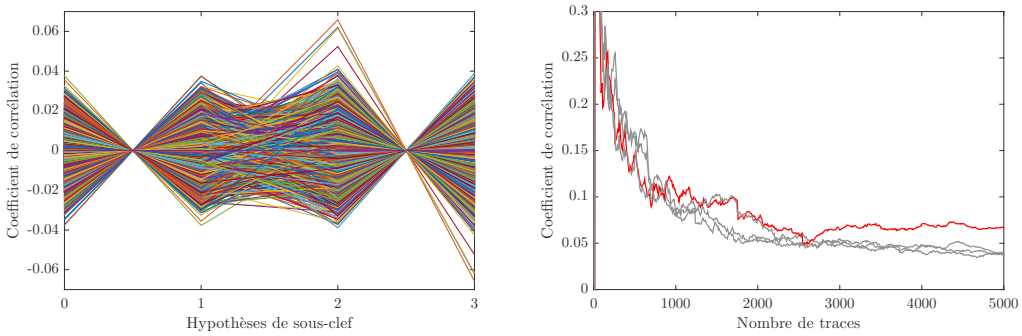
$$\mathsf{CPA}\left(\varphi_i^{1\,\text{ACORN}}, \mathsf{HW}, T^{1\cdots 5000}, \left[10000 \times \left\lceil \frac{i}{32} \right\rceil + 7000, 10000 \times \left\lceil \frac{i+32}{32} \right\rceil\right], IV^{1\cdots 5000}\right).$$

L'attaque est concluante et la formule logique propositionnelle découlant de $\mathscr{F}^1_{[128,209]}$ est uniquement satisfaite par l'assignation correspondant à la clef de chiffrement. De plus, on remarque que la version $\mathscr{F}^1_{[128,i]}$ retourne une unique assignation pour $i \geq 206$ au lieu de $i \geq 209$ pour $\mathscr{F}_{[128,i]}$. La figure 6.4 illustre les résultats obtenus pour chaque type de fonction de sélection.

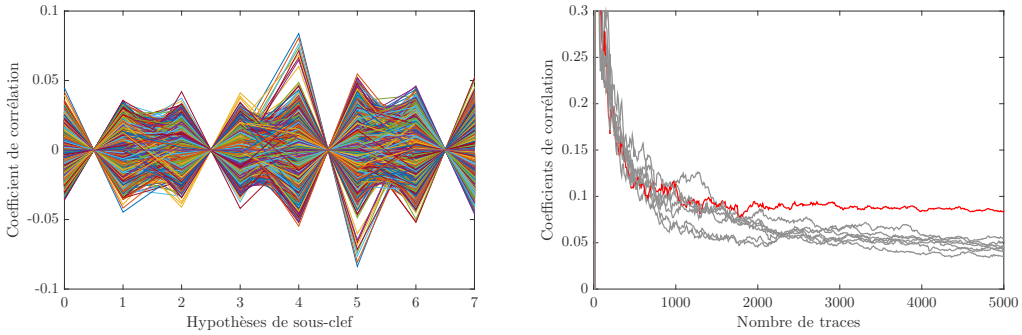
Plusieurs conclusions peuvent être tirées de la mise en pratique de l'attaque algébrique par canaux auxiliaires contre ACORN. Premièrement, le calcul du bit chiffré devrait être écarté durant la phase d'initialisation, au détriment d'une taille d'implémentation supérieure. En effet, ce calcul peut être la source de fuites supplémentaires qui faciliteraient une attaque. Deuxièmement, les fonctions de sélection théoriquement définies peuvent en réalité être utilisées pour retrouver plusieurs combinaisons de bits de clef, en plus du pseudo-aléa de chiffrement. Par conséquent, il est possible de décliner autant de variantes de cette attaque que le nombre de systèmes d'équations booléennes qui peuvent être définis.



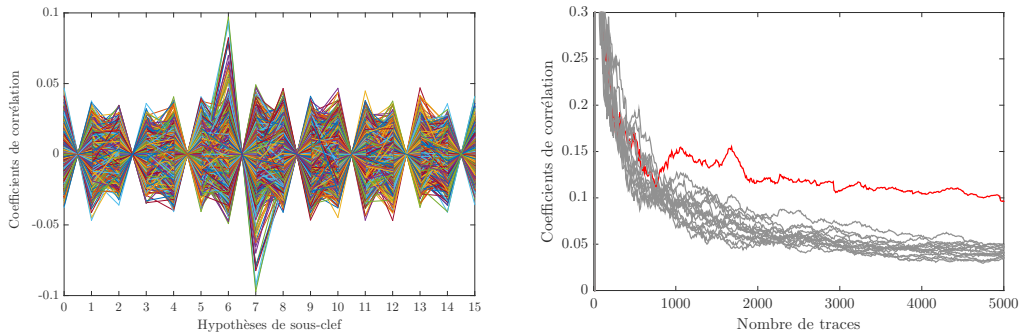
(a) $\varphi_0^{1_{\text{ACORN}}}$ retourne $f_{128}^1 = 1$



(b) $\varphi_{49}^{1_{\text{ACORN}}}$ retourne $f_{177}^1 \parallel S_{177,160} = (10)_2$



(c) $\varphi_{58}^{1_{\text{ACORN}}}$ retourne $f_{186}^1 \parallel S_{186,160} \parallel S_{186,193} = (100)_2$



(d) $\varphi_{63}^{\text{ACORN}}$ retourne $f_{191}||S_{191,160}||S_{191,193}||S_{191,111} = (0110)_2$

FIGURE 6.4 – Résultats expérimentaux pour les différentes fonctions de sélection

6.3 Intégration efficace de schémas de masquage à l'ordre 1

Maintenant que nous avons démontré théoriquement et expérimentalement le besoin de contre-mesures pour ACORN, nous proposons ici des schémas de masquage à l'ordre 1 pour ACORN et ASCON et comparons les résultats obtenus avant et après l'intégration de ces derniers.

6.3.1 Schémas de masquage partiels

6.3.1.1 ACORN

Puisque la phase d'initialisation est la seule étape sensible vis-à-vis des attaques physiques, nous proposons un schéma de masquage qui s'applique uniquement à cette étape. Les seules opérations non-linéaires au sein d'ACORN sont les opérations `AND` bit à bit. Nous utilisons la méthode définie par l'algorithme 4.4 pour calculer des portes `AND` sur des opérandes masqués booléennement, qui est la plus optimale pour un masquage d'ordre 1. De plus, nous réécrivons la fonction $Maj(x,y,z)$ comme détaillé ci-dessous afin d'économiser 2 portes `AND` à chaque mise à jour de l'état interne et par conséquent de réduire l'impact du schéma de masquage.

$$\begin{aligned} Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\ &= (x \oplus y) \wedge (y \oplus z) \oplus y \end{aligned} \tag{6.15}$$

Le schéma de masquage proposé est le suivant :

1. Un état interne (pseudo-)aléatoire S' de 293 bits est généré à chaque exécution de l'algorithme. Puisque l'état interne initial S est initialisé à zéro, on affecte S' à S de telle sorte que $S \oplus S' = S$.
2. Durant la phase d'initialisation, la fonction d'itération opère sur les deux états internes simultanément étant donné que chaque opération bit à bit manipule la valeur masquée et le masque.
3. Á la fin de la fonction d'itération, le masque correspondant au bit/mot de rétroaction est injecté seul à S' tandis le bit/mot d'entrée est ajouté (via `XOR`) au bit/mot de rétroaction masqué avant d'être injecté à S.
4. Une fois la mise à jour terminée, on réapplique S' à S et la suite de l'algorithme est opère simplement sur S sans avoir recours à aucune technique de masquage.

Il est à noter que le masque n'est pas directement appliqué à la clef ou à l'IV. Cependant, cela n'affecte pas la sécurité du schéma de masquage puisqu'elles sont injectées via un `XOR` avec le bit/mot de rétroaction masqué, et bénéficient par conséquent du masque de ce dernier.

6.3.1.2 ASCON

Comme mentionné au début de ce chapitre, la vulnérabilité d'ASCON face aux attaques par obersation a été avérée en pratique. Pour rappel, l'état interne d'ASCON est représenté par cinq variables de 64 bits notées $x_0, \cdots, x_4$. Lors de l'initialisation de l'algorithme, une constante est affectée à x_0 tandis que la clef est affectée à x_1 et x_2 tandis que le l'IV est affecté à x_3 et x_4. L'attaque présentée dans [SD17] exploite l'interaction de ces variables durant le premier tour de la permutation p^a lors de l'initialisation. Ainsi, tout comme ACORN, la phase d'initialisation est sujette aux attaques par obersavtion.

Concernant les autres étapes de l'algorithme, les auteurs d'ASCON affirment que la récupération de l'état interne durant le traitement des données ne permet pas de trouver

la clef ou du forger des messages (*"A recovery of the secret state during data processing does not directly lead to a key-recovery or universal forgery"* [DEMS16]). Cependant, comme illustré par la figure 3.6, la clef est additionnée à l'état interne avant et après la phase d'initialisation. Ainsi il est possible de mener une attaque à partir du code d'authentitifcation en ciblant $T \oplus K$ (*i.e*, les bits en sortie de p^a durant la finialisation). Ce chemin d'attaque a notamment été souligné dans [GWDE17].

Ainsi, notre schéma de masquage partiel pour ASCON se limite aux phases d'initialisation et de finalisation et consiste en les étapes suivantes :

1. Les variables $x_0, \cdots, x_4$ sont masquées avec des variables $x'_0, \cdots, x'_4$ générées (pseudo-)aléatoirement à chaque exécution. On note $M_r = x'_0$ et $M_c = x'_1 \| \cdots \| x'_4$ les masques correspondant respectivement à la taille des blocs traités et à la capacité de la construction en éponge.
2. La permutation p^a durant l'initialisation opère sur les données masquées. Une fois l'initialisation complétée, la clef est ajoutée à l'état interne avant de démasquer l'état interne.
3. Les phases de traitement des données additionnelles et des données à chiffrer/déchiffrer sont exécutées sans avoir recours à aucune technique de masquage.
4. Avant l'addition de la clef précédant la phase d'initialisation, l'état interne est de nouveau masqué. Notre schéma de masquage réutilise les masques en sortie d'initialisation afin de limiter la quantité de (pseudo-)aléa a générer. La permutation p^a durant la finilisation opère donc sur les données masquées.
5. Finalement, les bits de poids faible de l'état interne après p^a sont démasqués une fois que la clef a été additionnée à ces derniers. Il en résulte le code d'authentification.

La figure 6.5 illustre le schéma de masquage décrit ci-dessus.

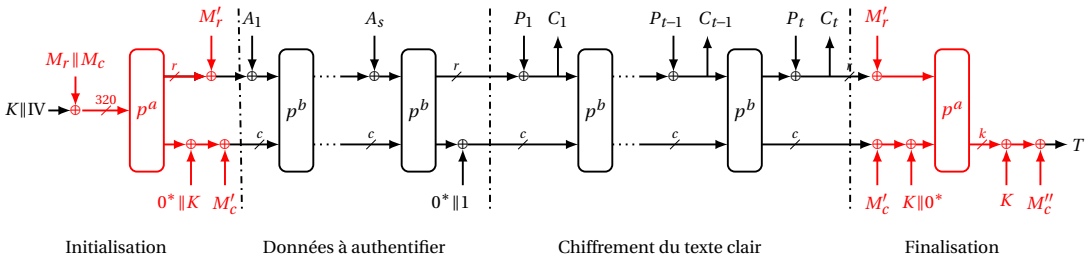


FIGURE 6.5 – Schéma de masquage partiel pour ASCON

Tout comme ACORN, la seule opération non-linéaire à considérer est l'opérateur `AND`. Par conséquent, nos deux schémas de masquage reposent sur le même algorithme pour calculer les opérations non-linéaires, ce qui apporte une certaine cohérence à notre étude comparative.

6.3.2 Comparaison des surcoûts d'implémentation

6.3.2.1 Méthodologie d'évaluation

Dans le cadre de la compétition CAESAR, il était exigé que chaque soumission soit accompagnée d'un code logiciel de référence. Pour des raisons d'homogénéité, chaque code de référence devait être écrit en langage C et être conforme à une certaine interface de programmation applicative. Notamment, le prototype de la fonction à faire appel pour chiffrer/authentifier des données est le suivant.

```
int crypto_aead_encrypt(unsigned char *c, unsigned long long *clen,
                        unsigned char *m, unsigned long long *mlen,
                        const unsigned char *ad, unsigned long long adlen,
                        const unsigned char *nsec, const unsigned char *npub,
                        const unsigned char *k);
```

Afin d'implémenter ACORN et ASCON sur notre plateforme embarquée, nous sommes partis de ces codes de référence mais avons remplacé la fonction principale de chaque algorithme (*i.e.*, respectivement la fonction d'itération et la permutation p) par une implémentation en langage Assembleur écrite par nos soins. Ce choix est principalement justifié pour des raisons de performance. De plus, toutes les fonctions auxiliaires appelées par `crypto_aead_encrypt` sont déclarées avec le mot clef `inline` afin d'optimiser les performances en évitant les surcoûts liés aux appels de fonction (*i.e.*, opérations sur la pile).

Pour notre évaluation, nous nous sommes concentrés sur des données à chiffrer de petites tailles, allant de 8 à 256 octets. Ce choix est justifié par les critères de sélection définis dans la compétition CAESAR mais aussi parce que la taille maximale des paquets au sein des réseaux de l'IdO se situe généralement dans cet intervalle (*e.g.*, jusqu'à 12 octets pour Sigfox et 243 octets pour LoRa). Concernant la taille des données additionnelles, nous avons choisi de la fixer arbitrairement à 16 octets puisque la taille de ces données est généralement faible dans le cadre de protocoles cryptographiques (*e.g.*, 5 octets pour TLS, 7 octets pour Bluetooth mesh et jusqu'à 25 pour LoRa).

6.3.2.2 Résultats

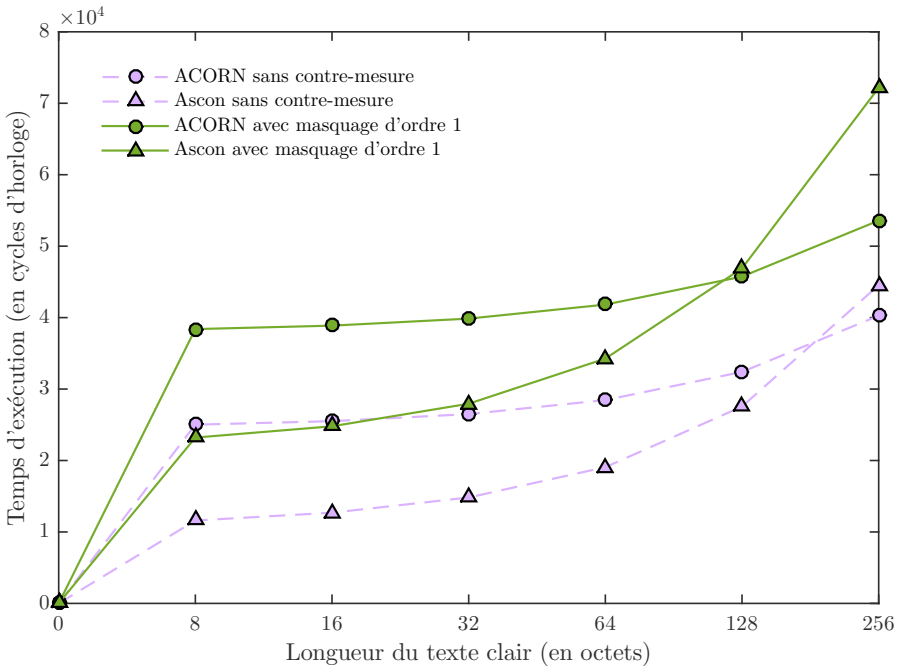


FIGURE 6.6 – Temps d'exécution de ACORN et ASCON, avec et sans masquage

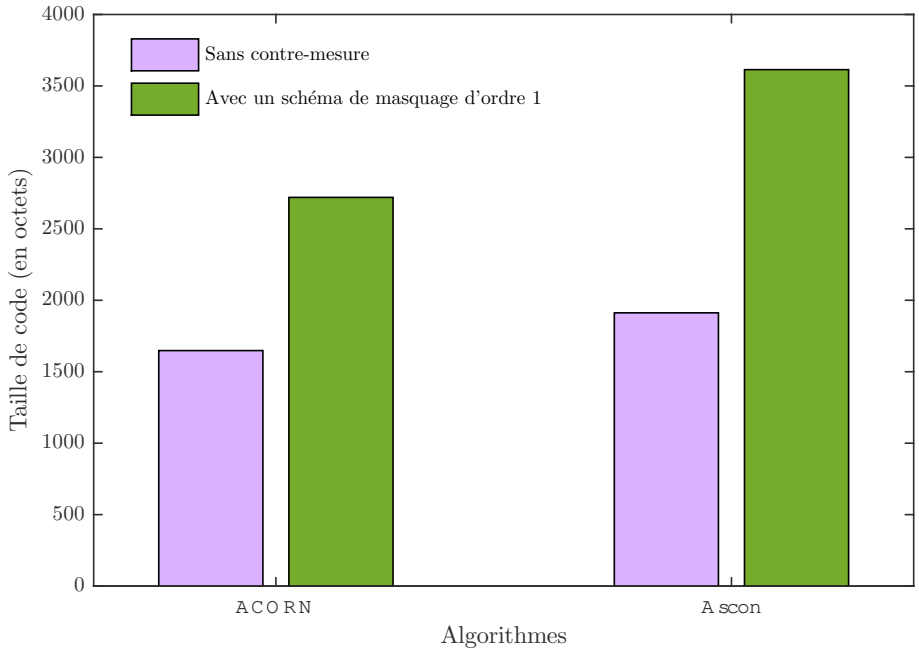


FIGURE 6.7 – Taille de code de ACORN et ASCON, avec et sans masquage

Les figures 6.6 et 6.7 présentent respectivement les résultats obtenus pour les temps d'exécution et les tailles de code. Concernant les performances d'exécution, on remarque qu'avec ou sans masquage, ASCON est plus efficace que ACORN pour des messages inférieurs à 128 octets. Les surcoûts liés à nos schémas de masquage sont fixes étant donné qu'il se basent uniquement sur des étapes qui ne dépendent pas de la taille des données à traiter. Plus précisément, on observe un surcoût d'environ 13 000 cycles d'exécution pour ACORN contre environ 10 000 pour ASCON. Ainsi, les schémas de masquage proposés ne changent pas la donne de manière significative. Il est intéressant de noter que pour des messages de tailles inférieures ou égales à 32, ASCON avec un schéma de masquage à l'ordre 1 a une efficacité similaire à ACORN sans protection. En ce qui concerne les tailles d'implémentation, nos résultats sont en phase avec ceux publiés par les précédents travaux sur des implémentations matérielles, à savoir que ACORN est plus compact, avec ou sans contre-mesures.

Bien que nos mesures ne prennent pas en considération la génération de nombres (pseudo-)aléatoires, les exigences en la matière sont quasi similaires pour les deux algorithmes (*i.e.*, respectivement 293 et 320 bits, soient 37 et 40 octets).

6.4 Estimations pour un masquage d'ordre supérieur

Étant donné la compacité des schémas de masquage proposés ci-dessus, il pourrait être envisageable d'appliquer un masquage d'ordre supérieur (*e.g.*, 2 ou 3) pour des cas d'utilisation sensibles et particulièrement exposés. On peut donc se demander si les observations dérivées de nos résultats restent les mêmes dès lors que le masquage est d'ordre supérieur.

Le fait qu'il faille générer plus d'aléa en début d'algorithme pour construire tous les états internes ne joue pas un rôle déterminant puisque la différence de taille des états internes est seulement de 3 octets. Cependant, un élément qui pourrait faire la différence

est le calcul des portes AND, qui à l'ordre supérieur, nécessite de l'aléa supplémentaire. Par exemple, le schéma de Ishai-Sahai-Wagner [ISW03] nécessite précisément 3 nombres aléatoires de la taille des opérandes pour un masquage à l'ordre 2, et 6 pour un masquage à l'ordre 3. Par conséquent, il peut être intéressant de compter le nombre de portes AND contenues dans nos schémas de masquage respectifs à ACORN et ASCON afin de déterminer si la tendance pourrait s'inverser à un certain ordre.

Dans le cas d'ACORN, en utilisant l'astuce détaillée en équation 6.15, une mise à jour de l'état interne contient six portes AND. De plus, deux d'entre elles prennent comme opérandes les bits de contrôle ca et cb qui valent soit `0x00000000`, soit `0xffffffff`, et donc ne nécessitent pas d'utiliser une technique particulière. Puisque la phase d'initialisation d'ACORN consiste en 1792 étapes, une implémentation 32 bits utilise en réalité $\frac{1792}{32} = 56$ appels de fonction, soit un total de $56 \times (6-2) = 224$ portes AND à calculer de manière sécurisée pour une initialisation complète et masquée. Concernant ASCON, p ne contient que cinq portes AND définies sur 64 bits, soit dix dans le cas d'une implémentation 32 bits. Étant donné que p est itéré 12 fois pendant l'initialisation et la finalisation, le nombre total de portes AND à sécuriser par nos schémas de masquage est égal à $10 \times 12 \times 2 = 240$.

Par conséquent, nos deux schémas de masquage requièrent approximativement le même nombre de portes AND à sécuriser. Bien que la complexité aléatoire pour ACORN n'est que légèrement inférieure, c'est un critère qui peut être d'intérêt pour les cas d'utilisation sensibles à forte exposition où la génération de nombre aléatoires est problématique.

6.5 Synthèse

Dans ce chapitre, nous avons présenté la première attaque par observations à l'encontre d'ACORN. Notre étude souligne le fait qu'une attaque par observations contre cet algorithme est nécessairement algébrique, dû à la manière dont les bits de clef interagissent avec les bits d'IV. Ainsi, l'attaque nécessite une étape supplémentaire à l'aide d'un solveur SAT afin de résoudre des problèmes de satisfaisabilité booléenne. Ce résultat met l'accent sur le fait que s'il est possible de concevoir des algorithmes qui se prêtent bien à l'intégration de contre-mesures, il est également possible concevoir des algorithmes pour lesquels le chemin d'attaque le plus trivial est nettement plus complexe que ceux des structures traditionnelles. Dans un premier temps, nous avons étudié la pertinence de notre attaque en essayant de résoudre les systèmes d'équations booléennes qui pourraient être extraits des fuite en consommation de courant et/ou émanations électromagnétiques. Cette étape nous a permis d'établir le nombre d'opérations à cibler pour mener à bien une attaque en pratique. Il s'ensuivit une mise en pratique expérimentale sur un microcontrôleur embarquant une implémentation d'ACORN écrite par nos soins en nous inspirant des optimisations publiées dans le cadre de la compétition CAESAR. Notre attaque a initialement échoué du fait que certaines fonctions de sélection ciblent plusieurs valeurs intermédiaires. Par conséquent, certaines fuites ne correspondant en réalité pas à l'équation booléenne supposée, cela a faussé la résolution du problème de satisfaisabilité. Une analyse plus détaillée nous a permis d'identifier les points d'intérêts à prendre en considération et ceux à éviter. De plus il a fallu légèrement modifier la définition des fonctions de sélection à cause des spécificités de notre implémentation traitant 32 bits en parallèle. L'attaque s'est finalement montrée fructueuse puisque nous avons été en mesure d'établir un système dont la clef de chiffrement constituait l'unique assignation. Afin de définir ce système, nous avons travaillé sur 5 000 traces dont l'acquisition a duré une dizaine de minutes et nous avons adopté une approche CPA mono-bit pour la définir la valeur de chaque équation. Le temps de calcul global nécessaire à cette étape fut d'environ une vingtaine de minutes. Finalement, la résolution du problème de

satisfaisabilité booléenne fut presque immédiate, étant donné que cela a nécessité moins d'une seconde sur un ordinateur portable standard (processeur i5-6200U).

À partir de ce résultat et des précédents travaux sur ASCON, nous avons ensuite proposé un schéma de masquage à l'ordre 1 pour ACORN et ASCON. Leurs constructions respectives permettent de limiter l'intégration de la contre-mesure à certaines étapes de l'algorithme uniquement, allègeant significativement l'impact sur les performances. Nous avons ensuite comparé leur temps d'exécution pour plusieurs tailles de messages, allant de 8 à 256 octets. Il s'avère que ASCON présente de meilleurs résultats dans le cas où les messages n'excedent pas 128 octets, mais que la tendance s'inverse dans le cas contraire. Cela est dû au fait que ACORN étant un chiffrement par flot, est désavantagé pour les courts messages à cause de sa conséquente phase d'initialisation. Ainsi, pour des applications à faibles ressources qui n'échangent qu'une faible quantité de données (*e.g.*, paquets de quelques dizaines d'octets), ASCON semble mieux adapté. Il est également à noter que comparé à CHACHA20 pour lequel nous obtenions un facteur de dégradation égal à 5 en appliquant la contre-mesure de masquage au nombre de tour minimal pour se protéger d'une attaque, nous observons ici un facteur de dégradation approximativement égal à 2. Par conséquent, nos travaux soulignent que ces algorithmes sont de meilleurs candidats dès lors que la sécurité physique est impérative.

Ces travaux ont donnés lieu aux publications avec comité de lecture *"Masking the Lightweight Authenticated Ciphers ACORN and Ascon in Software"* [AFM18c] et *"Practical Algebraic Side-Channel Attacks Against ACORN"* [AFM18b].

Chapitre 7

Conclusion et perspectives

« *There is no real ending. It's just the place where you stop the story.* »

Frank Herbert

Sommaire

7.1 Synthèse des travaux et résultats

Dans cette thèse, nous nous sommes intéressés à plusieurs algorithmes cryptographiques légers, tous ayant vocation à être efficacement intégrés dans des systèmes embarqués contraints de l'IdO.

Dans un premier temps, nous avons présenté deux réseaux de l'IdO qui intègrent la sécurité au sein même de leur spécification. Les réseaux LoRa et Bluetooth mesh témoignent du fait que dans l'IdO, la sécurité repose essentiellement, voire exclusivement dans le cas de LoRa, sur de la cryptographie symétrique. Plus précisément, l'algorithme généralement utilisé est le standard AES. Si l'implémentation de ce dernier sur des plateformes embarquées, même à faibles ressources, ne représente plus un réel défi à ce jour, cela est uniquement vrai dans le cas d'une implémentation non protégée. L'algorithme AES n'ayant pas été conçu en prenant en compte l'intégration de contre-mesure pour se protéger des attaques physiques, les surcoûts induits sont loin d'être optimaux. Nous nous sommes alors intéressés à des algorithmes considérés comme légers par rapport à l'AES, et à trois en particulier qui sont dotés d'un fort potentiel d'applications industrielles au vu des intérêts exprimés par les comités de standardisation à leur égard. Après avoir introduit ces trois algorithmes qui sont ChaCha, ACORN et ASCON, nous avons pour chacun d'eux étudié leurs vulnérabilités face aux attaques par observation ainsi que leur capacité à intégrer efficacement des contre-mesures.

L'expérimentation pratique d'une attaque exploitant les émanations électromagnétiques d'un microcontrôleur exécutant la primitive cryptographique ChaCha20 nous a permi de constater que notre matériel d'expérimentation ne nous permettait pas d'exploiter les instructions de l'unité arithmétique et logique, mais uniquement celles liées aux accès mémoires. Par conséquent, nous avons été capables de tirer parti de cette propriété pour développer une implémentation en langage Assembleur de ChaCha20 afin de minimiser les accès mémoires, et par conséquent, de rendre inefficace la seule attaque par observation publiée à l'encontre de cet algorithme. Cependant, ChaCha20 opérant sur un état interne de 64 octets, il était impossible de se passer d'accès RAM pour exécuter l'intégralité de l'algorithme. Ainsi, nous avons décidé d'étudier la possibilité d'une attaque plus complexe que celle précédemment publiée, afin d'évaluer la résistance qu'offre notre implémentation contre un banc d'attaque similaire à celui utilisé lors de nos expérimentations. Après l'étude des différents chemins d'attaques, nous sommes arrivés à la conclusion qu'il est plus efficace d'exploiter des fuites en fin d'algorithme (*i.e.*, attaquer à partir du pseudo-aléa de chiffrement) plutôt qu'au début (*i.e.*, attaquer à partir de l'IV). Cela est dû au fait que la fonction de quart de tour de ChaCha est inversible et qu'un des mots en sortie ne dépend pas de tous les mots en entrées. Cette observation nous a amené à développer une attaque que nous avons nommée "l'attaque du poseur de brique", qui permet de retrouver l'intégralité de la clef de chiffrement et dont nous avons démontré l'efficacité en pratique. Puisque cette attaque a permis de mettre en échec notre implémentation minimisant les accès mémoire, démontrant que cela ne constitue pas une contre-mesure suffisante dans le cas de ChaCha, nous avons appliqué des schémas de masquage spécialement adaptés aux structures ARX. Comme cela avait déjà été souligné par des publications antérieures, le surcoût est plus que considérable et dans le cas de ChaCha, on observe un facteur de pénalité des performances supérieur à 20, et ce pour seulement un masquage d'ordre 1. Même en limitant le masquage aux deux premiers et derniers tours, ce qui est le minimum pour contrer l'attaque que nous avons développée, le facteur de pénalité est d'environ 5. De plus, rien n'indique qu'une attaque ne serait pas possible en ciblant une valeur intermédiaire en sortie du second tour, bien que cela semble complexe au vu de nos travaux visant à définir un chemin d'attaque sur un quart

de tour complet.

Puisque les avantages apportés par ChaCha sont nuancés par d'important surcoûts dus à l'intégration de contre-mesure contre les attaques par observation, nous nous sommes intéressés à d'autres algorithmes. Nos recherches nous ont amené à nous pencher sur les deux finalistes de la compétition CAESAR pour les applications à faibles ressources, qui sont ACORN et ASCON. Ces algorithmes font tous deux finalement partie du portfolio de la compétition, mais ASCON aspire également à être un futur standard américain puisqu'il a été soumis au processus de standardisation *Lightweight Cryptography* du NIST début 2019. Les travaux sur ces deux algorithmes débutèrent suite à deux constats. Premièrement, le seul comparatif de performances entre les deux algorithmes publié, avec et sans contre-mesure contre les attaques par observation, concerne uniquement les implémentations matérielles. Au vu de nos résultats dans le cadre de l'étude sur ChaCha, il nous a paru pertinent de réitérer l'exercice pour chacun des deux algorithmes afin de dresser un comparatif, et ce pour des implémentations logicielles. Deuxièmement, aucune attaque par observation n'a été réalisée à l'encontre de ACORN, pourtant arrivé jusqu'au stade de finaliste. La seule analyse sécuritaire publiée concerne un test statistique sur une implémentation matérielle de ACORN, qui d'ailleurs exhibe de meilleurs résultats que d'autres algorithmes (dont ASCON), laissant supposer qu'il présente de meilleures caractéristiques face aux attaques par observation. Nous avons alors investiguer la définition d'un chemin d'attaque, avant de nous intéresser à un schéma de masquage. Nous avons abouti à une attaque dont nous avons démontré l'efficacité en pratique, sur le même microcontrôleur que celui utilisé dans le cadre de l'étude sur ChaCha. Cependant, cette attaque est particulièrement complexe puisqu'elle nécessite d'avoir recours à des moyens algébriques, plus précisément la satisfaisabilité d'une formule logique propositionnelle, afin de retrouver la clef à partir des fuites enregistrées. Ce résultat souligne le fait que lors de la conception d'un algorithme cryptographique, s'il est judicieux de faciliter l'intégration de contre-mesures contre les attaques physiques, il est également possible de jouer sur la manière dont les données secrètes interagissent avec les données en entrée, afin que les fuites par canaux auxiliaires soient liées à des valeurs intermédiaires complexes. Nous avons ensuite proposé un schéma de masquage à l'ordre 1 optimisé pour ACORN et ASCON. Il est intéressant à noter que pour chacun d'eux, il est possible d'appliquer la contre-mesure à un sous-ensemble de l'algorithme uniquement, ce qui permet de réduire drastiquement le surcoût de cette dernière. Finalement, nos mesures nous ont permis de conclure que si ACORN permet d'atteindre une implémentation plus compacte, ASCON est significativement plus efficace pour traiter des messages de petites tailles (*e.g.*, inférieurs à 128 octets), ce qui constitue un atout considérable pour les nombreux cas d'usage de l'IdO qui manipulent énormément de données en petites quantités. De plus, le facteur de pénalité en termes de performances est d'environ 2, ce qui est bien plus attractif que dans le cas de ChaCha. Pour couronner le tout, ASCON est extrêmement efficace sur les plateformes 64-bit puisque par définition, il opère sur des variables de cette taille. Cela a le mérite d'être souligné car si une implémentation efficace du côté de l'objet connecté est désirable, cette même propriété est cruciale côté serveur puisque ce dernier sera en charge de traiter les requêtes de tous les objets auxquels il est associé. Somme toute, ASCON a de sérieux arguments pour faire face aux autres algorithmes candidats au processus de standardisation du NIST.

7.2 Perspectives d'approfondissement

Si les résultats sus-mentionnés ont permis de répondre à certaines questions, ils ont également soulevé des problématiques que nous détaillons ci-dessous.

Concernant nos travaux sur ChaCha, une question laissée sans réponse par nos travaux est probablement celle concernant les fuites des différents types d'instruction. En effet, il serait intéressant d'étudier plus en détails la manière dont les accès mémoires au sein des implémentations logicielles facilitent une attaque par observation et par conséquent à quel point il est possible d'en jouer pour bénéficier d'une certaine protection à un coût minimal. Il est à noter que la difficulté d'exploiter ces opérations a également été soulignée dans le cas où le canal auxiliaire employé est la consommation de courant, sur un autre processeur [BDG16]. Si l'efficacité de minimiser les accès mémoires à l'encontre des attaques par observation serait avérée, le simple fait d'écrire la primitive cryptographique en langage Assembleur constituerait une contre-mesure efficace dans le cas où il serait possible de faire transiter sur le bus des données uniquement des valeurs intérmédiaires complexes à exploiter au sens algébrique. Aussi, l'important surcoût lié à l'intégration de contre-mesures de masquage pour les structures ARX pourrait constituer un frein à leur adoption pour les cas d'usage nécessitant une résistance aux attaques physiques. Par conséquent, de nouvelles méthodes pour calculer efficacement des additions modulaires sur des opérandes booléennes masquées permettraient de rendre ces algorithmes beaucoup plus attractifs.

Notre attaque contre ACORN souligne le fait qu'il est possible de rendre une attaque par canaux auxiliaire complexe par conception, en plus de favoriser l'intégration de contre-mesures adéquates. Nous espérons que notre travail pourra être utile à de futures conceptions de chiffrements symétriques basés sur des registres à rétroaction. Concernant le cas spécifique d'ACORN, dans le cadre d'une implémentation sécurisée, les autres contre-mesures que le masquage n'ont pas encore été étudiée. Notamment, les difficultés que nous avons rencontrées lorsque nous avons éxecuté l'attaque en pratique laissent penser qu'il pourrait être judicieux de tirer avantage du fait que la fonction de sélection utilisée fait en réalité référence aux bits de pseudo-aléa de chiffrement. En effet, au lieu de supprimer le calcul du bit chiffré durant la phase d'initialisation, mettre en place un mécanisme qui détermine aléatoirement si on stocke en RAM d'abord la valeur $f_i \oplus IV_i$ ou bien la valeur $ks_i \oplus IV_i$ compliquerait l'attaque en pratique. De cette manière, l'attaquant ne serait pas capable de déterminer à quelle combinaison de bits de clef correspond l'hypothèse retournée par l'attaque CPA, et donc serait incapable de construire un système d'équations booléennes correctes. Plus largement, les contre-mesures de désynchronisation semblent d'un grand intérêt contre l'attaque que nous avons introduite contre ACORN.

Nos travaux d'implémentation et de sécurisation d'ASCON pourront servir de base pour des analyses plus poussées durant le processus de standardisation du NIST. En effet, il reste à étudier s'il est possible de réduire la quantité de (pseudo-)aléa à générer, et si les contre-mesures de désynchronisation sont efficaces pour cet algorithme. Notre schéma de masquage est particulièrement efficace pour ASCON grâce à sa structure en éponge qui permet de limiter les nombres de permutation à sécuriser face aux attaques par canaux auxiliaire. Par conséquent, il est non seulement possible d'optimiser les coûts liés à l'intégration de contre-mesures au niveau de la primitive cryptographique sous-jacente, mais également au niveau du mode opératoire de cette dernière. Récemment, plusieurs travaux ont proposé des structures pour les chiffrements authentifiés qui permettent de limiter l'intégration de contre-mesures à certaines sous-fonctions de l'algorithme,

comme nous l'avons proposé pour ASCON [GPPS18, BGP+19]. Il sera intéressant de suivre les avancées dans ce domaine afin de voir jusqu'à quel point il est possible de limiter l'intégration de contre-mesures dès la phase de conception.

Finalement, toutes les contre-mesures étudiées dans le cadre de cette étude nécessitent la génération de nombres (pseudo-)aléatoires, problématique sur laquelle nous ne nous sommes pas attardés puisque cette fonctionnalité est présente sur la solution Secure Element de Trusted Objects. Dans le cas d'un microcontrôleur standard ne disposant pas de cette fonctionnalité, cela peut-être un réel défi. Un moyen courant est de construire un générateur pseudo-aléatoire déterministe à partir d'une primitive cryptographique, un secret et un état initial. Cependant, opérer de la sorte nécessiterait de protéger cette même primitive cryptographique contre les attaques physiques, introduisant ainsi un cercle vicieux sans fin. En plus d'avoir un important impact sur les performances, les défis techniques peuvent être nombreux. Pour cette raison, la recherche de schémas de masquage minimisant la quantité d'aléa à générer est un sujet de recherche actif. À titre d'exemple, il a été recemment démontré que les schémas de masquage à l'ordre 1 peuvent être calculés à l'aide de seulement deux bits aléatoires, avec une application à l'AES [GMKM18]. Une approche intéressante serait d'appliquer cette technique aux algorithmes cryptographiques légers étudiés dans ce manuscrit, dans le but d'estimer si le gain obtenu sur la génération de nombres aléatoires est plus significatif que la dégradation des performances induite par cette technique.

Bibliographie

[AA16] Ralph Ankele and Robin Ankele. Software Benchmarking of the 2nd round CAESAR Candidates. Cryptology ePrint Archive, Report 2016/740, 2016. https://eprint.iacr.org/2016/740.

[AAB+17] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, 2017. USENIX Association. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis.

[ABF+03] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault Attacks on RSA with CRT : Concrete Results and Practical Countermeasures. In *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 260–275, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. https://eprint.iacr.org/2002/073.pdf.

[ADN+10] Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson, and Assia Tria. When Clocks Fail : On Critical Paths and Clock Faults. In *Smart Card Research and Advanced Application*, pages 182–193, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. http://dl.ifip.org/db/conf/cardis/cardis2010/AgoyanDNRT10.pdf.

[ADO06] Gildas Avoine, Etienne Dysli, and Philippe Oechslin. Reducing Time Complexity in RFID Systems. In *Selected Areas in Cryptography*, pages 291–306. Springer Berlin Heidelberg, 2006. https://link.springer.com/chapter/10.1007/11693383_20.

[AFM17] Alexandre Adomnicai, Jacques J. A. Fournier, and Laurent Masson. Bricklayer Attack : A Side-Channel Analysis on the ChaCha Quarter Round. In *Progress in Cryptology – INDOCRYPT 2017*, pages 65–84. Springer International Publishing, 2017. https://eprint.iacr.org/2017/1021.pdf.

[AFM18a] A. Adomnicai, J. J. A. Fournier, and L. Masson. Hardware Security Threats Against Bluetooth Mesh Networks. In *2018 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9, May 2018.

[AFM18b] Alexandre Adomnicai, Jacques J. A. Fournier, and Laurent Masson. Practical Algebraic Side-Channel Attacks Against ACORN. In *Information Security and Cryptology – ICISC 2018*, pages 325–340. Springer International Publishing, 2018. https://s3.ap-northeast-2.amazonaws.com/journal-home/site/icisc/icisc_journals/paper_7_1_adomnicai.pdf.

[AFM18c] Alexandre Adomnicai, Jacques J.A. Fournier, and Laurent Masson. Masking the Lightweight Authenticated Ciphers ACORN and Ascon in Software. Cryptology ePrint Archive, Report 2018/708, 2018. https://eprint.iacr.org/2018/708.

[AG01] Mehdi-Laurent Akkar and Christophe Giraud. An Implementation of DES and AES, Secure against Some Attacks. In *Cryptographic Hardware and Embedded Systems — CHES 2001*, pages 309–318. Springer Berlin Heidelberg, 2001. https://link.springer.com/chapter/10.1007/3-540-44709-1_26.

[AK96] Ross Anderson and Markus Kuhn. Tamper Resistance : A Cautionary Note. In *Proceedings of the 2Nd Conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*, WOEC'96, pages 1–1, Berkeley, CA, USA, 1996. USENIX Association. http://dl.acm.org/citation.cfm?id=1267167.1267168.

[ALC+16] Alexandre Adomnicai, Benjamin Lac, Anne Canteaut, Jacques Jean-Alain Fournier, Laurent Masson, Renaud Sirdey, and Assia Tria. On the importance of considering physical attacks when implementing lightweight cryptography. In *Lightweight Cryptography Workshop | NIST*, 2016. https://hal-cea.archives-ouvertes.fr/cea-01436006.

[ALOW15] Alex Arbit, Yoel Livne, Yossef Oren, and Avishai Wool. Implementing public-key cryptography on passive RFID tags is practical. *International Journal of Information Security*, 14(1) :85–99, Feb 2015.

[App18] iOS Security. Technical report, Apple Inc., November 2018.

[ARM18] ARM. Cortex-M35P : A tamper-resistant Cortex-M processor with optional software isolation using TrustZone for Armv8-M. https://developer.arm.com/products/processors/cortex-m/cortex-m35p, 2018.

[Ava17] Roberto Avanzi. The QARMA Block Cipher Family. Almost MDS Matrices Over Rings With Zero Divisors, Nearly Symmetric Even-Mansour Constructions With Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes. *IACR Transactions on Symmetric Cryptology*, 2017. https://tosc.iacr.org/index.php/ToSC/article/view/583.

[BBC+08] Côme Berbain, Olivier Billet, Anne Canteaut, Nicolas Courtois, Henri Gilbert, Louis Goubin, Aline Gouget, Louis Granboulan, Cédric Lauradoux, Marine Minier, Thomas Pornin, and Hervé Sibert. *Sosemanuk, a Fast Software-Oriented Stream Cipher*, pages 98–118. Springer Berlin Heidelberg, 2008. http://www.ecrypt.eu.org/stream/p3ciphers/sosemanuk/sosemanuk_p3.pdf.

[BBR16] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Atomic-AES : A compact implementation of the AES encryption/decryption core. In *Progress in Cryptology – INDOCRYPT 2016*, pages 173–190. Springer, 2016. https://eprint.iacr.org/2016/927.pdf.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Message Authentication using Hash Functions- The HMAC Construction. *CryptoBytes*, 2, 1996.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In *Cryptographic Hardware and Embedded Systems - CHES 2004*, pages 16–29. Springer Berlin Heidelberg, 2004. https://www.iacr.org/archive/ches2004/31560016/31560016.pdf.

[BCZ18] Luk Bettale, Jean-Sébastien Coron, and Rina Zeitoun. Improved High-Order Conversion From Boolean to Arithmetic Masking. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(2) :22–45, May 2018. https://tches.iacr.org/index.php/TCHES/article/view/873/825.

[BDG16] Alex Biryukov, Daniel Dinu, and Johann Großschädl. Correlation Power Analysis of Lightweight Block Ciphers : From Theory to Practice. In *Applied Cryptography and Network Security*, pages 537–557. Springer International Publishing, 2016. https://link.springer.com/chapter/10.1007/978-3-319-39555-5_29.

[BDH+12] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Permutation-based encryption, authentication and authenticated encryption. Directions in Authenticated Ciphers – DIAC, 2012. https://keccak.team/files/KeccakDIAC2012.pdf.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *Advances in Cryptology — EUROCRYPT '97*, pages 37–51, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[BDLCU18] Alex Biryukov, Daniel Dinu, Yann Le Corre, and Aleksei Udovenko. Optimal First-Order Boolean Masking for Embedded IoT Devices. In *Smart Card Research and Advanced Applications*, pages 22–41. Springer International Publishing, 2018. http://orbilu.uni.lu/bitstream/10993/37740/1/Optimal_Masking.pdf.

[BDPVA] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The making of keccak.

[BDPVA07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge Functions. ECRYPT Hash Workshop, 2007. https://keccak.team/files/SpongeFunctions.pdf.

[BDPVA12] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the Sponge : Single-Pass Authenticated Encryption and Other Applications. In *Selected Areas in Cryptography*, pages 320–337. Springer Berlin Heidelberg, 2012. https://keccak.team/files/SpongeDuplex.pdf.

[Ber05] Daniel J. Bernstein. Cache-timing attacks on AES. Technical report, 2005.

[Ber08a] Daniel J. Bernstein. ChaCha, a variant of Salsa20. The Sate of the ART of Stream Ciphers - SASC, 2008. https://cr.yp.to/chacha/chacha-20080120.pdf.

[Ber08b] Daniel J. Bernstein. *The Salsa20 Family of Stream Ciphers*, pages 84–97. Springer Berlin Heidelberg, 2008. https://cr.yp.to/snuffle/812.pdf.

[BGG+15] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the Cost of Lazy Engineering for Masked Software Implementations. In *Smart Card Research and Advanced Applications*, pages 64–81. Springer International Publishing, 2015. https://eprint.iacr.org/2014/413.pdf.

[BGK+07] L. Batina, J. Guajardo, T. Kerins, N. Mentens, P. Tuyls, and I. Verbauwhede. Public-key cryptography for rfid-tags. In *Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW'07)*, pages 217–222, March 2007. https://www.esat.kuleuven.be/cosic/publications/article-821.pdf.

[BGP+19] Francesco Berti, Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. TEDT, a Leakage-Resilient AEAD mode for High (Physical) Security Applications. Cryptology ePrint Archive, Report 2019/137, 2019. https://eprint.iacr.org/2019/137.

[BGRV15] Josep Balasch, Benedikt Gierlichs, Oscar Reparaz, and Ingrid Verbauwhede. DPA, Bitslicing and Masking at 1 GHz. In *Cryptographic Hardware and Embedded Systems – CHES 2015*, pages 599–619. Springer Berlin Heidelberg, 2015. https://eprint.iacr.org/2015/727.pdf.

[BGS+05] Stephen C. Bono, Matthew Green, Adam Stubblefield, Ari Juels, Aviel D. Rubin, and Michael Szydlo. Security Analysis of a Cryptographically-enabled RFID Device. In *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*. USENIX Association, 2005. https://www.usenix.org/legacy/event/sec05/tech/bono/bono.pdf.

[BHG+13] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt. RIOT OS : Towards an OS for the Internet of Things. In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 79–80, April 2013.

[Bih97] Eli Biham. A Fast New DES Implementation in Software. In *Fast Software Encryption*, pages 260–272. Springer Berlin Heidelberg, 1997. http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-get.cgi/1997/CS/CS0891.pdf.

[Bil15] Begül Bilgin. *Threshold Implementations : as Countermeasure Against Higher-Order Differential Power Analysis*. PhD thesis, Netherlands, 2015. https://www.esat.kuleuven.be/cosic/publications/thesis-256.pdf.

[BJM+14] Lejla Batina, Domagoj Jakobovic, Nele Mentens, Stjepan Picek, Antonio de la Piedra, and Dominik Sisejkovic. S-box Pipelining Using Genetic Algorithms for High-Throughput AES Implementations : How Fast Can We Go? In *Progress in Cryptology – INDOCRYPT 2014*, pages 322–337, Cham, 2014. Springer International Publishing.

[BKL+07] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT : An Ultra-Lightweight Block Cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 450–466. Springer Berlin Heidelberg, 2007. https://www.iacr.org/archive/ches2007/47270450/47270450.pdf.

[BKL+17a] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. Gimli : A Cross-Platform Permutation. In *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 299–320. Springer International Publishing, 2017. https://cryptojedi.org/papers/gimli-20170627.pdf.

[BKL+17b] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. Gimli in two tweets. Twitter, 2017. https://twitter.com/TweetGimli.

[Bla79] G. R. Blakley. Safeguarding cryptographic keys. In *Managing Requirements Knowledge, International Workshop on(AFIPS)*, 1979. https://pdfs.semanticscholar.org/32d2/1ccc21a807627fcb21ea829d1acdab23be12.pdf.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated Encryption : Relations among Notions and Analysis of the Generic Composition Paradigm. In *Advances in Cryptology — ASIACRYPT 2000*, pages 531–545. Springer Berlin Heidelberg, 2000. https://eprint.iacr.org/2000/025.pdf.

[BP10] Joan Boyar and René Peralta. A New Combinational Logic Minimization Technique with Applications to Cryptology. In *Experimental Algorithms*, pages 178–189. Springer Berlin Heidelberg, 2010. https://eprint.iacr.org/2009/191.pdf.

[BP17] Alex Biryukov and Leo Perrin. State of the Art in Lightweight Symmetric Cryptography. Cryptology ePrint Archive, Report 2017/511, 2017. https://eprint.iacr.org/2017/511.

[BRSH18] Mathias Baert, Jen Rossey, Adnan Shahid, and Jeroen Hoebeke. The Bluetooth Mesh Standard : An Overview and Experimental Evaluation. *Sensors*, 18(8), 2018. http://www.mdpi.com/1424-8220/18/8/2409.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Advances in Cryptology — CRYPTO '97*, pages 513–525, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. https://eprint.iacr.org/2013/404.pdf.

[BSS+17] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. Notes on the design and analysis of SIMON and SPECK. Cryptology ePrint Archive, Report 2017/560, 2017. https://eprint.iacr.org/2017/560.pdf.

[BT16] Mihir Bellare and Björn Tackmann. The Multi-user Security of Authenticated Encryption : AES-GCM in TLS 1.3. In *Advances in Cryptology – CRYPTO 2016*, pages 247–276. Springer Berlin Heidelberg, 2016. https://eprint.iacr.org/2016/564.

[Bur14] Elie Bursztein. Speeding up and strengthening HTTPS connections for Chrome on Android . Technical report, April 2014.

[BZD+16] Hanno Böck, Aaron Zauner, Sean Devlin, Juraj Somorovsky, and Philipp Jovanovic. Nonce-Disrespecting Adversaries : Practical Forgery Attacks on GCM in TLS. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, Austin, TX, 2016. https://eprint.iacr.org/2016/475.pdf.

[Can11] Anne Canteaut. *A5/1*, pages 1–2. Springer US, Boston, MA, 2011. https://doi.org/10.1007/978-1-4419-5906-5_332.

[CB18] Paul Crowley and Eric Biggers. Adiantum : length-preserving encryption for entry-level processors. *IACR Transactions on Symmetric Cryptology*, 2018(4) :39–61, Dec. 2018. https://eprint.iacr.org/2018/720.pdf.

[cbc99] ISO/IEC 9797-1 :1999 Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1 : Mechanisms using a block cipher. Standard, International Organization for Standardization, 1999.

[CC05] Cecile Canovas and Jessy Clediere. What do S-boxes Say in Differential Side Channel Attacks? Cryptology ePrint Archive, Report 2005/311, 2005. https://eprint.iacr.org/2005/311.

[CG00] Jean-Sébastien Coron and Louis Goubin. On Boolean and Arithmetic Masking against Differential Power Analysis. In *Cryptographic Hardware and Embedded Systems — CHES 2000*, pages 231–237. Springer Berlin Heidelberg, 2000. http://www.goubin.fr/papers/boolean.pdf.

[CGP+12] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of Security Proofs from One Leakage Model to Another : A New Issue. In *Constructive Side-Channel Analysis and Secure Design*, pages 69–81. Springer Berlin Heidelberg, 2012. https://www.math.u-bordeaux.fr/~srenner/Cosade2012.pdf.

[CGTV15] Jean-Sébastien Coron, Johann Großschädl, Mehdi Tibouchi, and Praveen Kumar Vadnala. Conversion from Arithmetic to Boolean Masking with Logarithmic Complexity. In *Fast Software Encryption*, pages 130–149. Springer Berlin Heidelberg, 2015. https://eprint.iacr.org/2014/891.pdf.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *Advances in Cryptology — CRYPTO' 99*, pages 398–412. Springer Berlin Heidelberg, 1999. https://link.springer.com/chapter/10.1007/3-540-48405-1_26.

[CK09] Jean-Sébastien Coron and Ilya Kizhvatov. An Efficient Method for Random Delay Generation in Embedded Software. In *Cryptographic Hardware and Embedded Systems - CHES 2009*, pages 156–170. Springer Berlin Heidelberg, 2009. https://eprint.iacr.org/2009/419.pdf.

[CM16] Arka Rai Choudhuri and Subhamoy Maitra. Differential Cryptanalysis of Salsa and ChaCha – An Evaluation with a Hybrid Model. Cryptology ePrint Archive, Report 2016/377, 2016. https://eprint.iacr.org/2016/377.

[CMM13] M. Cazorla, K. Marquet, and M. Minier. Survey and benchmark of lightweight block ciphers for wireless sensor networks. In *2013 International*

Conference on Security and Cryptography (SECRYPT), July 2013. https://hal.inria.fr/hal-00918974/document.

[Cor14] Jean-Sébastien Coron. Higher Order Masking of Look-Up Tables. In *Advances in Cryptology – EUROCRYPT 2014*, pages 441–458. Springer Berlin Heidelberg, 2014. https://eprint.iacr.org/2013/700.pdf.

[Cor17] Jean-Sébastien Coron. High-Order Conversion from Boolean to Arithmetic Masking. In *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 93–114. Springer International Publishing, 2017. https://eprint.iacr.org/2017/252.pdf.

[CPR07] Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side Channel Cryptanalysis of a Higher Order Masking Scheme. In *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 28–44. Springer Berlin Heidelberg, 2007. https://iacr.org/archive/ches2007/47270028/47270028.pdf.

[CR17] Christophe Clavier and Léo Reynaud. Improved Blind Side-Channel Analysis by Exploitation of Joint Distributions of Leakages. In *Cryptographic Hardware and Embedded Systems – CHES 2017*. Springer International Publishing, 2017.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28. Springer Berlin Heidelberg, 2002. https://link.springer.com/chapter/10.1007/3-540-36400-5_3.

[CSF+08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008. https://www.rfc-editor.org/rfc/rfc5280.txt.

[CT03] Jean-Sébastien Coron and Alexei Tchulkine. A New Algorithm for Switching from Arithmetic to Boolean Masking. In *Cryptographic Hardware and Embedded Systems - CHES 2003*, pages 89–97. Springer Berlin Heidelberg, 2003. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.2516&rep=rep1&type=pdf.

[CZFB14] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. A Large-scale Analysis of the Security of Embedded Firmwares. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 95–110, Berkeley, CA, USA, 2014. USENIX Association. http://dl.acm.org/citation.cfm?id=2671225.2671232.

[CZP14] R. Callan, A. Zajic, and M. Prvulovic. A Practical Methodology for Measuring the Side-Channel Signal Available to the Attacker for Instruction-Level Events. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 242–254, Dec 2014.

[DAF+18] W. Diehl, A. Abdulgadir, F. Farahmand, J. Kaps, and K. Gaj. Comparison of cost of protection against differential power analysis of selected authenticated ciphers. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 147–152, April 2018. https://www.mdpi.com/2410-387X/2/3/26/pdf.

[DCK+15] Daniel Dinu, Yann Le Corre, Dmitry Khovratovich, Léo Perrin, Johann Großschädl, and Alex Biryukov. Triathlon of Lightweight Block Ciphers for the Internet of Things. Cryptology ePrint Archive, Report 2015/209, 2015. https://eprint.iacr.org/2015/209.

[Deb12] Blandine Debraize. Efficient and Provably Secure Methods for Switching from Arithmetic to Boolean Masking. In *Cryptographic Hardware and Embedded Systems – CHES 2012*, pages 107–121. Springer Berlin Heidelberg, 2012. https://www.iacr.org/archive/ches2012/74280103/74280103.pdf.

[DEK+16] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Victor Lomné, and Florian Mendel. Statistical Fault Attacks on Nonce-Based Authenticated Encryption Schemes. In *Advances in Cryptology – ASIACRYPT 2016*, pages 369–395. Springer Berlin Heidelberg, 2016. https://eprint.iacr.org/2016/616.pdf.

[DEMS16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Submission to the CAESAR competition, 2016. http://ascon.iaik.tugraz.at.

[DG17] Seyed Mahdi Darroudi and Carles Gomez. Bluetooth Low Energy Mesh Networks : A Survey. *Sensors*, 17(7), 2017. http://www.mdpi.com/1424-8220/17/7/1467.

[DGLC17] Daniel Dinu, Johann Großschädl, and Yann Le Corre. Efficient Masking of ARX-Based Block Ciphers Using Carry-Save Addition on Boolean Shares. In *Information Security*, pages 39–57. Springer International Publishing, 2017.

[dGPdLP+17] Wouter de Groot, Kostas Papagiannopoulos, Antonio de La Piedra, Erik Schneider, and Lejla Batina. Bitsliced Masking and ARM : Friends or Foes? In *Lightweight Cryptography for Security and Privacy*, pages 91–109. Springer International Publishing, 2017. https://eprint.iacr.org/2016/946.pdf.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22 :644–654, 1976. https://ee.stanford.edu/~hellman/publications/24.pdf.

[dMS10] Giacomo de Meulenaer and François-Xavier Standaert. Stealthy Compromise of Wireless Sensor Nodes with Power Analysis Attacks. In *Mobile Lightweight Wireless Systems*, pages 229–242. Springer Berlin Heidelberg, 2010. https://perso.uclouvain.be/fstandae/PUBLIS/77.pdf.

[DPU+16] Daniel Dinu, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov. Design Strategies for ARX with Provable Bounds : Sparx and LAX. In *Advances in Cryptology – ASIACRYPT 2016*, pages 484–513. Springer Berlin Heidelberg, 2016. https://eprint.iacr.org/2016/984.pdf.

[DRA16] Prakash Dey, Raghvendra Singh Rohit, and Avishek Adhikari. Full key recovery of acorn with a single fault. *Journal of Information Security and Applications*, 29 :57 – 64, 2016. http://www.sciencedirect.com/science/article/pii/S2214212616300138.

[Dwo01] Morris J. Dworkin. SP 800-38A 2001 Edition. Recommendation for Block Cipher Modes of Operation : Methods and Techniques. Technical report, Gaithersburg, MD, United States, 2001. https://csrc.nist.gov/publications/detail/sp/800-38a/final.

[Dwo07] Morris J. Dworkin. SP 800-38D. Recommendation for Block Cipher Modes of Operation : Galois/Counter Mode (GCM) and GMAC. Technical report, Gaithersburg, MD, United States, 2007.

[eBA] eBAEAD. Measurements of CAESAR candidates, indexed by machine. https://bench.cr.yp.to/results-caesar.html.

[ECR18] ECRYPT. Algorithms, Key Size and Protocols Report (2018), 2018. http://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf.

[Edg19] Jake Edge. Adiantum : encryption for the low end. Linux Weekly News, Jan. 2019. https://lwn.net/Articles/776721/.

[Egl15] Peter R. Egli. LPWAN Technologies for IoT and M2M Scenarios. Technical report, 2015. https://www.slideshare.net/PeterREgli/lpwan.

[FDA+18] Farnoud Farahmand, William Diehl, Abubakr Abdulgadir, Jens-Peter Kaps, and Kris Gaj. Improved Lightweight Implementations of CAESAR Authenticated Ciphers. Cryptology ePrint Archive, Report 2018/573, 2018. https://eprint.iacr.org/2018/573.

[FFZ+11] Dengguo Feng, Xiutao Feng, Wentao Zhang, Xiubin Fan, and Chuankun Wu. Loiss : A Byte-Oriented Stream Cipher. In *Coding and Cryptology*, pages 109–125. Springer Berlin Heidelberg, 2011. https://eprint.iacr.org/2010/489.pdf.

[FK14] Ingar Fredriksen and Paal Kastnes. 8 Bit or 32 Bit? Choosing Your Next Design's MCU, Dec 2014. https://www.electronicdesign.com/microcontrollers/8-bit-or-32-bit-choosing-your-next-design-s-mcu.

[FSZW] Lingzhi Fu, Xiang Shen, Linghao Zhu, and Junyu Wang. A low-cost UHF RFID tag chip with AES cryptography engine. *Security and Communication Networks*, 7(2) :365–375. https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.723.

[GBC+08] Benedikt Gierlichs, Lejla Batina, Christophe Clavier, Thomas Eisenbarth, Aline Gouget, Helena Handschuh, Timo Kasper, Kerstin Lemke-Rust, Stefan Mangard, Amir Moradi, and Elisabeth Oswald. Susceptibility of eSTREAM candidates towards side channel analysis. pages 123–150, 07 2008. https://www.esat.kuleuven.be/cosic/publications/article-1030.pdf.

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual Information Analysis. In *Cryptographic Hardware and Embedded Systems – CHES 2008*, pages 426–442. Springer Berlin Heidelberg, 2008. https://eprint.iacr.org/2007/198.pdf.

[GG14] M. Goll and S. Gueron. Vectorization on ChaCha Stream Cipher. In *2014 11th International Conference on Information Technology : New Generations*, pages 612–615, April 2014. https://eprint.iacr.org/2013/759.pdf.

[GLSV15] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varıcı. LS-Designs : Bitslice Encryption for Efficient Masked Software Implementations. In *Fast Software Encryption*, pages 18–37. Springer Berlin Heidelberg, 2015. https://hal.inria.fr/hal-01093491/document.

[GMKM18] Hannes Gross, Lauren De Meyer, Martin Krenn, and Stefan Mangard. Masking the AES with Only Two Random Bits. Cryptology ePrint Archive, Report 2018/1007, 2018. https://eprint.iacr.org/2018/1007.

[Gou01] Louis Goubin. A Sound Method for Switching between Boolean and Arithmetic Masking. In *Cryptographic Hardware and Embedded Systems — CHES 2001*, pages 3–15. Springer Berlin Heidelberg, 2001. http://www.goubin.fr/papers/arith-final.pdf.

[GPPS18] Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Authenticated Encryption with Nonce Misuse and Physical Leakages : Definitions, Separation Results, and Leveled Constructions. Cryptology ePrint Archive, Report 2018/484, 2018. https://eprint.iacr.org/2018/484.

[GST14] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis. In *Advances in Cryptology – CRYPTO 2014*, pages 444–461, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[Gue09] Shay Gueron. Intel's New AES Instructions for Enhanced Performance and Security. In *Fast Software Encryption*, pages 51–66. Springer Berlin Heidelberg, 2009.

[Gue10] Shay Gueron. Intel® Advanced Encryption Standard (AES) New Instructions Set, 2010. https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf.

[GWDE15] H. Groß, E. Wenger, C. Dobraunig, and C. Ehrenhöfer. Suit up! – Made-to-Measure Hardware Implementations of ASCON. In *2015 Euromicro Conference on Digital System Design*, pages 645–652, 2015. https://eprint.iacr.org/2015/034.pdf.

[GWDE17] Hannes Gross, Erich Wenger, Christoph Dobraunig, and Christoph Ehrenhfer. Ascon Hardware Implementations and Side-channel Evaluation. *Microprocess. Microsyst.*, 52(C) :470–479, 2017. http://www.sciencedirect.com/science/article/pii/S0141933116302721.

[HAI+17] Naofumi Homma, Kazumaro Aoki, Tetsu Iwata, Kazuto Ogawa, Hisashi Oguma, Kazuo Sakiyama, Kyoji Shibutani, Daisuke Suzuki, Yuichiro Nariyoshi, Kazuhiko Minematsu, Hideyuki Miyake, and Dai Watanabe. CRYPTREC Cryptographic Technology Guideline (Lightweight Cryptography), 2017. https://www.cryptrec.go.jp/report/cryptrec-gl-2003-2016en.pdf.

[HC14] T. Hongsongkiat and P. Chongstitvatana. AES implementation for RFID Tags : The hardware and software approaches. In *2014 International Computer Science and Engineering Conference (ICSEC)*, pages 118–123, July 2014.

[HGDM+11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine Learning in Side-Channel Analysis : A First Study. *Journal of Cryptographic Engineering*, 2011. https://www.esat.kuleuven.be/cosic/publications/article-2075.pdf.

[HJ08] Martin Hell and Thomas Johansson. Breaking the F-FCSR-H Stream Cipher in Real Time. In *Advances in Cryptology - ASIACRYPT 2008*, pages 557–569. Springer Berlin Heidelberg, 2008. https://www.iacr.org/archive/asiacrypt2008/53500563/53500563.pdf.

[HLK04] M. Honkanen, A. Lappetelainen, and K. Kivekas. Low end extension for Bluetooth. In *IEEE Radio and Wireless Conference*, pages 199–202, Sep. 2004. https://ieeexplore.ieee.org/document/1389107?arnumber=1389107.

[HO12] N. Hanley and M. ONeill. Hardware Comparison of the ISO/IEC 29192-2 Block Ciphers. In *2012 IEEE Computer Society Annual Symposium on VLSI*, pages 57–62, Aug 2012.

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES Smart Card Implementation Resistant to Power Analysis Attacks. In *Applied Cryptography and Network Security*, pages 239–252. Springer Berlin Heidelberg, 2006. https://pdfs.semanticscholar.org/8f0c/8df7a81d53d7227c2eea6552c9cd4b1a846d.pdf.

[HT16] Michael Hutter and Michael Tunstall. Constant-Time Higher-Order Boolean-to-Arithmetic Masking. Cryptology ePrint Archive, Report 2016/1023, 2016. https://eprint.iacr.org/2016/1023.pdf.

[HTT18] Viet Tung Hoang, Stefano Tessaro, and Aishwarya Thiruvengadam. The Multi-user Security of GCM, Revisited : Tight Bounds for Nonce Randomization. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1429–1440, 2018. https://eprint.iacr.org/2018/993.

[IAN19] IANIX. ChaCha Usage & Deployment, 2019. https://ianix.com/pub/chacha-deployment.html.

[IK03] Tetsu Iwata and Kaoru Kurosawa. OMAC : One-Key CBC MAC. In *Fast Software Encryption*, pages 129–153. Springer Berlin Heidelberg, 2003. https://eprint.iacr.org/2002/180.pdf.

[Inc17] Qualcomm Technologies Inc. Pointer Authentication on ARMv8.3, 2017. https://www.qualcomm.com/media/documents/files/whitepaper-pointer-authentication-on-armv8-3.pdf.

[ISO09] Technologies de l'information – Techniques de sécurité – Critères d'évaluation pour la sécurité TI. Standard, International Organization for Standardization, Geneva, CH, 2009.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits : Securing Hardware against Probing Attacks. In *Advances in Cryptology - CRYPTO 2003*, pages 463–481. Springer Berlin Heidelberg, 2003. https://people.eecs.berkeley.edu/~daw/papers/privcirc-crypto03.pdf.

[JB17] B. Jungk and S. Bhasin. Don't fall into a trap : Physical side-channel analysis of ChaCha20-Poly1305. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 1110–1115, March 2017.

[JN00] Tommi A. Junttila and Ilkka Niemelä. Towards an Efficient Tableau Method for Boolean Circuit Satisfiability Checking. In *Computational Logic — CL 2000*, pages 553–567. Springer Berlin Heidelberg, 2000. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.79.8102&rep=rep1&type=pdf.

[JPS18] Bernhard Jungk, Richard Petri, and Marc Stöttinger. Efficient Side-Channel Protections of ARX Ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3) :627–653, Aug. 2018. https://eprint.iacr.org/2018/693.pdf.

[Kan15] David Kanter. Intel's Sandy Bridge Microarchitecture, 2015. https://www.realworldtech.com/sandy-bridge/.

[KBA+79] W. A. Kolasinski, J. B. Blake, J. K. Anthony, W. E. Price, and E. C. Smith. Simulation of Cosmic-Ray Induced Soft Errors and Latchup in Integrated-Circuit Computer Memories. *IEEE Transactions on Nuclear Science*, 26(6) :5087–5091, Dec 1979.

[KDK+14] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping Bits in Memory Without Accessing Them : An Experimental Study of DRAM Disturbance Errors. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 361–372, June 2014. https://users.ece.cmu.edu/~yoonguk/papers/kim-isca14.pdf.

[Ker83] Auguste Kerckhoffs. La cryptographie militaire. IX :5–83, Jan 1883. http://www.petitcolas.net/fabien/kerckhoffs/.

[KHF+19] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks : Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019. https://spectreattack.com/spectre.pdf.

[KHYK+17] Sachin Kumar, Jawad Haj-Yahya, Mustafa Khairallah, Mahmoud A. Elmohr, and Anupam Chattopadhyay. A Comprehensive Performance Analysis of Hardware Implementations of CAESAR Candidates. Cryptology ePrint Archive, Report 2017/1261, 2017. https://eprint.iacr.org/2017/1261.

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

[Kob87] Neal Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177) :203–209, 1987. http://www.jstor.org/stable/2007884.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[Kön08] Robert Könighofer. A Fast and Cache-Timing Resistant Implementation of the AES. In *Topics in Cryptology – CT-RSA 2008*, pages 187–202. Springer Berlin Heidelberg, 2008.

[KPB+17] S. V. D. Kumar, S. Patranabis, J. Breier, D. Mukhopadhyay, S. Bhasin, A. Chattopadhyay, and A. Baksi. A Practical Fault Attack on ARX-Like Ciphers with a Case Study on ChaCha20. In *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 33–40, Sep. 2017. http://conferenze.dei.polimi.it/FDTC17/shared/FDTC%202017%20-%20session%202.3.pdf.

[KRJ14] Mohamed Karroumi, Benjamin Richard, and Marc Joye. Addition with Blinded Operands. In *Constructive Side-Channel Analysis and Secure Design*, pages 41–55. Springer International Publishing, 2014.

[KS09] Emilia Käsper and Peter Schwabe. Faster and Timing-Attack Resistant AES-GCM. In *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 1–17. Springer-Verlag Berlin Heidelberg, 2009. https://cryptojedi.org/papers/aesbs-20090616.pdf.

[LCM+16] A. Langley, W. Chang, N. Mavrogiannopoulos, J. Strombergson, and S. Josefsson. ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS). RFC 7905, June 2016. https://tools.ietf.org/html/rfc7905.

[LDLL14] Yanis Linge, Cécile Dumas, and Sophie Lambert-Lacroix. Using the Joint Distributions of a Cryptographic Function in Side Channel Analysis. In *Constructive Side-Channel Analysis and Secure Design*, pages 199–213. Springer International Publishing, 2014. https://eprint.iacr.org/2013/859.pdf.

[Lec17] Tedelyne Lecroy. WaveRunner 6 Zi Oscilloscopes, 2017. https://cdn.teledynelecroy.com/files/pdf/waverunner-6zi-datasheet.pdf.

[LSG+18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown : Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018. https://meltdownattack.com/meltdown.pdf.

[LSP04] Kerstin Lemke, Kai Schramm, and Christof Paar. DPA on n-Bit Sized Boolean and Arithmetic Operations and Its Application to IDEA, RC6, and the HMAC-Construction. In *Cryptographic Hardware and Embedded Systems - CHES 2004*, pages 205–219. Springer Berlin Heidelberg, 2004. https://link.springer.com/chapter/10.1007/978-3-540-28632-5_15.

[Man04] Stefan Mangard. Hardware Countermeasures against DPA – A Statistical Analysis of Their Effectiveness. In *Topics in Cryptology – CT-RSA 2004*, pages 222–235. Springer Berlin Heidelberg, 2004.

[Max] James Clerk Maxwell. [a dynamical theory of the electromagnetic field. *Philosophical Transactions of the Royal Society of London*, pages 459–513.

[McN99] David McNett. US Governement's Encryption Standard Broken in Less than a Day. Technical report, 1999. http://www.distributed.net/images/d/d7/19990119_-_PR_-_release-des3.pdf.

[MDH+13] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, and E. Encrenaz. Electromagnetic Fault Injection : Towards a Fault Model on a 32-bit Microcontroller. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 77–88, Aug 2013. https://arxiv.org/pdf/1402.6421.pdf.

[MDS99] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Investigations of Power Analysis Attacks on Smartcards. In *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, WOST'99. USENIX Association, 1999. https://www.usenix.org/legacy/events/smartcard99/full_papers/messerges/messerges.pdf.

[Meg16] R. Meghana. An overview of the 6th generation Intel Core processor (code-named Skylake), 2016. https://software.intel.com/en-us/articles/an-overview-of-the-6th-generation-intel-core-processor-code-named-skylake.

[Mel18] T. Melamed. An Active Man-In-The-Middle Attack on Bluetooth Smart Devices. In *International Journal of Safety and Security Engineering*, volume 8, pages 200–211, 2018.

[MEO15] D. McCann, K. Eder, and E. Oswald. Characterising and Comparing the Energy Consumption of Side Channel Attack Countermeasures and Lightweight Cryptography on Embedded Devices. In *2015 International Workshop on Secure Internet of Things (SIoT)*, pages 65–71, Los Alamitos, CA, USA, sep 2015. IEEE Computer Society.

[Mes00] Thomas S. Messerges. Securing the AES Finalists Against Power Analysis Attacks. In *Fast Software Encryption*, pages 150–164. Springer Berlin Heidelberg, 2000. https://link.springer.com/chapter/10.1007/3-540-44706-7_11.

[Mil86] Victor S. Miller. Use of Elliptic Curves in Cryptography. In *Advances in Cryptology — CRYPTO '85 Proceedings*, pages 417–426, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.

[MMH+14] Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. Chaskey : An Efficient MAC Algorithm for 32-bit Microcontrollers. Cryptology ePrint Archive, Report 2014/386, 2014. https://eprint.iacr.org/2014/386.

[MMPS09] Amir Moradi, Nima Mousavi, Christof Paar, and Mahmoud Salmasizadeh. A Comparative Study of Mutual Information Analysis under a Gaussian Assumption. In *Information Security Applications*, pages 193–205. Springer Berlin Heidelberg, 2009. https://www.ei.ruhr-uni-bochum.de/media/crypto/veroeffentlichungen/2010/09/07/mia_cpa.pdf.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks : Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag, 2007.

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In *Topics in Cryptology – CT-RSA 2005*, pages 351–365. Springer Berlin Heidelberg, 2005. http://www.matpack.de/personal/Mangard_Popp_Gammel_CTRSA2005.pdf.

[MPJ11] Thrinatha R. Mutchukota, Saroj Kumar Panigrahy, and Sanjay Kumar Jena. Man-in-the-Middle Attack and Its Countermeasure in Bluetooth Secure Simple Pairing. In *Computer Networks and Intelligent Computing*, pages 367–376. Springer Berlin Heidelberg, 2011.

[MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully Attacking Masked AES Hardware Implementations. In *Cryptographic Hardware and Embedded Systems – CHES 2005*, pages 157–171. Springer Berlin Heidelberg, 2005. https://www.iacr.org/archive/ches2005/012.pdf.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking Cryptographic Implementations Using Deep Learning Techniques. In *Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer International Publishing, 2016.

[Mur18] Murata. Sub-G Module Data Sheet, 2018. https://wireless.murata.com/RFM/data/type_abz.pdf.

[MW78] T. C. May and M. H. Woods. A New Physical Mechanism for Soft Errors in Dynamic Memories. In *16th International Reliability Physics Symposium*, pages 33–40, April 1978.

[NIS] Nistir 8114 report on lightweight cryptography. NIST Internal or Interagency Reports. https://csrc.nist.gov/publications/detail/nistir/8114/final.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In *Information and Communications Security*, pages 529–545. Springer Berlin Heidelberg, 2006. https://www.esat.kuleuven.be/cosic/publications/article-847.pdf.

[OBSC10] Dag Arne Osvik, Joppe W. Bos, Deian Stefan, and David Canright. Fast Software AES Encryption. In *Fast Software Encryption*, pages 75–93, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. https://www.iacr.org/archive/fse2010/61470076/61470076.pdf.

[OC16] Colin O'Flynn and Zhizhang Chen. Power Analysis Attacks Against IEEE 802.15.4 Nodes. In *Constructive Side-Channel Analysis and Secure Design*, pages 55–70. Springer International Publishing, 2016. https://eprint.iacr.org/2015/529.pdf.

[OMPR05] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A Side-Channel Analysis Resistant Description of the AES S-Box. In *Fast Software Encryption*, pages 413–423. Springer Berlin Heidelberg, 2005. https://www.iacr.org/archive/fse2005/35570401/35570401.pdf.

[Pag02] Dan Page. Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel. *IACR Cryptology ePrint Archive*, 2002 :169, 2002.

[PC15] Gordon Procter and Carlos Cid. On Weak Keys and Forgery Attacks Against Polynomial-Based MAC Schemes. *J. Cryptol.*, 28(4) :769–795, October 2015. https://eprint.iacr.org/2013/144.pdf.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-Trace Side-Channel Attacks on Masked Lattice-Based Encryption. In *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 513–533. Springer International Publishing, 2017. https://eprint.iacr.org/2017/594.pdf.

[PR09] Emmanuel Prouff and Matthieu Rivain. Theoretical and Practical Aspects of Mutual Information Based Side Channel Analysis. In *Applied Cryptography and Network Security*, pages 499–518. Springer Berlin Heidelberg, 2009. http://www.matthieurivain.com/files/acns09.pdf.

[PV17] Kostas Papagiannopoulos and Nikita Veshchikov. Mind the Gap : Towards Secure 1st-Order Masking in Software. In *Constructive Side-Channel Analysis and Secure Design*, pages 282–297, 07 2017. https://eprint.iacr.org/2017/345.pdf.

[QS01] Jean-Jacques Quisquater and David Samyde. ElectroMagnetic Analysis (EMA) : Measures and Counter-measures for Smart Cards. In *Smart Card Programming and Security*, pages 200–210, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[RAL17] Tiago B. S. Reis, Diego F. Aranha, and Julio López. PRESENT Runs Fast : Efficient and Secure Implementation in Software. In *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 644–664. Springer International Publishing, 2017. https://ches.iacr.org/2017/slides/ches2017s11t2.pdf.

[RB08] Matthew Robshaw and Olivier Billet. *New Stream Cipher Designs : The eSTREAM Finalists*. Springer Berlin Heidelberg, 2008.

[reg14] Décision 2014-1263 du 6 novembre 2014 fixant les conditions d'utilisation des fréquences radioélectriques par des dispositifs à courte portée. Technical report, 2014. https://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000030160927.

[Res00] E. Rescorla. HTTP Over TLS. RFC 2818, May 2000. https://www.rfc-editor.org/rfc/rfc2818.txt.

[Res17] Allied Market Research. Microcontroller Market by Product Type (8-Bit, 16 Bit, and 32 Bit) and Application (Industrial, Consumer Goods, Automotive, Communication, and Computers) - Global Opportunity Analysis and Industry Forecast, 2014-2022. Technical report, January 2017. https://www.alliedmarketresearch.com/microcontrollers-market.

[Res18] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018. https://tools.ietf.org/html/rfc5246.

[Rog02] Phillip Rogaway. Authenticated-encryption with Associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 98–107, 2002. http://web.cs.ucdavis.edu/~rogaway/papers/ad.pdf.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably Secure Higher-Order Masking of AES. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 413–427. Springer Berlin Heidelberg, 2010. https://eprint.iacr.org/2010/441.pdf.

[RPD09] Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-Order Masking and Shuffling for Software Implementations of Block Ciphers. In *Cryptographic Hardware and Embedded Systems - CHES 2009*, pages 171–188, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. https://eprint.iacr.org/2009/420.pdf.

[RS10] Mathieu Renauld and François-Xavier Standaert. Algebraic Side-Channel Attacks. In *Information Security and Cryptology*, pages 393–410. Springer Berlin Heidelberg, 2010. https://eprint.iacr.org/2009/279.pdf.

[RS16] Ronald L. Rivest and Jacob C. N. Schuldt. Spritz—a spongy RC4-like stream cipher and hash function. Cryptology ePrint Archive, Report 2016/856, 2016. https://eprint.iacr.org/2016/856.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Commun. ACM*, 21(2) :120–126, February 1978. http://doi.acm.org/10.1145/359340.359342.

[RSWO17] E. Ronen, A. Shamir, A. Weingarten, and C. O'Flynn. IoT Goes Nuclear : Creating a ZigBee Chain Reaction. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 195–212, May 2017. https://eprint.iacr.org/2016/1047.pdf.

[SA02] Sergei P. Skorobogatov and Ross J. Anderson. Optical Fault Induction Attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 2–12, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[Saa12] Markku-Juhani Olavi Saarinen. Cycling Attacks on GCM, GHASH and Other Polynomial MACs and Hashes. In *Fast Software Encryption*, pages 216–225. Springer Berlin Heidelberg, 2012. https://eprint.iacr.org/2011/202.pdf.

[Sak87] K. Sakamura. The Tron Project. *IEEE Micro*, 7(2) :8–14, April 1987.

[SBM18] Pascal Sasdrich, René Bock, and Amir Moradi. Threshold Implementation in Software. In *Constructive Side-Channel Analysis and Secure Design*, pages 227–244. Springer International Publishing, 2018. https://eprint.iacr.org/2018/189.pdf.

[Sch17] Alex Schiffer. How a Fish Tank Helped Hack a Casino, July 2017. https://www.washingtonpost.com/news/innovations/wp/2017/07/21/how-a-fish-tank-helped-hack-a-casino.

[SD17] Niels Samwel and Joan Daemen. DPA on Hardware Implementations of Ascon and Keyak. In *Proceedings of the Computing Frontiers Conference*, CF'17, pages 415–424, New York, NY, USA, 2017. ACM.

[Sem16] Semtech. SX1276/77/78/79 - 137 MHz to 1020 MHz Low Power Long Range Transceiver, 2016. https://www.semtech.com/uploads/documents/DS_SX1276-7-8-9_W_APP_V5.pdf.

[Sem18] Nordic Semiconductor. nRF52840, 2018. http://www.nordicsemi.com/eng/Products/nRF52840.

[Sha79] Adi Shamir. How to Share a Secret. *Commun. ACM*, pages 612–613, November 1979. https://cs.jhu.edu/~sdoshi/crypto/papers/shamirturing.pdf.

[SHAS08] Takeshi Sugawara, Naofumi Homma, Takafumi Aoki, and Akashi Satoh. High-performance ASIC implementations of the 128-bit block cipher CLEFIA. In *2008 IEEE International Symposium on Circuits and Systems*, pages 2925–2928, May 2008.

[SIG17] Bluetooth SIG. Mesh Profile Specification 1.0. July 2017. https://www.bluetooth.com/specifications/mesh-specifications.

[SM87] Akihiro Shimizu and Shoji Miyaguchi. Fast Data Encipherment Algorithm FEAL. In *Advances in Cryptology — EUROCRYPT' 87*, pages 267–278. Springer Berlin Heidelberg, 1987.

[SM16] Tobias Schneider and Amir Moradi. Leakage 1ssessment Methodology. *Journal of Cryptographic Engineering*, 6 :85–99, 2016. https://eprint.iacr.org/2015/207.pdf.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT Solvers to Cryptographic Problems. In *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, pages 244–257, 2009. http://planete.inrialpes.fr/~soos/publications/Extending_SAT_2009.pdf.

[SNCC18] Arbia Riahi Sfar, Enrico Natalizio, Yacine Challal, and Zied Chtourou. A roadmap for security challenges in the Internet of Things. *Digital Communications and Networks*, 4(2) :118 – 137, 2018.

[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. Springer Berlin Heidelberg, 2006. https://pdfs.semanticscholar.org/74ee/60ec27ea2f27b9c3a1f1b11c3951f4010e10.pdf.

[SPB+79] L. L. Sivo, J. C. Peden, M. Brettschneider, W. Price, and P. Pentecost. Cosmic Ray-Induced Soft Errors in Static MOS Memory Cells. *IEEE Transactions on Nuclear Science*, 26(6) :5041–5047, Dec 1979.

[SS17] Peter Schwabe and Ko Stoffelen. All the AES You Need on Cortex-M3 and M4. In *Selected Areas in Cryptography – SAC 2016*, pages 180–194, Cham, 2017. Springer International Publishing. https://eprint.iacr.org/2016/714.pdf.

[SSA+07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit Blockcipher CLEFIA. In *Fast Software Encryption*, pages 181–195. Springer-Verlag, 2007. https://iacr.org/archive/fse2007/45930182/45930182.pdf.

[SSMC17] Akhilesh Siddhanti, Santanu Sarkar, Subhamoy Maitra, and Anupam Chattopadhyay. Differential Fault Attack on Grain v1, ACORN v3 and Lizard. In *Security, Privacy, and Applied Cryptography Engineering – SPACE 2017*, 2017. https://eprint.iacr.org/2017/678.pdf.

[Sta17] François-Xavier Standaert. How (not) to Use Welch's T-test in Side-Channel Security Evaluations. Cryptology ePrint Archive, Report 2017/138, 2017. https://eprint.iacr.org/2017/138.

[STM15] STMicroelectronics. Ultra-low-power 32-bit MCU Arm-based Cortex-M0+, up to 192KB Flash, 20KB SRAM, 6KB EEPROM, USB, ADC, DACs, 2015. https://www.st.com/resource/en/datasheet/stm32l072cz.pdf.

[STM16] STMicroelectronics. Low & medium-density value line, advanced ARM-based 32-bit MCU with 16 to 128KB Flash, 12 timers, ADC, DAC & 8 comm interfaces, 2016. https://www.st.com/resource/en/datasheet/stm32f100rb.pdf.

[Str10] Daehyun Strobel. Side Channel Analysis Attacks on Stream Ciphers. Master's thesis, Ruhr-Universität Bochum, 2010. https://www.emsec.rub.de/media/crypto/attachments/files/2010/04/mt_strobel.pdf.

[SW49] Claude E. Shannon and Warren Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana and Chicago, 1949.

[SWP03] Kai Schramm, Thomas Wollinger, and Christof Paar. A New Class of Collision Attacks and Its Application to DES. In *Fast Software Encryption*, pages 206–222. Springer Berlin Heidelberg, 2003. https://link.springer.com/chapter/10.1007/978-3-540-39887-5_16.

[TDSG03] Elena Trichina, Domenico De Seta, and Lucia Germani. Simplified Adaptive Multiplicative Masking for AES. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 187–197. Springer Berlin Heidelberg, 2003. https://link.springer.com/content/pdf/10.1007/3-540-36400-5_15.pdf.

[TDSK18] M. Tempelmeier, F. De Santis, G. Sigl, and J. Kaps. The CAESAR-API in the real world — Towards a fair evaluation of hardware CAESAR candidates. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 73–80, April 2018. https://ieeexplore.ieee.org/document/8383893.

[Teca] Langer EMV Technik. PA 303 BNC, Preamplifier 100 kHz to 3 GHz. https://www.langer-emv.de/en/product/preamplifier/37/pa-303-bnc-set-preamplifier-100-khz-up-to-3-ghz/519/pa-303-bnc-preamplifier-100-khz-up-to-3-ghz/41.

[Tecb] Langer EMV Technik. RF-U 5-2, H-Field Probe 30 MHz up to 3 GHz. https://www.langer-emv.de/en/product/rf-passive-30-mhz-3-ghz/35/rf-u-5-2-h-field-probe-30-mhz-up-to-3-ghz/16.

[Tex06] Texas Instruments. Efficient Multiplication and Division Using MSP430 MCUs. http://www.ti.com/lit/an/slaa329a/slaa329a.pdf, September 2006.

[TH08] Stefan Tillich and Christoph Herbst. Attacking State-of-the-Art Software Countermeasures– A Case Study for AES. In *Cryptographic Hardware and Embedded Systems – CHES 2008*, pages 228–243. Springer Berlin Heidelberg, 2008. https://www.iacr.org/archive/ches2008/51540224/51540224.pdf.

[THM+07] Michael Tunstall, Neil Hanley, Robert McEvoy, Claire Whelan, Colin Murphy, and William P Marnane. Correlation Power Analysis of Large Word Sizes. IET Irish Signals and Systems Conference (ISSC) 2007, 2007. http://www.geocities.ws/mike.tunstall/papers/THMWMM.pdf.

[Tri03] Elena Trichina. Combinational Logic Design for AES SubByte Transformation on Masked Data. Cryptology ePrint Archive, Report 2003/236, 2003. https://eprint.iacr.org/2003/236.

[TS95] H. Takada and K. Sakamura. Compact, low-cost, but real-time distributed computing for computer augmented environments. In *Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pages 56–63, Aug 1995.

[TSS+03] Yukiyasu Tsunoo, Teruo Saito, Tomoyasu Suzaki, Maki Shigeri, and Hiroshi Miyauchi. Cryptanalysis of DES Implemented on Computers with Cache. In *Cryptographic Hardware and Embedded Systems - CHES 2003*, pages 62–76, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[TW15] Biaoshuai Tao and Hongjun Wu. Improving the Biclique Cryptanalysis of AES. In *Information Security and Privacy*, pages 39–56, Cham, 2015. Springer International Publishing.

[UMHA16] Rei Ueno, Sumio Morioka, Naofumi Homma, and Takafumi Aoki. A High Throughput/Gate AES Hardware Architecture by Compressing Encryption and Decryption Datapaths. In *Cryptographic Hardware and Embedded Systems – CHES 2016*, pages 538–558, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. https://eprint.iacr.org/2016/595.

[VCMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against Side-Channel Attacks : A Comprehensive Study with Cautionary Note. In *Advances in Cryptology – ASIACRYPT 2012*, pages 740–757. Springer Berlin Heidelberg, 2012. https://www.iacr.org/archive/asiacrypt2012/76580728/76580728.pdf.

[VCS10] Nicolas Veyrat-Charvillon and François-Xavier Standaert. Adaptive Chosen-Message Side-Channel Attacks. In *Applied Cryptography and Network Security*, pages 186–199, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. https://perso.uclouvain.be/fstandae/PUBLIS/80.pdf.

[Ven10] Alexandre Venelli. Efficient Entropy Estimation for Mutual Information Analysis Using B-Splines. In *Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices*, pages 17–30. Springer Berlin Heidelberg, 2010. https://pdfs.semanticscholar.org/c322/781db8d22ea39760b798e2c9b0c4a7a1b420.pdf.

[VGE15] Roel Verdult, Flavio D. Garcia, and Baris Ege. Dismantling Megamos Crypto : Wirelessly Lockpicking a Vehicle Immobilizer. In *Proceedings of the 22th Conference on USENIX Security Symposium*, pages 703–718, Washington, D.C., 2015. USENIX Association. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/verdult.

[vWWB11] Jasper G. J. van Woudenberg, Marc F. Witteman, and Bram Bakker. Improving Differential Power Analysis by Elastic Alignment. In *Topics in Cryptology – CT-RSA 2011*, pages 104–119. Springer Berlin Heidelberg, 2011.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3) :66–75, September 1991.

[Wei09] Ralf-Philipp Weinmann. AXR - Crypto made from modular additions, XORs and word rotations. Dagstuhl Seminar 09031, 2009.

[WH17a] Mathias Wagner and Stefan Heyse. Single-Trace Template Attack on the DES Round Keys of a Recent Smart Card. Cryptology ePrint Archive, Report 2017/057, 2017. https://eprint.iacr.org/2017/057.

[WH17b] Yoo-Seung Won and Dong-Guk Han. Efficient Conversion Method from Arithmetic to Boolean Masking in Constrained Devices. In *Constructive Side-Channel Analysis and Secure Design*, pages 120–137. Springer International Publishing, 2017. https://eprint.iacr.org/2016/664.pdf.

[WHF03] D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). Technical report, 2003. https://tools.ietf.org/html/rfc3610.

[WN95] David J. Wheeler and Roger M. Needham. TEA, a Tiny Encryption Algorithm. In *Fast Software Encryption*. Springer Berlin Heidelberg, 1995.

[wol10] wolfSSL. The Benefit of Stream Ciphers, 2010. https://www.wolfssl.com/the-benefit-of-stream-ciphers/.

[Wu16] Hongjun Wu. ACORN : A Lightweight Authenticated Cipher (v3). Submission to the CAESAR competition, 2016. https://competitions.cr.yp.to/round3/acornv3.pdf.

[Zan01] Paul Van De Zande. The Day DES Died, July 2001. https://www.sans.org/reading-room/whitepapers/vpns/day-des-died-722.

[ZFL18] Xiaojuan Zhang, Xiutao Feng, and Dongdai Lin. Fault Attack on ACORN v3. *The Computer Journal*, 2018.

Annexe A

Codes sources

A.1 Implémentation du quart de tour de ChaCha en langage C

```
typedef unsigned int u32;
typedef unsigned char u8;
typedef union {
    u32 u[16];
    u8 c[64];
} chacha_buf;

# define ROTATE(v, n) (((v) << (n)) | ((v) >> (32 - (n))))

/* QUARTERROUND updates a, b, c, d with a ChaCha "quarter" round. */
# define QUARTERROUND(a,b,c,d) ( \
                x[a] += x[b], x[d] = ROTATE((x[d] ^ x[a]),16), \
                x[c] += x[d], x[b] = ROTATE((x[b] ^ x[c]),12), \
                x[a] += x[b], x[d] = ROTATE((x[d] ^ x[a]), 8), \
                x[c] += x[d], x[b] = ROTATE((x[b] ^ x[c]), 7)  )

/* chacha_core performs 20 rounds of ChaCha on the input words in
 * |input| and writes the 64 output bytes to |output|. */
static void chacha20_core(chacha_buf *output, const u32 input[16])
{
    u32 x[16];
    int i;
    const union {
        long one;
        char little;
    } is_endian = { 1 };

    memcpy(x, input, sizeof(x));

    for (i = 20; i > 0; i -= 2) {
        QUARTERROUND(0, 4, 8, 12);
        QUARTERROUND(1, 5, 9, 13);
        QUARTERROUND(2, 6, 10, 14);
        QUARTERROUND(3, 7, 11, 15);
        QUARTERROUND(0, 5, 10, 15);
        QUARTERROUND(1, 6, 11, 12);
        QUARTERROUND(2, 7, 8, 13);
        QUARTERROUND(3, 4, 9, 14);
    }

  for (i = 0; i < 16; ++i)
    output->u[i] = x[i] + input[i];
}
```

Code extrait de la librairie OpenSSL (version 1.1.0f)

A.2 Instructions ARM pour le quart de tour de ChaCha

```
LDR   r1,[sp,#0x10]
LDR   r0,[sp,#0x00]
ADD   r0,r0,r1
STR   r0,[sp,#0x00]
LDR   r1,[sp,#0x00]
LDR   r0,[sp,#0x30]
EORS  r0,r0,r1
LSLS  r1,r0,#16
LDR   r2,[sp,#0x00]
LDR   r0,[sp,#0x30]
EORS  r0,r0,r2
ORR   r0,r1,r0,LSR #16
STR   r0,[sp,#0x30]
LDR   r1,[sp,#0x30]
LDR   r0,[sp,#0x20]
ADD   r0,r0,r1
STR   r0,[sp,#0x20]
LDR   r1,[sp,#0x20]
LDR   r0,[sp,#0x10]
EORS  r0,r0,r1
LSLS  r1,r0,#12
LDR   r2,[sp,#0x20]
LDR   r0,[sp,#0x10]
EORS  r0,r0,r2
ORR   r0,r1,r0,LSR #20
STR   r0,[sp,#0x10]
LDR   r1,[sp,#0x10]
LDR   r0,[sp,#0x00]
ADD   r0,r0,r1
STR   r0,[sp,#0x00]
LDR   r1,[sp,#0x00]
LDR   r0,[sp,#0x30]
EORS  r0,r0,r1
LSLS  r1,r0,#8
LDR   r2,[sp,#0x00]
LDR   r0,[sp,#0x30]
EORS  r0,r0,r2
ORR   r0,r1,r0,LSR #24
STR   r0,[sp,#0x30]
LDR   r1,[sp,#0x30]
LDR   r0,[sp,#0x20]
ADD   r0,r0,r1
STR   r0,[sp,#0x20]
LDR   r1,[sp,#0x20]
LDR   r0,[sp,#0x10]
EORS  r0,r0,r1
LSLS  r1,r0,#7
LDR   r2,[sp,#0x20]
LDR   r0,[sp,#0x10]
EORS  r0,r0,r2
ORR   r0,r1,r0,LSR #25
STR   r0,[sp,#0x10]
```

Compilation de A.1 en -O0

```
LDR   r1,[sp,#0x10]
LDR   r0,[sp,#0x00]
ADD   r0,r0,r1
STR   r0,[sp,#0x00]
LDR   r1,[sp,#0x00]
LDR   r0,[sp,#0x30]
EORS  r0,r0,r1
ROR   r0,r0,#16
STR   r0,[sp,#0x30]
LDR   r1,[sp,#0x30]
LDR   r0,[sp,#0x20]
ADD   r0,r0,r1
STR   r0,[sp,#0x20]
LDR   r1,[sp,#0x20]
LDR   r0,[sp,#0x10]
EORS  r0,r0,r1
ROR   r0,r0,#20
STR   r0,[sp,#0x10]
LDR   r1,[sp,#0x10]
LDR   r0,[sp,#0x00]
ADD   r0,r0,r1
STR   r0,[sp,#0x00]
LDR   r1,[sp,#0x00]
LDR   r0,[sp,#0x30]
EORS  r0,r0,r1
ROR   r0,r0,#24
STR   r0,[sp,#0x30]
LDR   r1,[sp,#0x30]
LDR   r0,[sp,#0x20]
ADD   r0,r0,r1
STR   r0,[sp,#0x20]
LDR   r1,[sp,#0x20]
LDR   r0,[sp,#0x10]
EORS  r0,r0,r1
ROR   r0,r0,#25
STR   r0,[sp,#0x10]
```

Compilation de A.1 en -O3

```
PUSH  {r4-r7}
LDR   r4, [r0]
LDR   r5, [r1]
LDR   r6, [r2]
LDR   r7, [r3]
ADD   r4, r4, r5
EOR   r7, r7, r4
ROR   r7, r7, #16
ADD   r6, r6, r7
EOR   r5, r5, r6
ROR   r5, r5, #20
ADD   r4, r4, r5
EOR   r7, r7, r4
ROR   r7, r7, #24
ADD   r6, r6, r7
EOR   r5, r5, r6
ROR   r5, r5, #25
STR   r4, [r0]
STR   r5, [r1]
STR   r6, [r2]
STR   r7, [r3]
POP   {r4-r7}
```

Implémentation assembleur

A.3 Implémentation optimisée d'ACORN publiée dans le cadre de la compétition CAESAR

```
/*
   This is the optimized implementation of ACORN-128.

   In the implementation, we store the 293-bit register into 7 64-bit
     registers:
   293-bit register R:  r292 r291 r290 r289 r288 r287 r286 r285 ..... r5
     r4 r3 r2 r1 r0
```

```
6
7     state[0]:  r60  r59  r58  r57  ..... r2   r1   r0     (61 bits) (lsb:
        r0)
8     state[1]:  r106 r105 r104 r103 ..... r63  r62  r61    (46 bits) (lsb:
        r61)
9     state[2]:  r153 r152 r151 r150 ..... r109 r108 r107   (47 bits) (lsb:
        r107)
10    state[3]:  r192 r191 r190 r189 ..... r156 r155 r154   (39 bits) (lsb:
        r154)
11    state[4]:  r229 r228 r227 r226 ..... r195 r194 r193   (37 bits) (lsb:
        r193)
12    state[5]:  r288 r287 r286 r285 ..... r232 r231 r230   (59 bits) (lsb:
        r230)
13    state[6]:  r292 r291 r290 r289                        (4  bits) (lsb:
        r289)
14  */
15
16  #include <string.h>
17  #include <stdio.h>
18  #include "crypto_aead.h"
19
20  #define maj(x,y,z)    ( ((x) & (y)) ^ ((x) & (z)) ^ ((y) & (z)) )
21  #define ch(x,y,z)     ( ((x) & (y)) ^ ((~x) & (z)) )
22
23  //encrypt 32 bit
24  void encrypt_32bits(unsigned long long *state, unsigned int plaintextword,
        unsigned int *ciphertextword, unsigned int ca, unsigned int cb)
25  {
26    unsigned int f,ks;
27    unsigned long long word_244, word_23, word_160, word_111, word_66,
        word_196, word_0, word_107, word_230;
28    unsigned long long word_12,word_154,word_235,word_61,word_193;
29
30    word_235 = state[5] >> 5;
31    word_196 = state[4] >> 3;
32    word_160 = state[3] >> 6;
33    word_111 = state[2] >> 4;
34    word_66  = state[1] >> 5;
35    word_23  = state[0] >> 23;
36    word_244 = state[5] >> 14;
37    word_12  = state[0] >> 12;
38
39    //update using those 6 LFSRs
40    state[6] ^= (state[5] ^ word_235) & 0xffffffff;
41    state[5] ^= (state[4] ^ word_196) & 0xffffffff;
42    state[4] ^= (state[3] ^ word_160) & 0xffffffff;
43    state[3] ^= (state[2] ^ word_111) & 0xffffffff;
44    state[2] ^= (state[1] ^ word_66)  & 0xffffffff;
45    state[1] ^= (state[0] ^ word_23)  & 0xffffffff;
46
47    //compute keystream
48    ks = word_12 ^ state[3] ^ maj(word_235, state[1], state[4]) ^ ch(state
        [5], word_111, word_66);
49
50    f = state[0] ^ (~state[2]) ^ maj(word_244, word_23, word_160) ^ (
        word_196 & ca) ^ (cb & ks);
51    *ciphertextword = plaintextword ^ ks;
52    f = f ^ plaintextword;
53    state[6] = state[6] ^ ( (unsigned long long)f << 4 );
54
55    //shift by 32 bits
56    state[0] = (state[0] >> 32) | ((state[1] & 0xffffffff) << 29);
57    state[1] = (state[1] >> 32) | ((state[2] & 0xffffffff) << 14);
58    state[2] = (state[2] >> 32) | ((state[3] & 0xffffffff) << 15);
59    state[3] = (state[3] >> 32) | ((state[4] & 0xffffffff) << 7);
60    state[4] = (state[4] >> 32) | ((state[5] & 0xffffffff) << 5);
61    state[5] = (state[5] >> 32) | ((state[6] & 0xffffffff) << 27);
62    state[6] =  state[6] >> 32;
63
64    return;
65  }
```

Code C de référence (version optimisée) tiré de
http://www3.ntu.edu.sg/home/wuhj/research/caesar/caesar.html

A.4 Code Assembleur ARM pour la mise à jour de l'état interne d'ACORN

```
;state[0]:  S_31...S_0    (32 bits)
;state[1]:  S_60...S_32   (29 bits)
;state[2]:  S_92...S_61   (32 bits)
;state[3]:  S_105...S_93  (14 bits)
;state[4]:  S_138...S_107 (32 bits)
;state[5]:  S_153...S_139 (15 bits)
;state[6]:  S_185...S_154 (32 bits)
;state[7]:  S_192...S_186 (7  bits)
;state[8]:  S_224...S_193 (32 bits)
;state[9]:  S_229...S_225 (5  bits)
;state[10]: S_261...S_230 (32 bits)
;state[11]: S_288...S_262 (27 bits)
;state[12]: S_292...S_289 (4  bits)
acorn_asm
  ; r0 points to the internal state
  PUSH  {r4-r11, lr}
  SUB   sp, sp, #0x1c
  ; --- computes temp data ---
  ; computes S_275...S_244
  LDRD  r4, r5, [r0,#0x28]
  LSR   r4, r4, #14
  ORR   r4, r4, r5, LSL #18
  STR   r4, [sp, #0x18]
  ; computes S_266...S_235
  LDR   r4, [r0, #0x28]
  LSR   r4, r4, #5
  ORR   r4, r4, r5, LSL #27
  STR   r4, [sp, #0x14]
  ; computes S_227...S_196
  LDRD  r4, r5, [r0, #0x20]
  LSR   r4, r4, #3
  ORR   r4, r4, r5, LSL #29
  STR   r4, [sp, #0x10]
  ; computes S_191...S_160
  LDRD  r4, r5, [r0, #0x18]
  LSR   r4, r4, #6
  ORR   r4, r4, r5, LSL #26
  STR   r4, [sp, #0x0c]
  ; computes S_142...S_111
  LDRD  r4, r5, [r0, #0x10]
  LSR   r4, r4, #4
  ORR   r4, r4, r5, LSL #28
  STR   r4, [sp, #0x08]
  ; computes S_97...S_66
  LDRD  r4, r5, [r0, #0x08]
  LSR   r4, r4, #5
  ORR   r4, r4, r5, LSL #27
  STR   r4, [sp, #0x04]
  ; computes S_54...S_23
  LDRD  r4, r5, [r0]
  LSR   r4, r4, #23
  ORR   r4, r4, r5, LSL #9
  STR   r4, [sp]

  ; --- updates LFSRs ---
  ; updates LFSR 5
  LDR   r4, [r0, #0x28]
  LDR   r5, [r0, #0x30]
  LDR   r6, [sp, #0x14]
  EOR   r6, r4, r6
  EOR   r5, r5, r6
  STR   r5, [r0, #0x30]
  ; updates LFSR 4
  LDR   r5, [r0, #0x20]
  LDR   r6, [sp, #0x10]
  EOR   r6, r5, r6
  EOR   r4, r4, r6
  STR   r4, [r0, #0x28]
  ; updates LFSR 3
```

```
70    LDR    r4, [r0, #0x18]
71    LDR    r6, [sp, #0x0c]
72    EOR    r6, r4, r6
73    EOR    r5, r5, r6
74    STR    r5, [r0, #0x20]
75    ; updates LFSR 2
76    LDR    r5, [r0, #0x10]
77    LDR    r6, [sp, #0x08]
78    EOR    r6, r5, r6
79    EOR    r4, r4, r6
80    STR    r4, [r0, #0x18]
81    ; updates LFSR 1
82    LDR    r4, [r0, #0x08]
83    LDR    r6, [sp, #0x04]
84    EOR    r6, r4, r6
85    EOR    r5, r5, r6
86    STR    r5, [r0, #0x10]
87    ; updates LFSR 0
88    LDR    r5, [r0]
89    LDR    r6, [sp]
90    EOR    r6, r5, r6
91    EOR    r4, r4, r6
92    STR    r4, [r0, #0x08]
93
94    ; --- generates the keystream ---
95    ; computes maj(W_235, W_61, W_193)
96    LDR    r5, [r0, #0x20]
97    LDR    r6, [sp, #0x14]
98    AND    r7, r4, r5
99    AND    r4, r4, r6
100   AND    r5, r5, r6
101   EOR    r4, r4, r5
102   EOR    r4, r4, r7
103   ; computes ch(W_230, W_111, W_66)
104   LDRD   r5, r6, [sp, #0x04]
105   LDR    r7, [r0, #0x28]
106   AND    r6, r6, r7
107   MVN    r7, r7
108   AND    r7, r5, r7
109   EOR    r6, r6, r7
110   ; finalizes keystream computation
111   EOR    r4, r4, r6
112   LDR    r5, [r0, #0x18] ; W_154
113   EOR    r4, r4, r5
114   ; computes S_43...S_12
115   LDRD   r5, r6, [r0]
116   LSR    r5, r5, #12
117   ORR    r5, r5, r6, LSL #20
118   EOR    r4, r4, r5
119
120   ; --- computes the ciphertext ---
121   EOR    r5, r1, r4
122   STR    r5, [r2]
123
124   ; --- generates the feedback word ---
125   ; computes (c_a & W_196) ^ (c_b & ks)
126   LDRD   r5, r6, [r3]
127   AND    r4, r4, r6
128   LDR    r6, [sp, #0x10]
129   AND    r5, r5, r6
130   EOR    r4, r4, r5
131   EOR    r1, r1, r4
132   ; computes maj(W_244, W_23, W_160)
133   LDR    r4, [sp, #0x18]
134   LDR    r5, [sp]
135   LDR    r6, [sp, #0x0c]
136   AND    r7, r4, r5
137   AND    r4, r4, r6
138   AND    r5, r5, r6
139   EOR    r4, r4, r5
140   EOR    r4, r4, r7
141   EOR    r1, r1, r4
142   ; computes state[0] ^ (~state[2])
```

```
143     LDR    r4, [r0, #0x10]
144     MVN    r4, r4
145     EOR    r1, r1, r4
146     LDR    r4, [r0]
147     EOR    r1, r1, r4
148
149     ; --- state[6] ^= feedback << 4 ---
150     LDR    r4, [r0, #0x30]
151     EOR    r4, r4, r1, LSL #4
152     STR    r4, [r0, #0x30]
153
154     ; --- shifts all LFSRs by 32 bits ---
155     ; 1st word
156     LDRD   r4, r5, [r0, #0x04]
157     ORR    r4, r4, r5, LSL #29
158     STR    r4, [r0]
159     ; 2nd word
160     LSR    r5, r5, #3
161     STR    r5, [r0, #0x04]
162     ; 3rd word
163     LDRD   r4, r5, [r0, #0x0c]
164     ORR    r4, r4, r5, LSL #14
165     STR    r4, [r0, #0x08]
166     ; 4th word
167     LSR    r5, r5, #18
168     STR    r5, [r0, #0x0c]
169     ; 5th word
170     LDRD   r4, r5, [r0, #0x14]
171     ORR    r4, r4, r5, LSL #15
172     STR    r4, [r0, #0x10]
173     ; 6th word
174     LSR    r5, r5, #17
175     STR    r5, [r0, #0x14]
176     ; 7th word
177     LDRD   r4, r5, [r0, #0x1c]
178     ORR    r4, r4, r5, LSL #7
179     STR    r4, [r0, #0x18]
180     ; 8th word
181     LSR    r5, r5, #25
182     STR    r5, [r0, #0x1c]
183     ; 9th word
184     LDRD   r4, r5, [r0, #0x24]
185     ORR    r4, r4, r5, LSL #5
186     STR    r4, [r0, #0x20]
187     ; 10th word
188     LSR    r5, r5, #27
189     STR    r5, [r0, #0x24]
190     ; 11th word
191     LDRD   r4, r5, [r0, #0x2c]
192     ORR    r4, r4, r5, LSL #27
193     STR    r4, [r0, #0x28]
194     ; 12th word
195     LSR    r5, r5, #5
196     STR    r5, [r0, #0x2c]
197     ; 13th word
198     LSR    r4, r1, #28
199     STR    r4, [r0, #0x30]
200
201     ADD    sp, sp, #0x1c
202     POP    {r4-r11, lr}
203     BX     lr
```

Code Assembleur pour la fonction d'itération d'ACORN

A.5 Code assembleur ARM pour la transformation p utilisée dans ASCON

```
permutation_ascon
  ; r0 points to the internal state
  PUSH  {r2-r11, lr}
  SUB   sp, sp, #0x08

  ; for each round
loop
  ; --- loads the entire state ---
  LDRD  r2, r3, [r0]
  LDRD  r4, r5, [r0, #0x08]
  LDRD  r6, r7, [r0, #0x10]
  LDRD  r8, r9, [r0, #0x18]
  LDRD  r10, r11, [r0, #0x20]

  ; --- adds round constant ---
  EOR   r7, r7, r1

  ; --- applies the nonlinear layer ---
  EOR   r2, r2, r10 ; x0 ^= x4
  EOR   r3, r3, r11 ; x0 ^= x4
  STRD  r2, r3, [r0]
  EOR   r10, r10, r8  ; x4 ^= x3
  EOR   r11, r11, r9  ; x4 ^= x3
  STRD  r10, r11, [r0, #0x20]
  EOR   r6, r6, r4  ; x2 ^= x1
  EOR   r7, r7, r5  ; x2 ^= x1
  STRD  r6, r7, [r0, #0x10]
  MVN   r2, r2  ; ~x0
  MVN   r3, r3  ; ~x0
  AND   r2, r2, r4  ; ~x0 & x1
  AND   r3, r3, r5  ; ~x0 & x1
  STRD  r2, r3, [sp]
  MVN   r4, r4  ; ~x1
  MVN   r5, r5  ; ~x1
  AND   r4, r4, r6  ; ~x1 & x2
  AND   r5, r5, r7  ; ~x1 & x2
  MVN   r6, r6  ; ~x2
  MVN   r7, r7  ; ~x2
  AND   r6, r6, r8  ; ~x2 & x3
  AND   r7, r7, r9  ; ~x2 & x3
  MVN   r8, r8  ; ~x3
  MVN   r9, r9  ; ~x3
  AND   r8, r8, r10 ; ~x3 & x4
  AND   r9, r9, r11 ; ~x3 & x4
  MVN   r10, r10  ; ~x4
  MVN   r11, r11  ; ~x4
  LDRD  r2, r3, [r0]
  AND   r10, r10, r2  ; ~x4 & x0
  AND   r11, r11, r3  ; ~x4 & x0
  EOR   r2, r2, r4  ; x0 ^= ~x1 & x2
  EOR   r3, r3, r5  ; x0 ^= ~x1 & x2
  STRD  r2, r3, [r0]
  LDRD  r2, r3, [r0, #0x08]
  EOR   r2, r2, r6  ; x1 ^= ~x2 & x3
  EOR   r3, r3, r7  ; x1 ^= ~x2 & x3
  STRD  r2, r3, [r0, #0x08]
  LDRD  r2, r3, [r0, #0x10]
  EOR   r2, r2, r8  ; x2 ^= ~x3 & x4
  EOR   r3, r3, r9  ; x2 ^= ~x3 & x4
  STRD  r2, r3, [r0, #0x10]
  LDRD  r2, r3, [r0, #0x18]
  EOR   r2, r2, r10 ; x3 ^= ~x4 & x0
  EOR   r3, r3, r11 ; x3 ^= ~x4 & x0
  STRD  r2, r3, [r0, #0x18]
  LDRD  r10, r11, [r0, #0x20]
  LDRD  r2, r3, [sp]
  EOR   r10, r10, r2  ; x4 ^= ~x0 & x1
  EOR   r11, r11, r3  ; x4 ^= ~x0 & x1
  LDRD  r2, r3, [r0]
```

```
70    LDRD  r4, r5, [r0, #0x08]
71    LDRD  r6, r7, [r0, #0x10]
72    LDRD  r8, r9, [r0, #0x18]
73    EOR   r4, r4, r2  ; x1 ^= x0
74    EOR   r5, r5, r3  ; x1 ^= x0
75    EOR   r2, r2, r10 ; x0 ^= x4
76    EOR   r3, r3, r11 ; x0 ^= x4
77    EOR   r8, r8, r6  ; x3 ^= x2
78    EOR   r9, r9, r7  ; x3 ^= x2
79    MVN   r6, r6    ; ~x2
80    MVN   r7, r7    ; ~x2
81    STRD  r8, r9, [r0, #0x18]
82    STRD  r10, r11, [r0, #0x20]
83    ; --- applies the linear layer ---
84    ; linear diffusion on x0
85    LSR   r10, r2, #19
86    ORR   r10, r10, r3, LSL #13
87    LSR   r11, r3, #19
88    ORR   r11, r11, r2, LSL #13
89    LSR   r8, r2, #28
90    ORR   r8, r8, r3, LSL #4
91    LSR   r9, r3, #28
92    ORR   r9, r9, r2, LSL #4
93    EOR   r2, r2, r10
94    EOR   r3, r3, r11
95    EOR   r2, r2, r8
96    EOR   r3, r3, r9
97    STRD  r2, r3, [r0]
98    ; linear diffusion on x1
99    LSR   r10, r5, #29
100   ORR   r10, r10, r4, LSL #3
101   LSR   r11, r4, #29
102   ORR   r11, r11, r5, LSL #3
103   LSR   r8, r5, #7
104   ORR   r8, r8, r4, LSL #25
105   LSR   r9, r4, #7
106   ORR   r9, r9, r5, LSL #25
107   EOR   r4, r4, r10
108   EOR   r5, r5, r11
109   EOR   r4, r4, r8
110   EOR   r5, r5, r9
111   STRD  r4, r5, [r0, #0x08]
112   ; linear diffusion on x2
113   LSR   r10, r6, #1
114   ORR   r10, r10, r7, LSL #31
115   LSR   r11, r7, #1
116   ORR   r11, r11, r6, LSL #31
117   LSR   r8, r6, #6
118   ORR   r8, r8, r7, LSL #26
119   LSR   r9, r7, #6
120   ORR   r9, r9, r6, LSL #26
121   EOR   r6, r6, r10
122   EOR   r7, r7, r11
123   EOR   r6, r6, r8
124   EOR   r7, r7, r9
125   STRD  r6, r7, [r0, #0x10]
126   ; linear diffusion on x3
127   LDRD  r6, r7, [r0, #0x18]
128   LSR   r10, r6, #10
129   ORR   r10, r10, r7, LSL #22
130   LSR   r11, r7, #10
131   ORR   r11, r11, r6, LSL #22
132   LSR   r8, r6, #17
133   ORR   r8, r8, r7, LSL #15
134   LSR   r9, r7, #17
135   ORR   r9, r9, r6, LSL #15
136   EOR   r6, r6, r10
137   EOR   r7, r7, r11
138   EOR   r6, r6, r8
139   EOR   r7, r7, r9
140   STRD  r6, r7, [r0, #0x18]
141   ; linear diffusion on x4
142   LDRD  r6, r7, [r0, #0x20]
```

```
143    LSR    r10, r6, #7
144    ORR    r10, r10, r7, LSL #25
145    LSR    r11, r7, #7
146    ORR    r11, r11, r6, LSL #25
147    LSR    r8, r7, #9
148    ORR    r8, r8, r6, LSL #23
149    LSR    r9, r6, #9
150    ORR    r9, r9, r7, LSL #23
151    EOR    r6, r6, r10
152    EOR    r7, r7, r11
153    EOR    r6, r6, r8
154    EOR    r7, r7, r9
155    STRD   r6, r7, [r0, #0x20]
156
157    ; --- loop iteration check ---
158    SUB    r1, r1, #0x0f
159    CMP    r1, #0x4b
160    BGE    loop
161
162    ADD    sp, sp, #0x08
163    POP    {r2-r11, lr}
164    BX     lr
```

Code Assembleur pour la permutation d'ASCON

Annexe B

Preuves mathématiques

B.1 Preuve de la proposition 1

Démonstration. Pour plus de lisibilité, notons l'espérance mathématique $\mathsf{E}(X) = \frac{1}{n}\sum_{i=1}^{n} x_i$ et l'écart type $\sigma_X = \sqrt{\mathsf{E}\left((X-\mathsf{E}(X))^2\right)}$.

$$
\begin{aligned}
\mathsf{Corr}(X,\ Y_i) &= \frac{\mathsf{Cov}(X,\ Y_i)}{\sigma_X\ \sigma_{Y_i}} \\
&= \frac{\mathsf{E}(X\ Y_i) - \mathsf{E}(X)\ \mathsf{E}(Y_i)}{\sigma_X\ \sqrt{\mathsf{E}\Big(\big(Y_i - \mathsf{E}(Y_i)\big)^2\Big)}} \\
&= \frac{\mathsf{E}\big(X\ (z-Y_j)\big) - \mathsf{E}(X)\ \mathsf{E}(z-Y_j)}{\sigma_X\ \sqrt{\mathsf{E}\Big(\big(z-Y_j-\mathsf{E}(z-Y_j)\big)^2\Big)}} \\
&= \frac{-\mathsf{E}\big(X\ Y_j\big) + \mathsf{E}(X)\ \mathsf{E}(Y_j)}{\sigma_X\ \sqrt{\mathsf{E}\Big(\big(-Y_j+\mathsf{E}(Y_j)\big)^2\Big)}} \\
&= \frac{-\big(\mathsf{E}\big(X\ Y_j\big) - \mathsf{E}(X)\ \mathsf{E}(Y_j)\big)}{\sigma_X\ \sqrt{\mathsf{E}\Big(\big(Y_j-\mathsf{E}(Y_j)\big)^2\Big)}} \\
&= \frac{-\mathsf{Cov}(X,\ Y_i)}{\sigma_X\ \sigma_{Y_i}} \\
&= -\mathsf{Corr}(X,\ Y_j)\,.
\end{aligned}
$$

□

B.2 Preuve de la proposition 2

Démonstration. Nous prouvons d'abord le cas spécifique où l'hypothèse $k = \hat{\mathrm{k}}_i^X \parallel \mathrm{k}_{i+4}^Y \parallel \mathrm{k}_i^Y \parallel \hat{\mathrm{k}}_i^Z$ est telle que $\varphi_{2,n}^{\mathsf{ChaCha}}(x,k) = x$ pour tout x et argumentons ensuite sur le fait que cela définisse une borne supérieure. Pour plus de clareté, nous utilisons $\varphi_{2,n}(x,a,b,c,d)$ pour faire référence à $\varphi_{2,n}^{\mathsf{ChaCha}}(x,k)$. Ainsi, nous recherchons le nombre de quadruplets (a,b,c,d) tels que $\varphi_{2,n}(x,a,b,c,d)$ définit la fonction identité.

Soit $\varphi_{2,n}(\cdot,a,b,c,d)$ la fonction qui associe $x \in \mathbb{Z}_{2^n}$ à $\varphi_{2,n}(x,a,b,c,d)$. D'après la définition de $\varphi_{2,n}$, il est trivial que si $\varphi_{2,n}(\cdot,a,b,c,d)$ est la fonction identité sur $\mathbb{Z}_{2^n}$, alors $\varphi_{2,m}(\cdot,a,b,c,d)$ est la fonction identité sur $\mathbb{Z}_{2^m}$ pour $m < n$.

Si b et d sont tous deux des multiples de 2^m pour $m < n$, alors les additions n'impactent pas les m bits de poids faibles. Par conséquent, il est possible de découper les opérandes en m bits de poids faibles et $n-m$ bits de poids forts comme suit $x = \overline{x} \cdot 2^m + \underline{x}$ afin de calculer indépendamment $\varphi_{2,n-m}(\overline{x}, \overline{a}, \overline{b}, \overline{c}, \overline{d})$ sur $\mathbb{Z}_{2^{n-m}}$ et $\varphi_{2,m}(\underline{x}, \underline{a}, 0, \underline{c}, 0)$ sur $\mathbb{Z}_{2^m}$. On remarque que $\varphi_{2,m}(\underline{x}, \underline{a}, 0, \underline{c}, 0)$ définit la fonction identité sur $\mathbb{Z}_{2^m}$ si et seulement si $\underline{a} = \underline{c}$. Ainsi, il reste à étudier le cas où b et d sont impairs pour établir les conditions nécessaires concernant $\varphi_{2,n-m}$.

Une classe évidente des fonctions identités sur $\mathbb{Z}_{2^n}$ est définie par $\varphi_{2,n}(\cdot, 0, b, 0, d)$ avec b et d tels que $b \boxplus d = 0$. À première vue, on pourrait penser que fixer $(a, c) \neq (0, 0)$ casserait la propriété de fonction identité à cause du fait que le XOR agit de manière non-linéaire sur $\mathbb{Z}_{2^n}$. Il y a cependant deux cas particuliers. Premièrement, $c \oplus 2^{n-1} = c \boxplus 2^{n-1}$ sur $\mathbb{Z}_{2^n}$. Par conséquent, $\varphi_{2,n}(\cdot, a, b, c, d)$ définit la fonction identité sur $\mathbb{Z}_{2^n}$ à conditon que $a \boxplus b \boxplus c \boxplus d = 0$ avec $a, c \in \{0, 2^{n-1}\}$. Deuxièmement, $x \oplus 2^n - 1 = 2^n - 1 - x$ sur $\mathbb{Z}_{2^n}$. Ainsi, $\varphi_{2,n}(\cdot, 2^n - 1, b, 2^n - 1, d)$ définit la fonction identité si $b = d$. En combinant ces deux observations, on obtient que $\varphi_{2,n}(\cdot, 2^n - 1, b, 2^n - 1, d)$ définit la fonction identité si $a, c \in \{2^{n-1} - 1, 2^n - 1\}$ et $a \boxplus b = c \boxplus d$. En somme, pour b et d impairs on peut choisir a, c de 8 différentes manières, soit $a, c \in \{0, 2^{n-1}\}$ soit $a, c \in \{2^{n-1} - 1, 2^n - 1\}$, fixer d arbitrairement et résoudre l'équation adéquate pour trouver b telle que $\varphi_{2,n}(\cdot, a, b, c, d)$ définit la fonction identité. Puisqu'il y a respectivement 2^{n-1} et 2^3 façons de choisir d et (a, c), on aboutit à 2^{n+2} quadruplets (a, b, c, d) qui définissent $\varphi_{2,n}(\cdot, a, b, c, d)$ comme la fonction identité.

Dans le cas où b et d ne sont pas tous deux impairs, on utilise la méthode décrite ci-dessus qui consiste à diviser chaque mot deux parties respectivement de $n-m$ et m bits. Pour $\varphi_{2,n-m}$ on a donc 2^{n-m+2} quadruplets qui définissent la fonction identité. Pour $\varphi_{2,m}$ on a $\underline{b} = \underline{d} = 0$ et $\underline{a} = \underline{c}$ soient 2^m quadruplets qui définissent la fonction identité pour $\varphi_{2,m}$, et par conséquent $2^{n-m+2} \cdot 2^m = 2^{n+2}$ pour $\varphi_{2,n}$.

Ainsi, en sommant pour toutes les valeurs de $m = 0 \cdots n-1$ (où $m = 0$ définit le cas où b et d sont impairs) on obtient au total $n \cdot 2^{n+2}$ quadruplets $(a, b, c, d) \in \mathbb{Z}_{2^n}$ tels que $\varphi_{2,n}(\cdot, a, b, c, d)$ définit la fonction identité.

L'avantage de traiter le cas de la fonction identité est que l'on cherche les opérandes pour lesquelles les additions et les XOR s'annulent. Dans les autres cas, l'analyse est plus complexe et dépend directement de la valeur des bits des opérandes. Par exemple, une attaque avec $\varphi_{2,3}$ et $k = (000 \parallel 001 \parallel 001 \parallel 000)_2$ retourne seulement 36 collisions. D'autre part, la simulation présentée en figure 5.12 retourne $32 = 2 \cdot 2^{2+2}$ collisions alors que $k = (00 \parallel 01 \parallel 11 \parallel 11)_2$ ne définit pas la fonction identité. Dans ce cas, la démarche de la preuve est similaire mais au lieu de chercher à annuler les additions et les XOR, il faut chercher un résultat autre que 0.

Dans le cas où $n > 4$, le nombre de collisions est inférieur à $n \cdot 2^{n+2}$ puisque plusieurs bits hypothétiques se superposent. □

Annexe C

Liste des acronymes

AES *Advanced Encryption Standard*. 5

ARX *Addition Rotation XOR*. 42

BLE *Bluetooth Low Energy*. 27

CAESAR *Competition for Authenticated Encryption: Security, Applicability, and Robustness*. 15

CCM *Counter with CBC-MAC*. 31

CNF *Conjunctive Normal Form*. 98

CPA *Correlation Power Analysis*. 58

CPU *Central Processing Unit*. 9

DES *Data Encryption Standard*. 5

DFA *Differential Fault Analysis*. 13

DPA *Differential Power Analysis*. 57

DRAM *Dynamic Random Access Memory*. 13

ECC *Elliptic Curve Cryptography*. 7

ECDH *Elliptic Curve Diffie Hellman*. 30

ECRYPT *European Network of Excellence in Cryptology*. 40

FPGA *Field-Programmable Gate Array*. 94

GCM *Galois Counter Mode*. 40

GE *Gate Equivalent*. 38

HMAC *keyed-Hash Message Authentication Code*. 6

HTTPS *HyperText Transfer Protocol Secure*. 22

IdO *Internet des Objets*. 2

IV *Initial Vector*. 74

LFSR *Linear Feedback Shit Register*. 49

LPWAN *Low-Power Wide-Area Networks*. 3

MAC *Message Authentication Code*. 6

MITM *Man In The Middle*. 21

NFSR *Nonlinear Feedback Shit Register*. 49
NIST *National Institute Standard of Technology*. 6
NMOS *N-type Metal-Oxide-Semiconductor*. 11
NSA *National Security Agency*. 41

OTA *Over The Air*. 23

PMOS *P-type Metal-Oxide-Semiconductor*. 11

RAM *Random Access Memory*. 52
RFID *Radio Frequency IDentification*. 20
RGPD *Règlement Général sur la Protection des Données*. 4
RSA *Rivest-Shamir-Adleman*. 7

SIMD *Single Instruction Multiple Data*. 44
SPA *Simple Power Analysis*. 54

TLS *Transport Layer Security*. 22

UAL *Unité Arithmétique et Logique*. 42

XOR *eXclusive OR*. 24

Annexe D

Liste des symboles

$\oplus$ OU exclusif bit à bit (`XOR`). 36

$\vee$ OU bit à bit (`OR`). 62

$\wedge$ ET bit à bit (`AND`). 50

$\neg$ Négation logique (`NOT`). 50

$\|$ Concaténation. 27

$\boxplus$ Addition modulo 2^n. 45

$\boxminus$ Soustraction modulo 2^n. 76

$\lll$ Rotation bit à bit vers la gauche. 75

$\ggg$ Rotation bit à bit vers la droite. 82

φ Fonction de sélection. 55

ÉCOLE NATIONALE SUPÉRIEURE DES
MINES DE SAINT-ÉTIENNE

NNT : 2019LYSEM021
Alexandre ADOMNICAI

CRYPTOGRAPHIE POUR L'INTERNET DES OBJETS : IMPLÉMENTATIONS ET INTÉGRATIONS SÉCURISÉES

Spécialité : Microélectronique

Mots clefs : Cryptographie légère, Internet des objets, Attaques physiques, ACORN, Ascon, ChaCha

Résumé : Si l'internet des objets laisse entrevoir de nouvelles perspectives, tant pour les entreprises que pour les administrations et les citoyens, son déploiement représente un défi majeur en termes de sécurité et de respect de la vie privée. Afin d'assurer la confidentialité des données générées par ces objets connectés, la plupart des protocoles de l'internet des objets intègrent des primitives cryptographiques au sein même de leur spécification. Bien que les algorithmes cryptographiques employés à ce jour bénéficient d'une sécurité éprouvée, ils sont directement dérivés des protocoles de sécurité traditionnels et par conséquent, n'ont pas été conçus pour être particulièrement efficaces sur des plateformes à faibles ressources telles que celles dédiées aux objets connectés. Cette thèse se concentre sur les primitives cryptographiques dites "légères" spécialement conçues pour l'internet des objets.

Outre les principaux objectifs des algorithmes cryptographiques légers, qui sont d'être plus efficaces et plus compacts que les algorithmes traditionnels, leur capacité à se protéger facilement des attaques physiques, qui ciblent l'implémentation d'un algorithme cryptographique plutôt que sa structure mathématique, est également un critère à forte valeur ajoutée. Bien que ces attaques nécessitent pour la plupart un accès physique au composant en charge d'exécuter la primitive cryptographique, elles constituent une réelle menace dans le modèle de l'internet des objets, où les objets connectés sont potentiellement déployés à proximité physique de tout type d'attaquant.

Nos travaux se concentrent sur l'étude de trois algorithmes de chiffrement, chacun ayant un fort potentiel d'application industrielle. En particulier, nous démontrons les besoins en matière de sécurité vis à vis des attaques physiques en introduisant deux nouvelles attaques par observation : une contre l'algorithme ChaCha20, standard IETF largement répandu, et une autre contre l'algorithme ACORN, un des gagnants de la compétition cryptographique CAESAR, qui vise à définir de nouveaux algorithmes de chiffrement pour différents cas d'usage. Ces attaques ont été vérifiées en laboratoire en exploitant les fuites électromagnétiques d'un microcontrôleur tels que ceux utilisés dans l'internet des objets. Nous proposons également des implémentations résistantes à ces attaques en appliquant à ces algorithmes des contre-mesures génériques publiées dans la littérature.

ÉCOLE NATIONALE SUPÉRIEURE DES
MINES DE SAINT-ÉTIENNE

NNT : 2019LYSEM021
Alexandre ADOMNICAI

SECURE IMPLEMENTATION AND INTEGRATION OF LIGHTWEIGHT CRYPTOGRAPHY FOR THE INTERNET OF THINGS

Speciality : Microelectronics

Keywords : Lightweight cryptography, Internet of things, Physical attacks, ACORN, Ascon, ChaCha

Abstract : While the internet of things (IoT) promises many advances for businesses, administrations and citizens, its deployment is a real challenge in terms of privacy and security. In order to ensure the confidentiality and the authenticity of information transmitted by these objects, numerous IoT protocols incorporate cryptographic algorithms within their specification. To date, these algorithms are the same as the ones used in traditionnal internet security protocols and thus, have not been designed with constrainted plateforms in mind. This thesis focuses on lightweight cryptography which aims at reduce as much as possible the cost of its implementation.

Apart from the main goal of lightweight cryptography which is to consume less ressources than traditional algorithms, it is also valuable to take into account the integration of countermeasures against physical attacks during the design phase in order to limit their impact. Although this kind of attacks require a physical access to the target, this can be a realistic scenario as connected objets might be deployed everywhere and thus, potentially accessible by malicious people.

Our works focus on the study of three lightweight cryptographic algorithms, each having a potential for industrial applications. Especially, we highlight the need of secure implementations by introducing two new side-channel attacks : one against ChaCha20, standardized by the IETF and now used in TLS 1.3, and another one against ACORN, an algorithm being part of the CAESAR portfolio. These attacks have been demonstrated in a practice by exploiting the electromagnetic emanation of a microcontroller. We also studied the integration of some generic countermeasures to these algorithms in order to protect from our attacks and report the obtained results.