Etude des compromis entre sécurité et disponibilité des systèmes embarqués communicants

Nicolas Burger

Thèse
de doctorat
de l'UTT

Nicolas BURGER

Étude des compromis entre sécurité et disponibilité des systèmes embarqués communicants

Champ disciplinaire :
Sciences pour l'Ingénieur

2020TROY0021 **Année 2020**

utt
UNIVERSITÉ DE TECHNOLOGIE
TROYES

THESE

pour l'obtention du grade de

DOCTEUR
de l'UNIVERSITE DE TECHNOLOGIE DE TROYES

EN SCIENCES POUR L'INGENIEUR

Spécialité : OPTIMISATION ET SURETE DES SYSTEMES

présentée et soutenue par

Nicolas BURGER

le 19 novembre 2020

Etude des compromis entre sécurité et disponibilité des systèmes embarqués communicants

JURY

Mme Caroline FONTAINE	DIRECTRICE DE RECHERCHE CNRS	Présidente
M. Gildas AVOINE	PROFESSEUR DES UNIVERSITES	Rapporteur
M. Frédéric BONIOL	MAITRE DE RECHERCHE - HDR	Rapporteur
M. Yves LANGERON	INGENIEUR DE RECHERCHE UTT	Examinateur
M. Benjamin NGUYEN	PROFESSEUR DES UNIVERSITES	Examinateur
M. Rémi COGRANNE	MAITRE DE CONFERENCES	Directeur de thèse
M. Patrick LALLEMENT	PROFESSEUR UTT	Directeur de thèse

Résumé

La sécurité d'un moyen de communication dépend d'algorithmes de sécurité qui reposent sur des calculs souvent lourds, en particulier au sein d'un système embarqué par rapport à ses autres tâches.

Dans un tel contexte, les algorithmes de sécurité consomment d'importantes ressources temporelles et énergétiques. La sécurité devient alors un curseur à faire varier grâce à des choix d'algorithmes et de paramètres adaptés aux besoins du système embarqué communicant.

Trouver le bon compromis entre sécurité, disponibilité et autonomie d'un réseau de capteurs sans fil est délicat pour un concepteur d'objets connectés.

L'objectif de cette thèse est de définir les paramètres sur lesquels il est possible de jouer pour trouver une solution acceptable. Un modèle sous forme de réseaux de Petri est proposé pour faire interagir ces paramètres et mettre en évidence leurs influences sur un réseau de capteurs sans fil.

Abstract (english)

The security of communications depends on security algorithms which are often based on heavy calculations, in particular within an embedded system in view of its other tasks.

In such a context, security algorithms consume significant time and energy resources. Security then becomes a slider, a slider that one can move by choosing the right algorithms and parameters for the needs of the embedded system.

Finding the right compromise between security, availability and autonomy of a wireless sensor network is difficult for an embedded systems designer.

The objective of this thesis is to define the parameters on which it is possible to act to find an acceptable solution. A model in the form of Petri nets is offered to make these parameters interact and highlight their mutual influences.

Remerciements

> *[...] Je dis merci à la vie, je lui dis merci, je chante la vie, je danse la vie... Je ne suis qu'amour ! Et finalement, quand beaucoup de gens aujourd'hui me disent : "Mais comment fais-tu pour avoir cette humanité ?" Eh bien je leur réponds très simplement. Je leur dis que c'est ce goût de l'amour, ce goût donc qui m'a poussé aujourd'hui à entreprendre une construction mécanique, mais demain, qui sait, peut-être simplement à me mettre au service de la communauté, à faire le don, le don de soi...*
>
> **— Otis**
> Astérix et Obélix : mission Cléopâtre, réalisé par Alain Chabat.

L'exercice de doctorat marque une vie et ce mémoire clôture cet exercice, donnant l'occasion de revenir sur le chemin parcouru et de remercier les personnes (physiques ou morales) qui m'ont permis d'avancer.

En effet, plusieurs années de travail pour mon doctorat à l'UTT mais également pour l'entreprise VIGILOCK et ses partenaires m'ont permis de côtoyer un grand nombre de personnes ; personnes qui m'ont accompagné dans de nouveaux domaines, qui m'ont mis à l'épreuve (volontairement ou non), qui m'ont montré les absurdités d'un système (volontairement ou non), qui m'ont permis de m'épanouir dans ma vie professionnelle et personnelle (volontairement ou non).

Comment ne pas commencer par l'UTT, que je fréquente depuis mon arrivée juste après le baccalauréat en 2010 et où j'ai pu progressivement monter en compétence. Le parcours à la carte et l'ensemble du personnel que j'ai pu fréquenter durant ces dix années (Emilie Colas, Delphine Ferry, Laure Grandhomme, Nadège Troussier,

Emmanuel Carquin, Christelle et Jean-Philippe Danau, Hélène Weber, Pauline Marty, Tatiana Reyes, Christine Champeau, Michel Doussot, Alain Ploix, ...) m'ont donné l'occasion d'évoluer dans un environnement aussi épanouissant que riche intellectuellement. Parmi ce personnel, Rémi Cogranne, Patrick Lallement et Yves Langeron m'ont directement suivi durant ce doctorat et ont fait le pari de me faire confiance dès le début, mais également dans les moments de doutes et de difficultés, tout en sachant me motiver. Pour cela, je leur suis grandement reconnaissant.

En parlant de reconnaissance justement, comment ne pourrais-je en avoir pour Stéphane Canet, gérant de VIGILOCK, qui m'a fait confiance d'abord pour un stage, puis pour ce doctorat où j'ai eu la chance de m'occuper, en plus de mon travail de recherche, de la conception et du développement de l'architecture logicielle des solutions de l'entreprise ainsi que du suivi et support client. Tout ce travail n'aurait pu être possible sans la synergie de l'équipe. C'est pourquoi je remercie également mes collègues Sullivan Savoyini, Jean-François Hénon (qui a dû partir avant que je ne finisse mon doctorat), Kaël Gustin-Hiron et l'ensemble des stagiaires et alternants que j'ai pu superviser : nos échanges ont été pour moi un véritable plaisir et source d'enseignements. Je conclurai ces remerciements pour VIGILOCK en souhaitant bon courage à Alexandre Madeline, qui prend la relève de mon poste depuis juillet 2020, et qu'il puisse tirer au moins autant d'enseignements que j'ai pu en tirer.

Je remercie Gildas Avoine et Frédéric Boniol pour leur intérêt envers mon travail de recherche en tant que rapporteurs, ainsi que Caroline Fontaine et Benjamin Nguyen pour juger mon travail en tant qu'examinateurs. Les échanges effectués lors de la soutenances ont été un plaisir et m'ont permis de discuter des futures pistes de manière concrète et pertinente.

Cette soutenance a également été l'occasion de présenter mon travail à mes proches et aux curieux grâce au collectif Les Vulgaires, et particulièrement à Matthieu Hennekine. Ce dernier s'est proposé pour vulgariser en direct la soutenance et les échanges, dans la bonne humeur et avec un *tchat* qui a été généreux en messages d'encouragements : merci à tous !

La lourdeur administrative imposée par l'UTT (exception culturelle française diront certains) a pu être allégée par la patience et l'expérience de notre trio de l'École Doctorale : Isabelle Leclerc, Pascale Denis et Thérèse Kazarian que je remercie sincèrement en leur souhaitant un jour d'avoir un système d'information digne de ce nom pour leur simplifier la vie. Merci également à Florian Bratec pour avoir rendu les derniers mois de rédaction du mémoire supportables et motivants. J'accepte d'avance son invitation à son pot de thèse et lui souhaite bon courage ainsi qu'à son associé, Guillaume, pour la suite de leurs aventures avec Altermaker.

Il est toujours difficile de remercier les personnes qui se sont acharnées à travers leurs mots, leurs actions et leurs mesquineries à me montrer l'exemple à ne pas suivre. Nombreux qu'ils sont à l'UTT, dans les milieux associatifs et politiques à Troyes, ou dans mon entourage personnel, je me contenterai donc d'un remerciement général en espérant qu'ils se reconnaîtront en lisant ces mots, et ne m'en voudront pas de ne pas les nommer personnellement.

Quelques remerciements personnels s'imposent tant la présence de ces personnes a été précieuse durant ces dernières années. Que ce soit pour des voyages en stop, des travaux, des projets associatifs, sportifs ou professionnels, Lou Grimal m'a accompagné avec son enthousiasme contagieux, enthousiasme qui m'a naturellement motivé dans le cadre de ce doctorat. Florence Grégoire non plus n'a pas été lasse de me soutenir : le sentiment maternel y est peut-être pour quelque chose. De manière générale, la famille Grégoire m'a apporté un soutient non négligeable. Mais il s'agissait aussi de bon nombre de petits mots reçus de la part de mes amis : Diane, Mohamed, Doriane, Arnaud, Maxime, Mathieu, Matthieu, Mathieux (et désolé d'avoir toujours mélangé vos prénoms), Benjamin, Floriane, Florian, Sophie, Ugo, Paul, Apolline, Michael, Hoai Phuong, Gauthier, Romain, Romain, Romain (vous vous reconnaîtrez), Léonore, Joffrey, Nina, Yiying, Tuan, Alexandre, Jean, Jade, Guillaume, Santiago, Julie, Cédric, Ines, Thomas, Orsolya, Mark, Rémi... Une thèse est bien plus qu'une expérience solitaire finalement !

Ces premières années dans le milieu de la recherche m'ont montré bien des mécanismes, et si le contexte de ma thèse m'a permis d'accéder aisément aux papiers scientifiques, je tiens à remercier, pour ceux qui n'ont pas les mêmes moyens, Alexandra Elbakyan et son initiative, et plus généralement les adeptes de l'*open science*.

Enfin, je tiens à remercier toutes ces personnes citées, mais également celles non mentionnées, pour avoir supporté mon cynisme et je tiens également à féliciter celles qui ont su discerner mon sarcasme - dans ces remerciements comme dans la vraie vie. Il ne me reste plus qu'à souhaiter au lecteur de parcourir ce document avec autant de plaisir que j'en ai eu à l'écrire.

Table des matières

1 Introduction

> *Si j'avais le pouvoir d'achat,*
> *J'achèterais plein d'objets sans fil,*
> *J'achèterais un écran plat,*
> *La vie serait plus facile...*
>
> **— Super pouvoir d'achat**
> La chanson du dimanche

1.1 Motivations et contributions

Dès l'élaboration du cahier des charges d'un projet informatique, il est désormais habituel d'y voir une considération importante pour la sécurité. Dans le cadre d'un projet mettant en oeuvre des objets connectés, ce même cahier des charges, en plus des besoins fonctionnels et des limitations budgétaires, spécifie des contraintes d'encombrements, d'autonomie, de réactivité... Répondre à toutes ces contraintes est parfois délicat, voir impossible : il s'agit dans ce cas de proposer le meilleur compromis qui réponde au cahier des charges.

Augmenter la sécurité rentre souvent en conflit avec l'augmentation de l'autonomie ou de la réactivité d'un objet connecté. La protection absolue d'une donnée étant impossible [1], **la sécurité n'est pas un interrupteur ouvert-fermé mais un curseur à faire varier au même titre que les autres contraintes : dimension de la batterie, puissance du micro-contrôleur... Mais alors, comment faire varier ces curseurs ? Sur quels critères ? Selon quels paramètres ?**

L'objectif final de ce document est d'**offrir une méthodologie aux concepteurs de systèmes embarqués afin de dimensionner au mieux leurs réseaux de capteurs selon l'influence des paramètres qui seront évoqués dans ce document.**

1. Ce point est abordé dans l'approche historique de la partie 2. Aujourd'hui, certains acteurs de la sécurité informatique proposent un nouveau (vieil) algorithme - le chiffrement de Vernam - offrant une sécurité absolue avec intervention de phénomènes quantiques aléatoires. La réalité de ces affirmations reste toutefois loin d'être pratique et concrète.

1.2 Contexte de la recherche

Ce travail de recherche a été abordé dans le cadre d'une collaboration entre l'entreprise VIGILOCK et l'équipe de Modélisation et Sûreté des Systèmes (M2S), de l'Institut Charles Delaunay à l'Université de Technologie de Troyes. VIGILOCK conçoit et programme des systèmes embarqués dans le domaine de l'industrie et de la domotique urbaine.

La proximité avec les industriels et le marché des objets connectés a rendu ce travail très opérationnel et pratique tandis qu'un cadre théorique rigoureux et scientifique a été apporté par l'état de l'art et le milieu de la recherche.

1.3 Plan

La problématique évoquée impose de bien définir les termes déjà employés jusqu'ici, notamment autour de la sécurité des communications qui sera abordée dans la partie 2 après une mise en contexte historique du problème.

Des exemples d'applications sont proposées dans la partie 3, plus spécifiquement autour de l'Internet des Objets (IoT pour *Internet of Things*). Ces exemples donnent déjà une idée des critères et paramètres à prendre en compte pour améliorer la performance d'un réseau de capteurs sans fil. Justement, cette partie se conclue sur un état de l'art à propos des problématiques d'optimisation autour des contraintes sur les réseaux de capteurs.

Le problème étant posé, cadré et défini, la partie 4 aborde les influences mutuelles entre autonomie, disponibilité et sécurité en proposant différents outils de mesure de ces critères appliqués aux systèmes embarqués. L'influence des paramètres est abordée selon deux points de vue : celui d'un système embarqué émetteur et celui d'un système embarqué récepteur.

La partie 5 utilise les notions et les paramètres évoqués dans la précédente afin de modéliser le problème à l'aide de réseaux de Petri. En plus de modéliser les systèmes, les réseaux de Petri permettent également de simuler leur comportement en fonction des paramètres et de l'environnement où ils évoluent. Ces outils de modélisation et simulation permettront d'estimer une indisponibilité d'un système embarqué selon le niveau de sécurité appliqué et les grandeurs mesurées : il s'agit finalement de créer l'outil d'aide à la décision grâce à la méthodologie définie.

Enfin, la partie 6 conclura ce travail en récapitulant les résultats et en apportant un regard critique sur ces travaux, tout en envisageant des axes d'améliorations dans le cadre de futures recherches.

Mais avant d'aborder le coeur de la thèse, une introduction historique sur le traitement cryptographique de l'information est abordée. Un tel historique permet d'expliquer comment les concepteurs de systèmes embarqués en sont arrivés à devoir réaliser les compromis entre les paramètres évoqués précédemment.

La sécurité des communications

2

> *Le code c'est "le code" ? Ça va, ils se sont pas trop cassé le bonnet, pour l'trouver celui-là !*
>
> **— Perceval de Galles**
> Kaamelott, Livre III, Poltergeist, écrit par Alexandre Astier.

2.1 Approche historique

D'abord diplomatique et militaire, l'information à protéger s'est propagée au milieu professionnel, puis au grand public, avant de se retrouver dans les objets du quotidien, de la ville, de l'industrie et du militaire - la boucle est bouclée.

Selon le type d'information, le niveau de protection peut varier. Une information diplomatique peut nécessiter une protection sur plusieurs années tandis qu'une information contrôlant un processus industriel a une durée de vie potentiellement beaucoup plus courte.

Si les informations échangées par la domotique peuvent paraître insignifiantes face aux informations militaires, il ne faut pas non plus négliger le potentiel d'attaques criminelles auxquelles les utilisateurs de la domotique peuvent être exposées. Par exemple, l'envoi d'informations non protégées peut indiquer à une personne malintentionnée qu'un logement est actuellement inoccupé (consommation électrique très stable).

Considérant cela, il peut être rassurant d'appliquer la plus grande sécurité partout. Cependant, l'évolution des algorithmes cryptographiques et de leurs attaques demande des ressources de calculs de plus en plus grandes. Ces puissances de calcul ont prouvé leurs limites d'abord pour les humains qui réalisaient ces calculs, puis pour les outils mécaniques, et enfin pour les objets informatiques et les systèmes embarqués.

En effet, le domaine de la sécurité des communications est le résultat d'une suite d'avancées mathématiques, technologiques, et de compromis. Une fuite en avant pourrait être évoquée, tant les technologies d'aujourd'hui permettent à la fois de renforcer la sécurité et de la briser, mais si cette fuite en avant s'est accélérée, elle est pourtant présente depuis longtemps.

2.1.1 Les premiers concepts de cryptographie

L'utilité de sécuriser la transmission d'un message est apparue bien avant l'informatique, notamment pour des besoins stratégiques. Il est ainsi peu surprenant de voir que ce domaine s'est d'abord perfectionné dans le milieu militaire.

César, alors général des armées romaines, utilisait ce qui a plus tard été appelé le *chiffrement de César* pour converser avec ses alliés. Ce chiffrement, relativement simple, consiste en une substitution mono-alphabétique : chaque lettre de l'alphabet est décalée de *N* rangs. Si le destinataire du message connaît *N*, il peut alors déchiffrer le message en décalant les lettres du message reçu dans le sens opposé. Le secret repose dans ce cas uniquement sur *N*, et, à l'époque de César, sur le secret de l'algorithme utilisé. *N* permettant le chiffrement et déchiffrement d'un message, ce nombre peut ainsi être appelé la clé du chiffrement de César, et cette clé a tout intérêt à rester cachée des personnes autres que l'émetteur et le destinataire.

Mathématiquement, cet algorithme consiste à additionner chaque lettre du message de *N* rangs pour chiffrer et de soustraire de *N* rangs pour déchiffrer. Plutôt qu'un nombre, l'envoyeur choisissait une lettre, la lettre clé. Additionner des lettres n'est pas forcément intuitif, aussi faut-il attribuer une valeur à chaque lettre : $A = 0$, $B = 1$, $C = 2$, ..., $Z = 25$. Afin de rester dans l'ensemble $\{0, ..., 25\}$ il convient de réaliser ces calculs modulo 26 [1]. La figure 2.1 est un exemple du chiffrement de César du mot $TROYES$ avec la clé $B = 1$.

Heureusement pour César, ses adversaires, tant en Gaule qu'à Rome où il préparait son retour en tant que consul, n'ont pas réussi à déchiffrer les messages interceptés. Mais ce chiffre, relativement simple, ne résista pas longtemps aux cryptanalystes, terme désignant ceux qui s'emploient (ou sont employés) à casser les méthodes de chiffrement ou les codes [2].

1. Le domaine de l'arithmétique modulaire, permet de réaliser des opérations dans des ensembles discrets et finis. Dans ce domaine, les calculs s'effectuent comme pour calculer l'heure dans le domaine $\{0, ..., 24\}$ (14h + 12h = 02h) : il s'agit alors d'une opération modulo 24 qu'il est possible de réécrire $14 + 12 = 2 \mod 24$.

2. Un chiffrement opérera sur des valeurs (lettre, mots, mots binaires, valeurs décimales...) sans se soucier de leur sens tandis qu'un code échangera des valeurs par d'autres à partir d'un répertoire significatif, toujours identique. Il s'agira donc de distinguer le décodage du déchiffrement. Retrouver la signification des hiéroglyphes ou du Linéaire B relève du décodage tandis que retrouver le message de

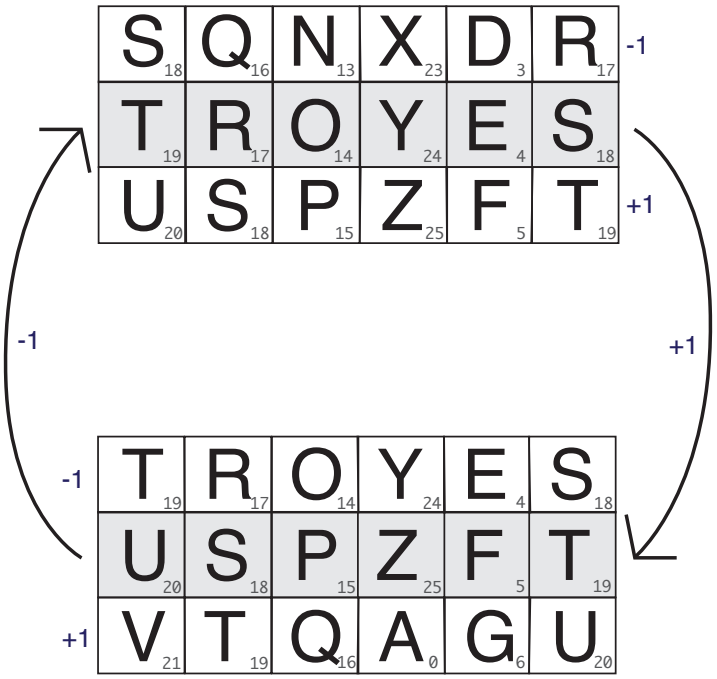

Fig. 2.1.: Chiffrement de César

Il est intéressant de noter que l'un des premiers systèmes de chiffrement et déchiffrement se basait déjà sur le calcul modulaire, base des logiques de chiffrements informatiques actuelles.

Depuis, de nombreux systèmes de chiffrement ont émergé, donnant un avantage certain aux utilisateurs de ces nouvelles techniques, le temps que les cryptanalystes trouvent un moyen de casser ledit système et rendent à nouveau inefficaces les méthodes développées par les cryptographes.

Ces découvertes cryptanalytiques n'étaient pas publiées immédiatement. La stratégie impose parfois de faire croire à la cible que son secret reste inviolé, permettant ainsi aux cryptanalystes d'intercepter et casser un nombre important de messages pouvant révéler des informations cruciales.

En 1587, Marie Stuart, reine d'Écosse, en fit les frais durant son procès où les échanges épistolaires qu'elle pensait sûrs furent décryptés, prouvant ainsi sa culpabilité et celle de ses complices. Les différents cercles de pouvoir virent alors l'intérêt de casser les méthodes de chiffrement et se munirent de cabinets noirs : des équipes de cryptanalystes et cryptographes permettant de sécuriser les communications

César sans connaissance de la clé relève du décryptage et retrouver le message de César en connaissance de la clé relève du déchiffrement

sensibles et de décrypter (trouver le message sans connaissance de la clé) celles des autres pays [49].

Ainsi [3], l'excès de confiance dans la sécurité d'un support de communication provoque la divulgation d'informations de grande importance. Il vaut donc mieux converser en public en sachant être écouté, qu'en public en pensant ne pas l'être. Autrement dit, pour le secret des communications, une méthode de chiffrement réputée inviolable est bien pire qu'une discussion publique.

2.1.2 Le chiffrement parfait ?

Longtemps, une bonne méthode de chiffrement tenait du secret de la méthode utilisée, mais les techniques des cryptanalystes se perfectionnant, cela ne suffisait plus. Kerckhoffs, stratège militaire, publia au XIXe siècle ses fameuses lois dans *La cryptographie militaire* [25] : "

1. *Le système doit être matériellement, sinon mathématiquement, indéchiffrable ; Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi ;*
2. *La clef doit pouvoir en être communiquée et retenue sans le secours de notes écrites, et être changée ou modifiée au gré des correspondants ;*
3. *Il faut qu'il soit applicable à la correspondance télégraphique ;*
4. *Il faut qu'il soit portatif, et que son maniement ou son fonctionnement n'exige pas le concours de plusieurs personnes ;*
5. *Enfin, il est nécessaire, vu les circonstances qui en commandent l'application, que le système soit d'un usage facile, ne demandant ni tension d'esprit, ni la connaissance d'une longue série de règles à observer.*"

Ces principes sont aujourd'hui encore le fondement de la cryptographie moderne. et sont souvent résumés en une seule phrase : le secret d'une communication doit reposer uniquement sur le secret de la clé. Les termes de méthodes ou systèmes de chiffrement ont été remplacés aujourd'hui par algorithmes de chiffrement : une suite d'opérations permettant à l'aide de la clé de transformer un message clair (lisible par tous), en un message chiffré. L'algorithme de déchiffrement est la méthode consistant à récupérer le message clair à partir du message chiffré. Chaque algorithme de chiffrement a donc son algorithme de déchiffrement.

3. L'exemple ci-après est à considérer comme une anecdote supplémentaire relativement ludique. Au XVIe siècle, François Viète, cryptanalyste français au service de Henry IV, roi de France, décryptait avec une facilité déconcertante les messages espagnols. Ces derniers, tellement sûrs de leur code, allèrent au Vatican accuser François Viète de pactiser avec le diable pour déchiffrer leurs messages. Ce qu'ils ignoraient, c'est que les cryptanalystes du Vatican décryptaient également les messages espagnols en secret. En plus de dévoiler leurs secrets en étant trop sûrs d'eux, les espagnols s'étaient ridiculisés face aux différentes cours européennes [46].

La mise en pratique réelle des lois de Kerckhoffs implique de fortes notions mathématiques afin que la clé soit la plus robuste possible et que l'algorithme utilisé ne laisse pas transparaître d'informations sur la clé ou le message.

Une application directe des lois de Kerckhoffs est le chiffre de Vernam [4]. La reprise par Vernam de l'algorithme de Vigenère (une extension polyalphabétique du chiffrement de César) permet un secret absolu. Ce secret absolu repose sur :

1. une clé de la longueur du message à chiffrer,
2. une utilisation unique de la clé,
3. un choix aléatoire de la clé.

La démonstration mathématique du secret absolu de l'algorithme de Vernam, également appelé algorithme à masque jetable (*One-time pad* en anglais), a été prouvée plusieurs années plus tard par Claude Shannon [48]. En cas d'interception d'un message, chaque clé testée par l'attaquant a la même probabilité d'être à l'origine du message chiffré. Autrement dit, $P(E|M) = P(M)$ où :

— $P(E|M)$: la somme des probabilités de toutes les clés qui produisent le message chiffré E à partir de M, ou, autrement dit, la probabilité d'avoir un message chiffré E si le message M est choisi.

— $P(M)$: la probabilité d'avoir le message M.

Cette équation (le théorème 6 dans [48]) est une condition suffisante pour avoir un secret absolu en se basant sur une clé à usage unique et totalement aléatoire (masque jetable). Pour être à usage unique, la clé doit donc être au moins de la taille du message à chiffrer, sinon elle serait répétée.

L'aspect aléatoire de la clé est important pour éviter de donner des indices à un attaquant potentiel. Par exemple, la génération de nombre pseudo-aléatoire (PRNG) en C++ utilise la fonction *rand*. Cette fonction nécessite une graine, c'est-à-dire une valeur de référence à partir de laquelle va se baser l'algorithme pour créer une nouvelle série de valeurs aléatoires. En particulier, la fonction *rand* est souvent utilisée avec la graine *time(null)*, fonction qui donne le temps du système à l'instant de l'appel. Avec une bonne synchronisation, l'attaquant peut alors *deviner* l'aléatoire à l'origine d'une séquence ou d'une clé [59].

Si la génération de nombres pseudo-aléatoires est souvent suffisant sous réserve d'une bonne implémentation, la génération purement aléatoire est quant à elle possible à l'aide, par exemple, de mesures d'évènements à base de radioactivité ou de phénomènes quantiques [5].

4. La paternité de cet algorithme est discutable : Frank Miller en a posé des bases en 1882 tandis que Gilbert Vernam l'inventa lui en 1917 avant que Joseph Mauborgne ne l'améliora plus tard en proposant une clé totalement aléatoire à usage unique.

5. Bien d'autres évènements peuvent être mesurés. Par exemple, l'utilitaire Putty pour les communication à base de clés SSH propose de générer une paire de clés en se basant sur les mouvements

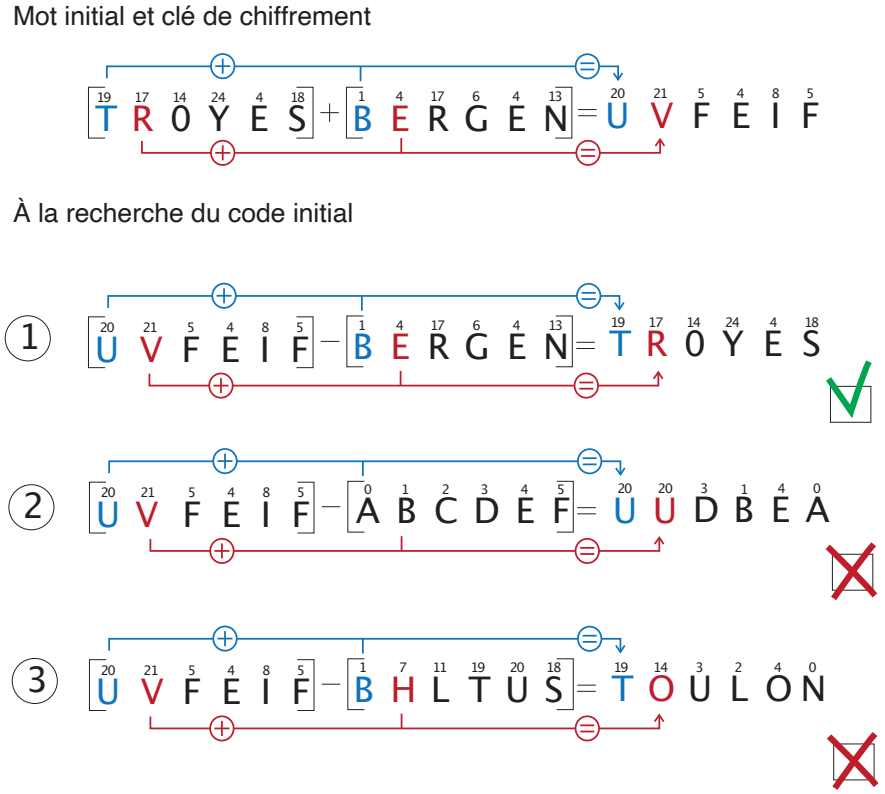

Fig. 2.2.: Chiffrement de Vernam

Pour en revenir au secret absolu, ce dernier peut s'expliquer comme décrit sur la figure 2.2, représentation graphique de la conclusion de la démonstration de Shannon. Il s'agit de considérer un message chiffré M_c intercepté par un attaquant sans connaissance de la clé de chiffrement $K = [B, E, R, G, E, N]$. Dans l'exemple de la figure 2.2, $M_c = [U, V, F, E, I, F]$. L'attaquant, en essayant diverses clés possibles, peut tout aussi bien tomber sur $M_3 = [T, O, U, L, O, N]$ (avec la clé testée $K = [B, H, L, T, U, S]$) que sur $M_1 = [T, R, O, Y, E, S]$ (le message d'origine avec la bonne clé K), ou bien n'importe quel autre message dépourvu (ou pourvu) de sens. Ainsi, sans indication aucune sur K et le message d'origine, l'attaquant ne pourra pas savoir si le lieu de rendez-vous est Troyes ou Toulon.

Autrement dit, le texte chiffré ne donne aucune indication sur le texte clair : $P(M|C) = P(M)$.

Le secret repose donc, comme le recommande Kerckhoffs, sur la clé et uniquement sur celle-ci. Toutefois, le principe de Kerckhoffs implique un système de gestion de clés avec toute une chaîne de confiance pour la génération et la transmission de ces

de la souris de l'utilisateur. La position à un instant t dépend toutefois de la position précédente et la question du niveau d'aléatoire requis pour une bonne sécurité reste en suspens. D'autres équipements comme des Smart-phones utilisent la luminosité captée par l'appareil photo.

dernières. L'usage unique d'une clé aléatoire qui doit être partagée entre les deux entités communicantes rend donc le chiffrement de Vernam peu pratique. Seules des communications critiques et particulièrement sensibles s'équipent de ce genre de chiffrement où l'échange des clés est crucial [6].

Ce système de chiffrement parfait étant difficile à mettre en place à grande échelle, les cryptographes firent un premier compromis : d'autres systèmes de chiffrement furent implémentés et utilisés à différents degrés, malgré l'imperfection de ceux-ci.

2.1.3 La mécanisation du chiffrement

Au XIXème siècle, l'invention du télégraphe apporte un problème qui est toujours d'actualité : envoyer un contenu chiffrer à un destinataire est plus coûteux qu'envoyer un contenu clair [7]. En cause, la capacité de l'opérateur à taper rapidement un message clair (dont il retiendra les mots) qu'un message chiffré (auquel il doit se rapporter pour envoyer chaque lettre, l'ensemble formant une phrase inintelligible). Malgré cet inconvénient, le télégraphe resta pendant longtemps un support très largement utilisé à grande échelle, pour des messages clairs tout comme pour des messages chiffrés.

Un inconvénient du télégraphe est l'infrastructure : tout comme la fibre aujourd'hui, il faut installer un ensemble d'équipements sur de grandes étendues (installation des pylônes, des câbles...) avant une première communication. L'invention de la radio apporta un avantage majeure pour certaines utilisations : marine, aviation, ou unités terrestres mobiles. En revanche, la radio rend l'interception de messages beaucoup plus aisée à l'aide d'une simple antenne correctement orientée, là où il fallait physiquement *se brancher* sur le réseau dans le cas du télégraphe.

C'est pourquoi en cas de progression rapide de l'ennemi sur un territoire, il était fréquent de détruire ses propres installations télégraphiques, forçant ainsi l'ennemi à utiliser un support radio pour envoyer ses messages et faciliter l'interception de ceux-ci. À une époque où aucune méthode de chiffrement sûre n'était utilisée ou utilisable - il a été vu que l'algorithme de Vernam n'était pas pratiquement utilisable à cause de la gestion des clés - il ne s'agissait que d'une question de temps pour qu'un

6. Par exemple, ce système aurait été utilisé pour le téléphone rouge entre Washington et Moscou où les clés étaient imprimées sur deux bandes magnétiques identiques transmises aux diplomates. Autre cas du chiffrement de Vernam, un papier retrouvé sur le corps de Che Guevara montra que ce dernier aurait également utilisé cette méthode de chiffrement avec Fidel Castro, mais la gestion des clés pour leurs communications reste un mystère [34]. Ces exemples montrent à quel point la gestion et le transport des clés est un point critique de la communication sécurisée. Les clés sont ici transportées par un canal de communication dédié et de confiance (les diplomates dans le cas du téléphone rouge).

message critique soit décrypté. Ce fut le cas lors de la première guerre mondiale avec le télégramme de Zimmerman [7].

L'affaire du télégramme cachait un avenir incertain pour les cryptanalystes : la multiplication des messages interceptés à décrypter devenait problématique. Ils avaient conscience que dès qu'une méthode de chiffrement impliquant un temps plus important de cassage pour les cryptanalystes verrait le jour, la tâche deviendrait impossible. Et justement, dans les années 20, la machine Enigma d'Arthur Scherbius permit une nouvelle avancée des cryptographes.

Le principe d'Enigma

La machine Enigma est un système de cryptographie permettant de chiffrer des messages de manière mécanique. Cette section détaille particulièrement son mécanisme afin de mettre en avant l'impact de sa cryptanalyse et de ses vulnérabilités sur la protection des données. Cette logique sera appliquée plus tard dans le document pour les algorithmiques de sécurité plus récents et courants, sans rentrer spécifiquement dans le détail de leur cryptanalyse.

Ce système repose sur une succession de rotors interchangeables. Un rotor permet un chiffrement par substitution poly-alphabétique. À chaque substitution, le rotor tourne d'un cran, permettant de changer la prochaine substitution et au bout de 26 substitutions, le rotor revient à sa position d'origine - les substitutions deviennent alors redondantes.

C'est pourquoi Arthur Scherbius introduisit deux autres rotors : le premier rotor substitue une lettre qui est envoyée au deuxième rotor qui substitue le résultat du rotor 1 et envoie la nouvelle lettre au rotor 3 qui la substitue à nouveau. Après cette opération, seul le rotor 1 a avancé d'un cran. Le rotor 2 avancera d'un cran une fois que le rotor 1 aura avancé 26 fois, et le rotor 3 avancera d'un cran une fois que le rotor 2 aura avancé 26 fois (comme un compteur kilométrique). Au lieu d'avoir une substitution unique pour 26 lettres, ce système permet un chiffrement unique pour $26 * 26 * 26 = 17576$ lettres.

7. Lorsque Arthur Zimmermann prit ses fonctions de ministre des affaires étrangères en Allemagne en novembre 1916, les américains s'en réjouirent, notamment le président Wilson qui espérait garder une position de médiateur en restant neutre. Cependant, Arthur Zimmermann jouait double jeu et souhaitait au contraire entrer en guerre contre les États-Unis en s'alliant avec le Mexique et le Japon. Au moment d'envoyer le message au Mexique, les allemands ne purent utiliser les lignes terrestres et maritimes : celles-ci avaient été sabotées par les anglais. Ils se rabattirent alors sur les communications radios, que les anglais purent intercepter plus aisément. Le Bureau 40, service anglais chargé du décryptage des messages ennemis, ne tarda pas à casser le télégramme de Zimmermann et dévoila en temps voulu le contenu au président Wilson. La neutralité des Etats-Unis ne pouvait plus durer, et l'interception de ce message changea le cours de la première guerre mondiale [21].

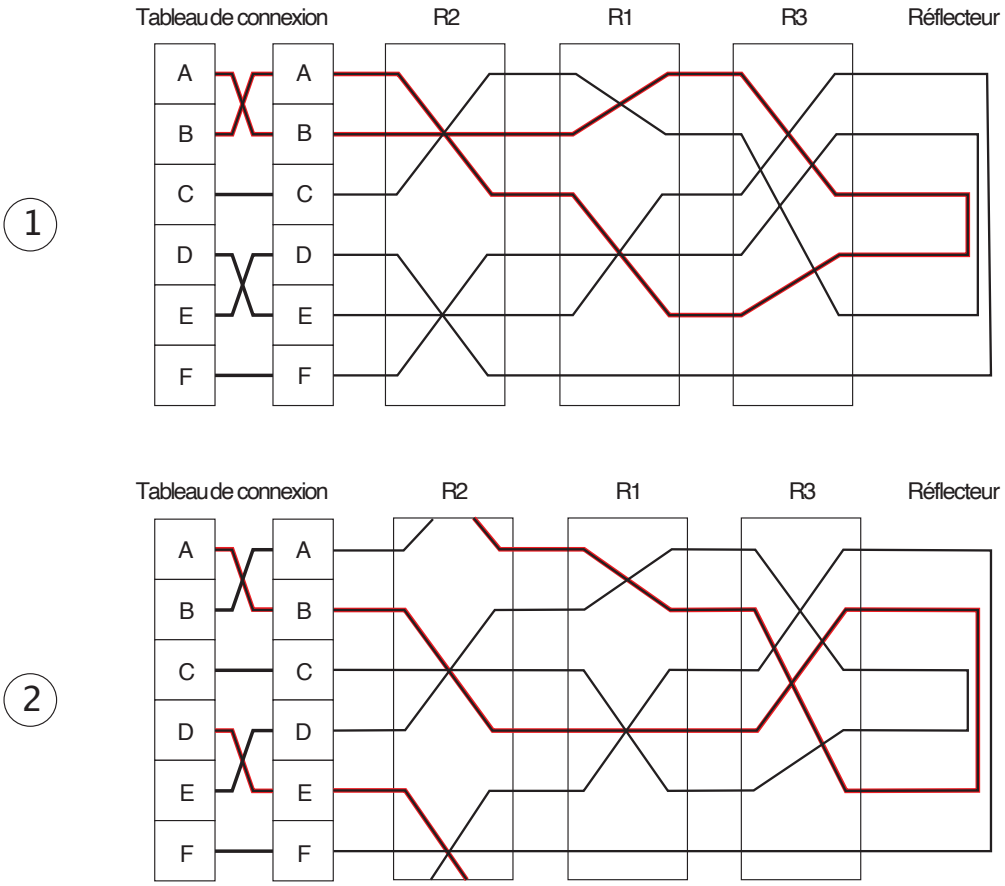

Fig. 2.3.: Enigma avec 6 lettres

Les rotors étant interchangeables, l'ensemble des clés est composé des 17576 positions initiales possibles, des 6 dispositions possibles des rotors : R1|R2|R3 - R1|R3|R2 - R2|R1|R3 - R2|R3|R1 - R3|R1|R2 et R3|R2|R1 ce qui donne 105 456 possibilités à tester pour trouver la bonne configuration [2]. Si ce nombre peut paraître décourageant pour un cryptanalyste solitaire, ce n'est pas forcément le cas pour des moyens étatiques embauchant un nombre important de cryptanalystes. Arthur Scherbius a ainsi apporté une amélioration supplémentaire à son système, permettant d'agrandir considérablement l'espace des clés possibles.

L'ingénieur a effectivement ajouté un tableau de connexions précédant les rotors. Ce tableau de connexions permet de changer certaines lettres avant qu'elles n'arrivent au premier rotor. L'opérateur de la machine place 6 fils pour substituer les lettres qu'il souhaite. Par exemple $A \leftrightarrow B$, $C \leftrightarrow D$, $E \leftrightarrow F$, $G \leftrightarrow H$, $I \leftrightarrow J$, $K \leftrightarrow L$. La figure 2.3 [8] récapitule le fonctionnement pour 6 lettres (au lieu des 26 de l'alphabet par soucis de clarté) et 2 connexions (au lieu des 6) mais toujours 3 rotors.

8. Le concepteur de boites de céréales arrivant jusqu'ici est libre de compléter et adapter ce schéma afin de le proposer pour petits et grands.

Sur cette figure, à l'étape 1, l'utilisateur appuie sur la lettre A, ce qui donne un signal électrique parcourant les rotors R2, R1 et R3 en suivant le trait rouge pour finalement donner la lettre B. Le rotor R2 change de position d'un cran, et si l'utilisateur appuie à nouveau sur la lettre A (étape 2), un signal suit le trait rouge donnant cette fois la lettre D.

L'invention, brevetée en 1918, fut proposée d'abord aux diplomates et aux industriels mais le prix élevé de la machine découragea ces derniers. Arthur Scherbius apportait un discours alarmiste auprès des industriels, leur indiquant que les pertes provoquées par un message intercepté par une entreprise concurrente seraient bien plus grandes que le coût pour s'équiper de machines Enigma, mais cet argument ne les convint pas. Quant à l'armée allemande, elle refusait de croire que leur système de chiffrement était obsolète et, lorsque l'affaire Zimmermann était évoquée, elle préférait croire à la trahison du Mexique.

Cependant, *The World Crisis*, publié par Winston Churchill en 1923, détaillait la manière dont les britanniques s'étaient procurés et avaient décrypté le télégramme de Zimmermann. L'armée allemande se dota alors de machines Enigma et à l'aube de la seconde guerre mondiale, était équipée de plus de 30 000 machines [49].

Première analyse

Si Arthur Scherbius pensait son système inviolable, c'est qu'il avait de solides arguments pour soutenir cette assertion : son système a le mérite de reposer sur la clé (la configuration de la machine) et la manière dont elle est gérée. La machine ayant été produite à grande échelle, il fallait s'attendre à ce que l'ennemi la récupère et comprenne la logique utilisée. L'armée allemande a tout de même tenu à rendre leurs machines plus spécifiques que les machines "publiques" afin de ralentir les avancées de l'ennemi en vue d'une tentative de cryptanalyse.

Le bureau du chiffre polonais se reposa énormément sur le fonctionnement d'Enigma. En considérant la complexité de la machine, les polonais décidèrent de recruter des mathématiciens en plus des linguistes, habituellement perçus comme les casseurs de codes à l'époque. Car en effet, les polonais en sont rapidement arrivés à la même conclusion qu'Arthur Scherbius : le système qu'ils avaient entre les mains n'étaient pas utilisables sans connaissance de la clé pour déchiffrer le message ennemi. La technique classique alors était d'utiliser l'ensemble des clés possibles. Or, les mécanismes mis en jeu donnent un espace de clés bien trop important.

En reprenant l'exemple d'un espace de 6 lettres comme sur la figure 2.3 avec 2 connexions au lieu de 6, cela représente un espace de taille égale à $N * R * C = K$ où :

— N : le nombre de positions initiales possibles des rotors, soit N = 6 * 6 * 6 = 216,

— R : le nombre de dispositions des rotors R1, R2, R3 soit R = 6,

— C : le nombre de positions des 2 connexions possibles. Pour appairer deux fois deux lettres parmi les 6, cela équivaut à choisir par exemple A-B pour la première connexion. Il y a donc 6 possibilités pour la première lettre, et 5 pour la deuxième (il faut retirer le A déjà pris) soit 6x5 possibilités. Choisir A puis B revient ici à choisir B puis A il faut donc diviser le nombre de possibilité par deux. Pour la première connexion, il existe donc $\frac{6*5}{2}$ possibilités.

  Pour la deuxième connexion (D-E) il reste 4 choix possibles pour choisir la troisième lettre (D) et 3 choix possibles pour la quatrième lettre (E), et avec le même principe de dédoublement de la paire, il résulte donc un total de $\frac{6*5}{2} * \frac{4*3}{2}$ combinaisons.

  Or l'ordre des connexion n'a pas d'importance : il est tout à fait possible de prendre la paire D-E avant la paire A-B, il convient donc de diviser par deux le nombre de possibilités ce qui donne C = $\frac{6*5}{2} * \frac{4*3}{2} * \frac{1}{2} = 45$.

Ainsi, l'espace de clé dans ce schéma simplifié d'Enigma est de K = $216 * 6 * 45 =$ 58 320.

En appliquant la même logique pour la machine Enigma standard à 3 rotors de 26 lettres et 6 connexions, il en résulte l'espace de clé à la taille de $K = N * R * C$ avec cette fois :

— $N = 26 * 26 * 26 =$ 17 576

— $R = 6$

— $C = \frac{26*25}{2} * \frac{24*23}{2} * \frac{22*21}{2} * \frac{20*19}{2} * \frac{18*17}{2} * \frac{16*15}{2} * \frac{1}{6!}$ où $\frac{1}{6!}$ représente le nombre de possibilités d'ordonner ces six connexions. Soit C = $\frac{26!}{14!*2^6*6!}$ =100 391 791 500.

Donc $K = N * R * C$ =10 586 916 764 424 000, qui est donc le nombre de combinaisons à tester...

Le raccourci de Rejewski

Les services de cryptanalyse français et britannique, face à ces difficultés, abandonnèrent l'idée de casser Enigma [9]. Le bureau du chiffre polonais ne se laissa pas

9. Et également face à une certaine assurance et un sentiment de supériorité sur les allemands au sortir de la première guerre mondiale...

décourager par la complexité du problème. Le cassage d'Enigma pour la Pologne fut sans aucun doute motivé par la position particulière de cette dernière entre les deux superpuissances au nationalisme expansionniste montant qu'étaient l'URSS et le troisième Reich. L'investissement financier et temporel de ce jeune pays pour casser Enigma permit au jeune Marian Rejewski de proposer une approche qui réduisit considérablement l'espace des clés à tester.

Les allemands avaient bien compris qu'envoyer tous les messages avec une seule clé rendrait le système vulnérable aux attaques par analyse de fréquence. Il est en effet possible avec une seule clé (une série de 6 connexions données, le placement et l'orientation de chaque rotor fixés) de chiffrer une seule et unique fois 17 576 lettres ($= 26 * 26 * 26$), ce qui équivaut au nombre de positions différentes possibles que prendront les rotors avant de revenir à leur position d'origine. Au-delà de ce nombre, le processus de chiffrement se répète et l'analyse de fréquence s'applique plus aisément : il s'agit d'un chiffrement de Vigenère avec une clé relativement longue.

Les allemands mirent en place des carnets de clés journalières, c'est-à-dire que les opérateurs se référaient à la page du jour pour appliquer sur la machine Enigma la clé suivante :

1. Le tableau des 6 connexions.
2. La disposition des rotors : R2, R1, R3,
3. L'orientation de chaque rotor : N, F et S [10].

Mais étant limité à 17 576 lettres par jour (pour cette combinaison donc) les allemands mirent en place un protocole d'échange de clés temporaires partielles correspondant à une nouvelle orientation initiale de chaque rotor, par exemple RGB, tandis que la table de connexions et la disposition des rotors restent identiques.

Cette nouvelle orientation initiale permet au destinataire de régler sa machine afin de correctement déchiffrer le reste du message. Il est donc primordial de ne pas faire d'erreur sur la clé temporaire et les allemands décidèrent donc de répéter cette clé, ce qui donne RGBRGB pour le début du message. Voici un récapitulatif du processus d'utilisation d'Enigma [11] :

1. L'émetteur et le destinataire utilisent la même clé journalière,
2. L'émetteur écrit la clé partielle (doublée) RGBRGB qui sera chiffrée grâce à la configuration initiale journalière,

10. Chaque rotor possède pour chacun des 26 crans une lettre. Le rotor 1 doit proposer sa face supérieur sur le N, le deuxième sur le F, et le troisième sur le S pour l'exemple donné.

11. Il s'agit ici d'un protocole de communication d'échange de clés utilisant l'algorithme Enigma. Autrement dit, Enigma est implémenté de manière à compliquer une cryptanalyse de messages. Il ne suffit parfois pas de regarder la difficulté de cassage d'un algorithme pour définir son niveau de sécurité, il faut aussi considérer son implémentation qui peut la renforcer... ou introduire de nouvelles failles.

A	B	**C**	**D**	E	F	G	H	I	J	K	**L**	M	N	O	**P**	Q	R	S	T	U	V	W	**X**	Y	Z
U	Z	**A**	**G**	T	N	O	C	K	M	W	**P**	B	E	Q	**Y**	V	H	L	X	R	I	F	**D**	S	J

Tab. 2.1.: Chaînes Enigma de Rejewski

3. Pour la suite du message, l'émetteur oriente le rotor 1 sur le R, le rotor 2 sur le G et le rotor 3 sur le B,
4. L'émetteur envoie le reste du message avec cette nouvelle configuration,
5. Le destinataire déchiffre les six premières lettres avec la configuration initiale journalière pour trouver la clé temporaire partielle,
6. Le destinataire positionne les rotors sur la configuration temporaire des rotors (RGB),
7. Le destinataire déchiffre le reste du message avec cette nouvelle configuration,
8. L'émetteur et le destinataire remettent les rotors sur la configuration initiale journalière.

Partant du principe que la redondance est un des grands maux de la cryptographie, Marian Rejewski chercha à exploiter cette information recueillie par le service d'espionnage polonais. RGBRGB peut ainsi, une fois chiffré avec la clé journalière, donner le résultat LXCPDA. Marian Rejewski savait alors que le L et le P venaient de la même lettre. En récupérant suffisamment de messages, et sans connaissance de la clé temporaire, ni de la clé journalière, le mathématicien put reconstituer un tableau de liens entre chaque première et quatrième lettres, deuxième et cinquième lettres, troisième et sixième lettres, de chaque message reçu chaque jour. Au bout de suffisamment de messages, il arrivait à un résultat similaire au tableau 2.1. Les lettres liées du message clé chiffré (LXCPDA) sont en gras.

Une fois le tableau défini, Marian Rejewski établit des chaînes de lettres. Par exemple $A \to U \to R \to H \to C \to A$ ou encore $Z \to J \to M \to B \to Z$. Il remarqua que ces chaînes dépendent des positions initiales des rotors.

En plus de cette découverte, il remarqua que le tableau de connexions se contente de substituer certaines lettres. Plus précisément, si l'effet des rotors est omis, un mot comme bibliothèque peut devenir *titsiothèque* si le t substitue la lettre b et le s substitue la lettre l. Une analyse de fréquence (sur un ensemble assez conséquent de mots) permet alors de casser ces substitutions sans trop de difficultés. Il est donc relativement peu utile de s'attaquer à trouver directement le tableau de connexions choisi par l'opérateur.

Le mathématicien se concentra alors sur les 17 576 combinaisons de rotors possibles et les testa toutes pendant 10 ans afin d'avoir les chaînes générées par chaque combinaison. Lorsqu'un nouveau message arrivait avec LXCPDA en en-tête, il pouvait

chercher parmi les combinaisons documentées quelle est la position initiale des rotors pour ce message.

Marian Rejewski a ainsi proposé d'une part une méthode réduisant la complexité de cassage d'Enigma, et d'autre part, une simili attaque par dictionnaire : il a conçu une documentation où il suffisait de tester la bonne combinaison pour avoir la clé d'origine. Cette méthode est assez proche de ce qui existe aujourd'hui pour tester des mots de passe ou des clés de chiffrement usuelles.

Malheureusement, ses travaux ne seront pas utilisés dans l'immédiat, la Pologne se faisant envahir. Enigma restait donc pour le moment toute puissante et donna un avantage certain aux allemands dans le secret de leurs communications. Cette toute puissance était favorisée par l'automatisation du processus de chiffrement permettant de gérer une quantité plus importante de données.

Puissance de calcul cryptanalytique

En se basant sur les travaux de Marian Rejewski, Alan Turing, qui avait rejoint l'équipe de Bletchley Park en charge de la cryptanalyse d'Enigma, partait du principe que les allemands ne tarderaient pas à remarquer la faiblesse introduite par la répétition de la clé temporaire partielle. Il chercha donc à trouver d'autres pistes en reprenant le principe des chaînes de Rejewski.

Au lieu de se baser sur les clés journalières répétées en début de message, Alan Turing se basa sur des mots probables. Le plus couramment cité est le mot WETTER (météo en allemand) qui était présent dans de nombreux messages. Le mot WETTER a la particularité de faire parti des mots qui possèdent une chaîne interne au mot, c'est à dire qu'il advient un schéma du type $W \rightarrow LETTRE1 \rightarrow LETTRE2 \rightarrow W$ comme vu sur la méthode de Rejewski.

Alan Turing proposa un mécanisme, appelé bombe de Turing afin de tester différentes positions initiales et de retrouver ces chaînes pour des messages interceptés. Une fois une chaîne trouvée la bombe s'arrête sur une configuration correspondant à la configuration initiale. Si les premières bombes de Turing mettaient jusqu'à une semaine pour casser Enigma, les dernières versions mettaient jusqu'à 5 heures.

Il faut noter que les machines Enigma de l'armée allemande possédaient jusqu'à 5 rotors. Il n'y avait donc pas uniquement 17 576 orientation initiales possibles, mais en considérant les 60 possibilités de positionner 3 rotors parmi 5, il y avait $60 * 17576 = 1054560$ positions à tester. Du côté des cryptanalystes, les principales

améliorations venaient de la parallélisation des calculs. Cette technique est encore utilisée aujourd'hui pour tester plus rapidement les clés possibles.

Probablement conscient des avancées des cryptanalystes de Bletchley Park, les allemands renforcèrent leur système pendant la guerre. La *Kriegsmarine* s'équipa de machines Enigma avec 8 rotors possibles (parmi lesquels 3 devaient être sélectionnés) et d'un réflecteur configurable sur une des 26 positions possibles [12]. Mais au delà de la machine elle-même, les allemands avaient changé leur méthode de transmission de la clé du jour et avaient cessé d'envoyer des messages prédictibles. Les anglais durent s'en remettre à nouveau sur l'espionnage pour anticiper les mouvements de la marine allemande.

La méthode de Rejewski [13] et Turing permit néanmoins de mettre en évidences des informations que laissent fuiter la machine Enigma, même si ces informations (les chaînes sur les lettres) peuvent paraître abstraites dans un premier temps.

La machine Colossus, digne successeur électronique binaire de la bombe de Turing (électro-mécanique) à Bletchley Park, fut employée pour casser le chiffre de Lorenz utilisé par les allemands en parallèle d'Enigma [1]. Cette machine, conçue par Tommy Flowers, donna un avantage considérable aux anglais sur les allemands à partir de 1944 mais permit également de faire entrer le chiffrement dans une nouvelle ère : celle de l'informatique. La confidentialité de ces travaux jusqu'en 1975 et la destruction des plans de Colossus nécessitèrent cependant la réinvention de ces méthodes outre-Atlantique.

2.1.4 L'informatisation du chiffrement

Les architectures de type Von Neumann et de type Harvard qui furent développées dans la deuxième moitié des années 40 permirent d'adapter plus facilement une machine à un problème (comme Colossus ou les bombes de Turing) en la rendant programmable. Plus précisément, une mémoire permet de gérer les instructions et une mémoire permet de gérer les données (calculées ou intermédiaires, et demandées ou finales) tandis qu'une Unité Arithmétique et Logique (UAL) puise dans cette mémoire grâce à une unité de contrôle qui séquence les opérations. La différence entre les deux types d'architecture est la gestion de la mémoire : l'architecture de

12. Non abordé jusqu'ici, le réflecteur était présent sur les machines Enigma classiques, et permettait de renvoyer la lettre sur l'écran de l'opérateur. Il est présent en bout de chaîne sur la figure 2.3 pour que le signal traverse à nouveau les rotors. Sa position étant fixe, il n'avait aucun impact sur la configuration initiale, ce qui n'est plus le cas s'il devient configurable.

13. Marian Rejewski ignorait que ses travaux avaient été utilisés pour casser Enigma pendant la guerre. Il l'appris bien plus tard lorsque les travaux de Bletchley Park furent rendus publiques.

type Harvard sépare la mémoire d'instructions de la mémoire de données tandis que l'architecture de type Von Neumann les regroupe.

Si une machine conçue spécifiquement pour un problème est plus efficace qu'une machine programmable et programmée pour ledit problème, l'apparition de ces architectures permit tout de même de mutualiser une machine pour plusieurs tâches et ainsi rentabiliser les coûts de celles-ci. Les développements allèrent donc dans ce sens, ce qui permit une amélioration continue des composants en terme d'encombrement, de coût et de puissance de calcul.

Les nouvelles puissances de calcul permirent d'appliquer des algorithmes de chiffrement plus performants qui se basent sur une clé symétrique (la même clé est utilisée pour le chiffrement que le déchiffrement). Cependant, la gestion des clés restait problématique pour une utilisation à grande échelle du chiffrement.

Diffie, Hellman et Merkle résolurent le problème en inventant le chiffrement asymétrique pour l'échange de clés tandis que Rivest, Shamir et Adleman développèrent le premier algorithme de chiffrement asymétrique : RSA.

Les calculs demandés par la cryptographie asymétrique se trouvent être beaucoup plus gourmands en temps et en mémoire. Différents algorithmes sont présentés dans la partie 2.2 et des comparaisons de performances ont été effectuées à puissance de calcul égale dans la partie 4. Aujourd'hui, les ordinateurs personnels et smarphones possèdent la puissance nécessaire pour réaliser ces calculs. Parallèlement à la course à la puissance de calcul s'effectuait une autre course : celle de la miniaturisation et de la portabilité, ce qui a permis l'émergence des systèmes embarqués.

2.1.5 Les systèmes embarqués

Si pendant longtemps la loi de Moore [14] semblait rendre possible à la fois une plus grande puissance de calcul et une miniaturisation accrue, la limite semble aujourd'hui atteinte et des choix doivent être faits.

La miniaturisation des composants a permis, pour des applications spécifiques, de concevoir ce qui est communément appelé un système embarqué. Un système embarqué est un objet électronique à faible encombrement. Ces systèmes embarqués sont contrôlés par des micro-contrôleurs : des composants équipés d'une mémoire vive, d'un processeur et d'une mémoire flash. Le compromis d'un tel condensé de fonction-

14. La loi de Moore est une loi empirique mettant en évidence que le nombre de transistor sur un circuit intégré double tous les deux ans pour une même surface, doublant ainsi la puissance de calcul du circuit intégré.

nalités est un système à puissance de calcul plus faible qu'un ordinateur par exemple, mais avec une consommation énergétique bien plus faible et un encombrement moindre.

La plupart des micro-contrôleurs, comme les PIC (*Programmable Integrated Circuit* de Microchip, possèdent une architecture de type Harvard. En séparant l'accès à la mémoire de données et d'instructions, le processeur peut préparer l'appel d'instruction suivante en traitant la donnée actuelle, ce qui peut optimiser certaines opérations. Les systèmes embarqués étant conçus pour des besoins spécifiques, ils sont dédiés à une tâche - ou série de tâches. Cette première définition d'un système embarqué exclut donc les smartphones et PC industriels basés sur des systèmes d'exploitation multi-tâches.

Le domaine d'utilisation des systèmes embarqués est très vaste : les transports (automobile, aéronautique...), l'industrie (Usine du futur, Industrie 4.0), le médical (implants, hôpitaux), la domotique (habitat connecté, *SmartCity*)... Il est aisé de voir que les systèmes embarqués manipulent des données personnelles ou critiques : relevé de valeurs, identification, système de contrôles-commandes. Souvent, ces systèmes embarqués sont communicants et l'appellation les désignant est alors objets connectés (connectés à un poste de contrôle local, à un ordinateur ou à un smartphone sur demande). Et lorsque ces objets communiquent avec des équipements en réseaux, le terme Internet des Objets (IoT pour *Internet of Things*) s'applique.

Pour une utilisation locale tout comme pour une transmission distante, ces données doivent être traitées avec précaution et différents mécanismes de sécurité doivent être appliqués. Les questions qui émergent de ces constats et des contraintes des systèmes embarqués sont alors les suivantes :

1. Comment assurer un **niveau de sécurité suffisant** sur des systèmes à faible puissance de calcul face à des machines beaucoup plus puissantes sur lesquelles ont été développées de nombreuses solutions de cryptanalyses et cryptographies ?
2. Sur quels critères se baser pour faire les bons choix lors de la phase de conception d'un système embarqué ?
3. Comment définir et quantifier ces critères ?

Ces questions paraphrasent la problématique initialement présentée dans la partie 1. Bien que cette étude se focalise sur les systèmes embarqués, les principes peuvent s'appliquer à n'importe quel autre entité communicante nécessitant des mécanismes de sécurité, que ce soit César rédigeant des messages, ou encore Enigma. Seuls la valeurs des paramètres qui seront listés dans les partie prochaines changeront.

Avant toute chose, afin de quantifier le niveau de sécurité et le temps de calcul associé, une revue des algorithmes de sécurité les plus fréquemment croisés dans les systèmes embarqués est proposée ci-après.

2.2 Algorithmes de sécurité et vulnérabilité

Cette partie permet d'introduire la notion de temps de cassage, temps nécessaire pour casser un algorithme et retrouver le contenu du message ou la clé d'origine en considérant les vulnérabilités connues. Ce *temps* est exprimé en complexité algorithmique afin d'être indépendant du support matériel sur lequel tournera l'algorithme de cassage. La complexité algorithmique donne en effet un ordre de grandeur du nombre d'opérations nécessaires à son exécution.

2.2.1 Complexité et vulnérabilité algorithmique

La complexité algorithmique

Un algorithme est une succession d'opérations élémentaires (additions, multiplications...) pouvant être réalisées par une unité logique, comme un processeur. La durée d'exécution dépendra du processeur en question mais également de la rapidité d'accès à la mémoire persistante, voire à la mémoire morte si des données y sont stockées.

Donald Knuth a formalisé lors de sa carrière et de ses recherches la notion d'algorithme à travers cinq points principaux [27] :

— Finitude : "*Un algorithme doit toujours se terminer après un nombre fini d'étapes.*"

— Définition précise : "*Chaque étape d'un algorithme doit être définie précisément, les actions à transposer doivent être spécifiées rigoureusement et sans ambiguïté pour chaque cas.*"

— Entrées : "*Quantités qui lui sont données avant qu'un algorithme ne commence. Ces entrées sont prises dans un ensemble d'objets spécifié.*"

— Sorties : "*Quantités ayant une relation spécifiée avec les entrées.*"

— Rendement : "*Toutes les opérations que l'algorithme doit accomplir doivent être suffisamment basiques pour pouvoir être en principe réalisées dans une durée finie par un homme utilisant un papier et un crayon.*"

Trier un tableau de n éléments (les entrées) dépendra de la taille du tableau, n. Il en résulte le tableau trié (les sorties). Déduire la durée exacte de l'algorithme (rendement et finitude éventuelle) n'est pas possible sans avoir plus d'information sur

son implémentation : temps d'allocation et taille des variables, gestion de celles-ci... Il est en revanche possible d'avoir un ordre de grandeur dépendant uniquement des données et de leur taille, ici n.

Cet ordre de grandeur est noté $\mathcal{O}$ et la complexité des meilleurs algorithmes permettant de résoudre le problème de tri d'un tableau est dite d'ordre $n \log(n)$ ce qui est noté $\mathcal{O}(n \log(n))$. Pour $n = 1000$ éléments, cela donne environ 3 000 opérations élémentaires. Soit $t = 10$ ns le temps d'exécution d'une opération élémentaire pour le processeur implémentant l'algorithme cela donne une estimation de durée de $T = 30\ \mu s$.

Pour reprendre l'exemple introduit dans l'historique de la partie 1, la machine Enigma, en prenant en compte la table de connexions, nécessitait de tester l'ensemble des 10 586 916 764 424 000 clés possibles. Codé en binaire, ce nombre a une taille de $\log_2(10586916764424000) = 53.23313201$, soit 54 bits. Le cassage d'Enigma reposant sur le test de chaque clé possible, ce problème est d'une complexité exponentielle $\mathcal{O}(2^n)$, dépendant directement de la taille de la clé (ici n = 54 bits). Sur le même processeur, il faudrait alors un temps de $T = 2^{54} * t = 2^{54} * 10 = 2085$ jours pour trouver la clé.

Cette attaque effectuée sur Enigma, où il s'agit de tester l'ensemble des clés possibles, est dite attaque par force brute. La logique sera la même pour les attaques par force brute des prochains algorithmes qui seront abordés.

La vulnérabilité algorithmique

Il est convenu à ce stade du document que la sécurité d'un algorithme dépend de la complexité algorithmique pour casser ce même algorithme. En particulier, l'attaque par force brute qui consiste à tester toutes les clés possibles de l'algorithme dépend directement de la taille de la clé utilisée par l'algorithme. Mais il a été vu pour Enigma que les découvertes mathématiques de Rejewski et Turing avaient permis de faire baisser le nombre de possibilités de 10 586 916 764 424 000 à 17 576, soit d'une complexité de 2^{54} à 2^{15} car $\log_2(17576) = 14.10131915$. Au lieu de passer 2 085 jours avec un processeur permettant une opération élémentaire en $t = 10$ ns, il faudra désormais passer environ 326 μs pour tester toutes les possibilités de clés : une faille a été trouvée et exploitée.

Mais ce gain temporel est trop dépendant du matériel utilisé. Pour mesurer cette faille, il est intéressant de comparer le gain de complexité permis par celle-ci : il s'agit de la vulnérabilité introduite pas cette faille. Ainsi, il est possible de mesurer la vulnérabilité d'un algorithme en calculant le rapport entre la complexité par force

brute et la complexité en exploitant une faille ou une cryptanalyse particulière [17] :

$$V_{alg} = log_2 \frac{T_{fb}}{T_{cp}} \tag{2.1}$$

avec

— T_{bf} la complexité de l'attaque par force brute,

— T_{cp} la meilleure attaque de complexité inférieure ou égale au test par force brute.

Il se peut que la force brute soit la méthode la plus efficace, c'est à dire que $T_{cp} > T_{bf}$. Autrement dit, la cryptanalyse proposée n'apporte pas d'amélioration par rapport à l'attaque par force brute. Si aucune cryptanalyse ne propose d'amélioration par rapport à l'attaque par force brute, l'algorithme aura alors une vulnérabilité de zéro.

Cette méthode a le mérite de définir une métrique spécifique à chaque algorithme représentant la vulnérabilité. Mais un algorithme A1 avec une vulnérabilité de 20 peut être plus efficace qu'un algorithme A2 de vulnérabilité 0 si la complexité de cassage de A1, malgré la vulnérabilité, reste plus importante que la complexité de cassage par force brute de A2. En effet, contrairement à [4] et [17] qui font varier les paramètres au sein d'un même algorithme afin d'estimer sa vulnérabilité, la problématique posée ici implique de comparer différents algorithmes entre eux.

L'unité de mesure V_{alg} reste donc utile pour estimer un certain niveau de cassage d'un algorithme. Toutefois, son utilisation pour comparer deux algorithmes entre eux reste limitée. Elle sera donnée à titre indicatif dans la suite de ce chapitre mais c'est le temps de cassage global qui sera intéressant pour définir le niveau de sécurité souhaitée.

De plus, seul le coût temporel est considéré ici (avec la complexité algorithmique). Cela permet d'estimer la durée d'attaque de l'adversaire mais ce dernier devra probablement s'équiper d'espace mémoire conséquent afin de faire face à une consommation de mémoire très importante pour l'attaque en question.

Une revue des algorithmes usuels est proposée ci-après. Ces algorithmes ont été choisis car ils sont implémentés sur de nombreux supports, dont embarqués, et proposés sous forme de librairies. Le choix d'utiliser des librairies réalisées et optimisées pour le support cible est double. Cela permet, d'une part, d'avoir des performances optimisées, et d'autre part, de s'assurer du recul et de la revue d'une communauté plus grande qu'un algorithme fait individuellement. Ce dernier point est d'autant plus vrai dans le cas de l'*open-source*, contrairement à un algorithme implémenté soi-même.

2.2.2 Intégrité et algorithmes de hachage

Une fonction de hachage permet d'associer à une donnée de taille variable une empreinte [15] de taille fixe. Cette empreinte permet d'identifier une donnée et est notamment utilisée pour la gestion des mots de passe. En effet, dans les bonnes pratiques du web d'aujourd'hui, il est fortement déconseillé de stocker un mot de passe en clair dans une base de données. À la place du mot de passe, c'est l'empreinte de celui-ci qui est stockée.

Lorsqu'un utilisateur renseigne son mot de passe, son empreinte est calculée, envoyée au serveur, puis comparée à l'empreinte stockée en base de données. Un attaquant qui intercepte l'empreinte ou bien qui récupère l'empreinte stockée ne pourra pas remonter à la donnée d'origine : le mot de passe de l'utilisateur [63].

Cette dernière phrase est vraie si la fonction de hachage respecte les conditions définies ci-après. Dans ce cas, la fonction de hachage est cryptographiquement sécurisée.

En considérant la fonction de hachage H d'un message m donnant une empreinte y, l'équation $H(m) = y$ est la représentation mathématique de l'application d'une fonction de hachage, et cette fonction doit :

1. Résister à la pré-image, c'est-à-dire qu'en connaissant y, il doit être difficile de trouver m.
2. Résister à la seconde pré-image, c'est-à-dire qu'il doit être difficile, en connaissant m de trouver $m' \neq m$ tel que $H(m) = H(m')$.
3. Résister à la collision, c'est-à-dire qu'il doit être difficile de trouver un m et un m' tels que $H(m) = H(m')$.

Cependant, les collisions sont inévitables puisque l'espace des images est plus petit que l'espace des messages (de taille variable et potentiellement illimitée). Respecter ces conditions impose à l'espace d'images d'être assez grand pour ne pas rendre les collisions trop facilement exploitables. Il s'agit du principe de Dirichlet, vulgarisé en *principe des tiroirs* [16] : si n chaussettes sont réparties dans m tiroirs avec $n > m$, alors au moins un tiroir doit contenir plus d'une chaussette. Plus formellement, cela veut dire que pour deux ensembles finis E et F avec $card(E) > card(F)$, et une application $f : E \mapsto F$, alors il y aura forcément au moins un élément de F qui admet au moins deux antécédents dans E.

15. Le mot empreinte ici est synonyme d'image ou encore de *hash*.
16. *Pigeonhole principle*, ou principe des pigeons pour nos amis anglophones, qui, n'en doutons pas, ont sûrement une justification pointue qui explique la préférence de ces oiseaux urbains à nos tiroirs à chaussettes afin d'aborder au mieux ce problème mathématique.

L'attaque des anniversaires

Une bonne fonction de hachage devra donc posséder un espace d'empreintes assez grand pour limiter au mieux les collisions et la fonction de hachage elle-même doit rendre difficile la possibilité de trouver deux pré-images. Comment déterminer si une fonction de hachage est suffisamment robuste aux collisions ? Tout comme l'attaque par force brute énoncée précédemment, une attaque similaire permet de quantifier la complexité de base nécessaire pour trouver une collision et casser l'algorithme, et cette attaque se base sur le paradoxe des anniversaires.

Ce paradoxe considère la date d'anniversaire d'un groupe de k personnes (ce qui correspondrait à k messages de l'ensemble d'origine). La date d'anniversaire représenterait ainsi l'ensemble des empreintes possibles, soit 365 jours possibles [17].

La notion de paradoxe intervient ici car il est tentant de penser que la probabilité que deux personnes possèdent la même date d'anniversaire (ou que deux messages aient la même empreinte) est faible. Or, mathématiquement, il est aisé de démontrer que cette probabilité est importante (P_{seuil}) dés qu'un nombre de personnes (ou un nombre de messages) k_{seuil} donné est atteint.

Démonstration :

1. Supposons les dates d'anniversaire purement aléatoires et uniformément distribuées (les empreintes associées aux messages sont aléatoires et uniformément distribuées).
2. Soit p la probabilité que 2 personnes aient le même anniversaire (que deux messages aient la même empreinte). $p = 1/N$. Avec $N = 365$ (nombre de jours ou d'empreintes possibles).
3. Soit P_{diff_2} la probabilité que 2 personnes aient une date d'anniversaire différente. $P_{diff_2} = 1 - p = 1 - 1/N$,
4. De même, P_{diff_k} est la probabilité que les dates d'anniversaire de chaque personne (les empreintes de chaque message) $a_i, 1 \leq i \leq k$, soient différentes.
5. Enfin, soit $Q = 1 - P_{diff_k}$ la probabilité d'avoir deux anniversaires identiques parmi les k personnes (la probabilité d'avoir une collision parmi les k messages).

Démontrer ce paradoxe revient à se poser la question suivante : à partir de quel seuil k_{seuil} la probabilité d'avoir deux anniversaires identiques est supérieur à une probabilité P_{seuil} choisie ?

1. $k = 2$, $P_{diff_2} = 1 - \frac{1}{N} = \frac{N-1}{N}$

17. Les années bissextiles sont ignorées pour simplifier la comparaison.

2. $k = 3$, $P_{diff_3} = \frac{N-1}{N}\frac{N-2}{N}$
3. ...
4. $P_{diff_k} = \frac{N-1}{N}\frac{N-2}{N}...\frac{N-(k-1)}{N} = \prod_{i=1}^{k-1}(1 - \frac{i}{N})$

Or, $1 + x \leq e^x, \forall x \in \mathbf{R}$.

Donc $P_{diff_k} = \prod_{i=1}^{k-1}(1 - \frac{i}{N}) \leq \prod_{i=1}^{k-1} e^{-\frac{i}{N}} = e^{-\sum_{i=1}^{k-1}\frac{i}{N}} = e^{-\frac{k(k-1)}{2N}}$.

Donc, $Q = 1 - P_{diff_k} = 1 - \prod_{i=1}^{k-1}(1 - \frac{i}{N}) \geq 1 - e^{-\frac{k(k-1)}{2N}}$

L'objectif est maintenant de chercher pour un P_{seuil} donné, avec $0 < P_{seuil} < 1$, k tel quel $1 - e^{-\frac{k(k-1)}{2N}} > P_{seuil}$, ce qui implique :

$$\begin{aligned}
& Q > P_{seuil} \\
\iff \quad & 1 - e^{-\frac{k(k-1)}{2N}} > P_{seuil} \\
\iff \quad & -e^{-\frac{k(k-1)}{2N}} > P_{seuil} - 1 \\
\iff \quad & e^{-\frac{k(k-1)}{2N}} < 1 - P_{seuil} \\
\iff \quad & -\frac{k(k-1)}{2N} < \ln(1 - P_{seuil}) \\
\iff \quad & \frac{k(k-1)}{2N} > \ln(\frac{1}{1-P_{seuil}}) \\
\iff \quad & k(k-1) > 2N\ln(\frac{1}{1-P_{seuil}})
\end{aligned}$$

Or $k^2 > k(k-1)$, donc

$$\begin{aligned}
& k^2 > 2N\ln(\frac{1}{1-P_{seuil}}) \\
\iff \quad & k > \sqrt{2N\ln(\frac{1}{1-P_{seuil}})}
\end{aligned}$$

Ainsi, pour avoir une probabilité $P_{seuil} > 0.5$ d'avoir deux anniversaires identiques (une collision) avec $N = 365$, il faut $k = 23$ personnes. À partir de $k = 58$, la probabilité d'avoir une collision est $P_{seuil} > 0.99$.

Cette attaque des anniversaires réduit la complexité pour casser les algorithmes de hachage à $\mathcal{O}(\sqrt{N})$ avec N le nombre d'empreintes possibles. Si N est codé sur n bits, la complexité est de l'ordre de $\mathcal{O}(2^{n/2})$.

Le stockage sécurisé des mots de passe est une application concrète qui doit respecter ces conditions. Cependant, une autre utilisation des fonctions de hachage respectant ces conditions est le contrôle d'intégrité.

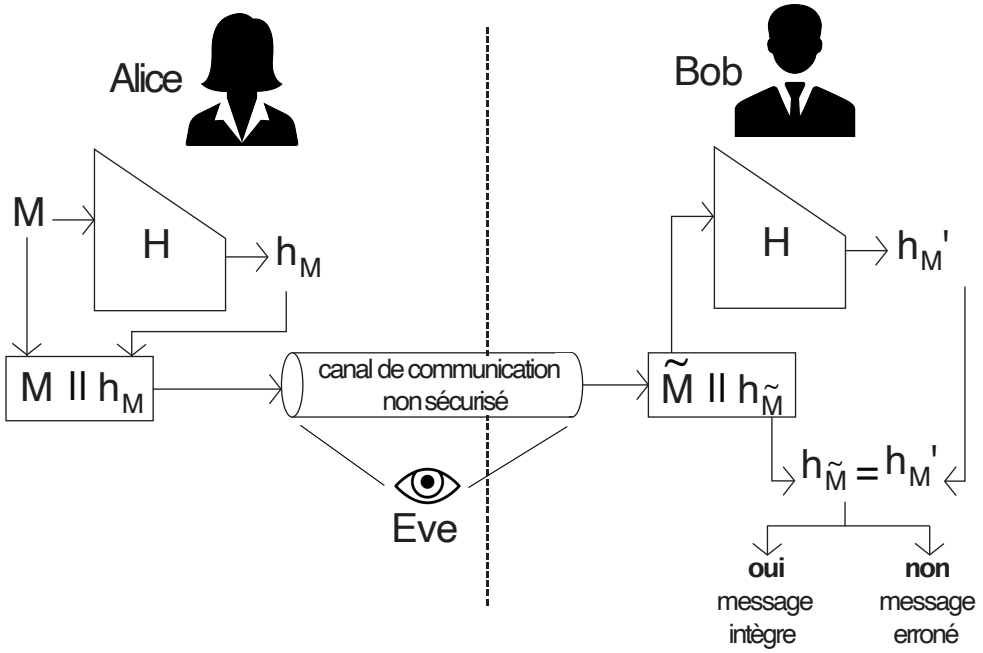

Fig. 2.4.: Contrôle d'intégrité

Si Alice souhaite envoyer un message M, elle va calculer son empreinte $h_M = H(M)$. Elle envoie ensuite $M||h_M$ à Bob [18]. Bob, à la réception, reçoit $\widetilde{M}||h_{\widetilde{M}}$, c'est-à-dire un message potentiellement modifié.

Pour s'assurer de l'intégrité du message, Bob extrait $\widetilde{M}$ et calcule $h'_M = H(\widetilde{M})$. Si $h'_M = h_{\widetilde{M}}$. Alors Bob peut supposer qu'il a reçut un message non altéré [19] puisqu'il correspond bien à l'empreinte reçue $h_{\widetilde{M}}$. La figure 2.4 schématise l'usage général des fonctions de hachage.

Afin de pouvoir gérer des messages de taille variable, la construction de Merkle-Damgård peut être utilisée. Cette dernière, à partir d'une fonction H recevant en données d'entrées des données de taille fixe, permet de calculer une empreinte de taille fixe. Pour ce faire, la construction découpe le message d'entrée en plusieurs blocs de taille fixe (avec un bourrage éventuel en fin de message). L'enchaînement des blocs peut suivre différents schémas. La figure 2.5 donne un exemple de construction de Merkle-Damgård avec le schéma Davies-Meyer. Le schéma Davies-Mayer permet de construire une fonction de hachage à partir d'une fonction de chiffrement E.

18. Ici || représente la concaténation.

19. Plus précisément, Bob a une grande probabilité d'obtenir un message non altéré. Il existe toujours une probabilité (infinitésimale) que le message M ait été modifié en $\widetilde{M}$ et que l'image de $\widetilde{M}$, $h_{\widetilde{M}}$, soit égale à h'_M ; autrement dit, que M et $\widetilde{M}$ forment une collision. D'où l'importance du choix de la fonction de hachage.

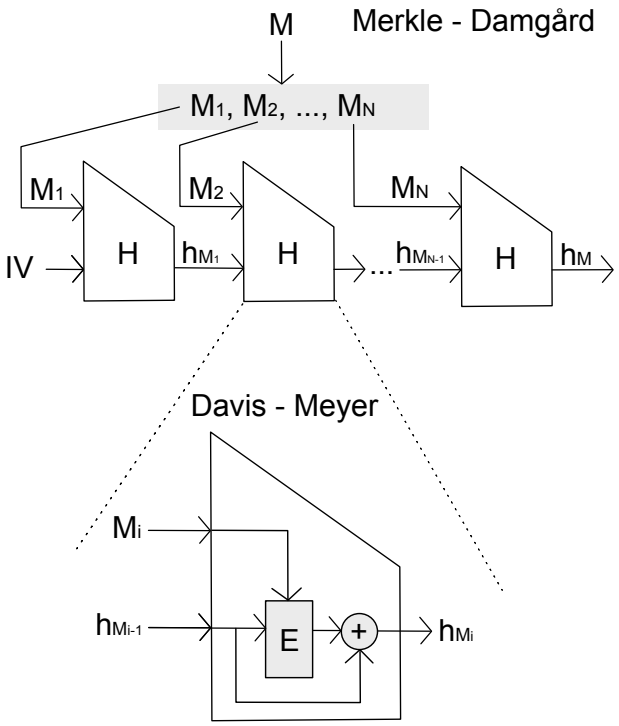

Fig. 2.5.: Construction Merkle-Damgård avec le schéma Davies-Meyer

Mais la fonction H peut représenter aussi un algorithme de hachage construit de toute pièce, comme par exemple MD5, SHA1 et la famille SHA2 (SHA256, SHA312 et SHA512). Les fonctions de type HMAC, proposent également du contrôle d'intégrité, mais vont plus loin en y ajoutant l'authentification.

MD5

L'algorithme MD5 a été proposé par R. Rivest dans la RFC[20] 1321 [44]. Cette RFC définit MD5 comme le successeur de MD4 qui possédait des failles déjà connues à l'époque. Pour combler ces failles, MD5 implémente un algorithme plus robuste mais plus lent que MD4. Cette perte de performance se faisant au profit d'une sécurité bien plus robuste, elle ne dérangea pas les utilisateurs et n'empêcha donc pas son déploiement à grande échelle, notamment à des fins de signatures comme la RFC l'indique.

Les données d'entrée de MD5 sont découpées en blocs de 512 bits, le dernier faisant $512 - 64 = 448$ bits, permettant de stocker sur les 64 derniers bits la taille des données hachées. Un bourrage (consistant en un unique bit "1" suivi d'autant de bits "0" que nécessaire) est ajouté au message pour arriver à un dernier bloc de

20. Les RFC sont des documents techniques rédigés par des personnes afin de proposer des normes détaillées autour des technologies d'Internet. Ces documents sont relus et commentés par toute une communauté, et certaines institutions comme l'IETF (par exemple) en font des normes.

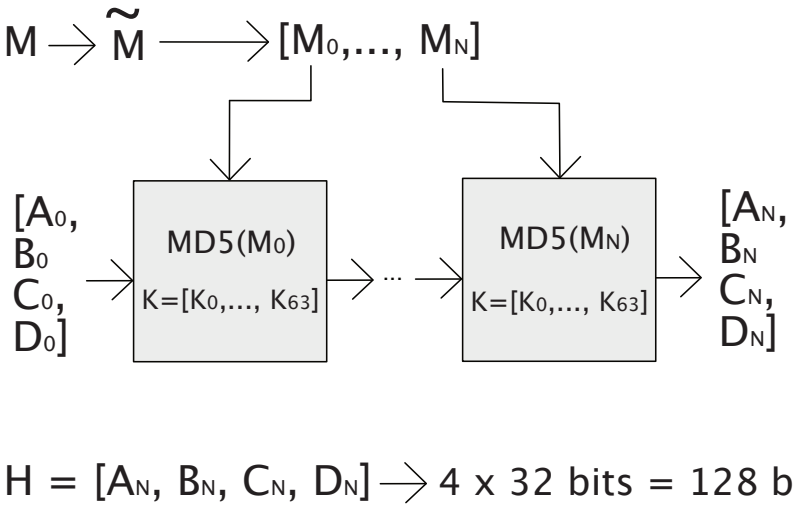

Fig. 2.6.: Entrées et sorties de MD5

taille égale à 448 bits. La sortie finale $[A_N, B_N, C_N, D_N]$ sur la figure 2.6 représente l'empreinte finale pour une taille totale d'empreinte de 128 bits.

Pour arriver à ce résultat, chaque bloc du message est envoyé dans le coeur de l'algorithme représenté sur la figure 2.7. Le bloc sera *mélangé* avec des constantes propres à MD5, indépendantes du message à hacher. Chaque bloc peut être vu comme 16 mots de 32 bits ($16 * 32 = 512$ bits). Ces mots sont utilisés dans le coeur de l'algorithme pour calculer l'empreinte finale après une succession de 64 rondes. La fonction F change 4 fois (une fois toutes les 16 rondes) en passant par F_1, F_2, F_3 et F_4 qui valent respectivement :

1. $F_1 = (B \wedge C) \vee ((\neg B) \wedge D), \forall 0 \leq r < 16$
2. $F_2 = (D \wedge B) \vee ((\neg D) \wedge C), \forall 16 \leq r < 32$
3. $F_3 = B \oplus C \oplus D, \forall 32 \leq r < 48$
4. $F_4 = C \oplus (B \vee (\neg D)), \forall 48 \leq r < 64$

où r représente le numéro de ronde.

Comme pour MD4, MD5 a été soumis à l'épreuve du temps et à l'augmentation de la puissance de calcul des ordinateurs ainsi qu'à la découverte de failles et de raccourcis pour trouver des collisions. Une estimation de la complexité algorithmique pour trouver des collisions est de 2^{33} [30].

En appliquant l'équation 2.1, l'indice de vulnérabilité est alors de :

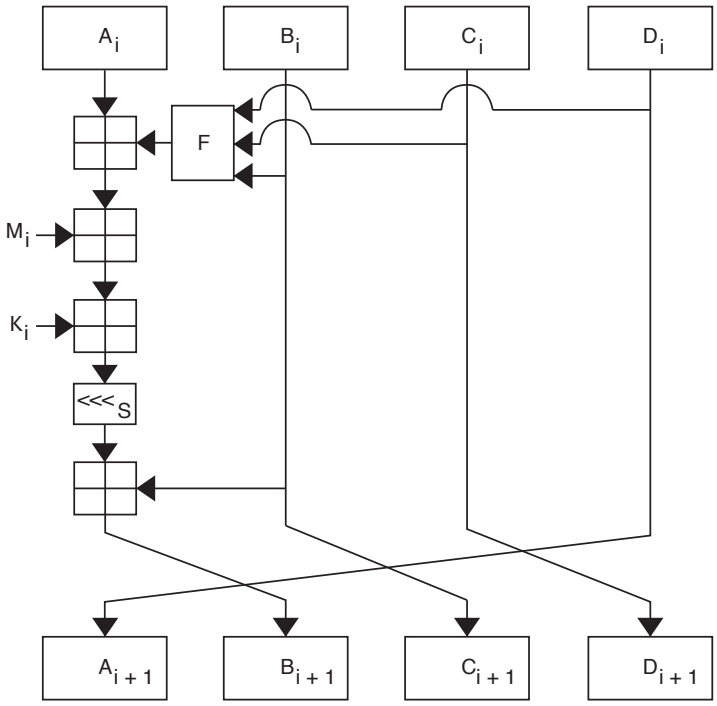

Fig. 2.7.: Coeur de l'algorithme MD5

$$V_{MD5} = log_2 \frac{T_{fb}}{T_{cp}} = log_2 \frac{2^{64}}{2^{33}} = 31$$

Bien que la RFC 1321 (datant de 1992) indique que "*le niveau de sécurité présent dans [cette RFC] est considéré comme suffisant pour la mise en oeuvre de schémas de signatures électroniques hybrides basés sur un système à clé publiques et sur MD5*", aujourd'hui son utilisation n'est plus recommandée, sauf pour la détection d'erreurs non intentionnelles (similaire donc à un checksum classique)[33]. Pour ce dernier cas (souvent couplé à de l'authentification et à du chiffrement), il reste présent dans de nombreux systèmes et c'est pourquoi il a été intégré dans cette étude malgré sa vulnérabilité.

Il est à noté également que différentes méthodes d'implémentation de MD5, comme l'intégration d'un élément supplémentaire (appelé *salt*), permettent de se protéger des attaques par dictionnaires [21]. Cet élément rend les collisions plus complexes à deviner car il faudrait connaître le *salt* pour générer le message à hacher.

21. Cette attaque consiste à se référer à une table de correspondance message - empreinte déjà existante car pré-calculée.

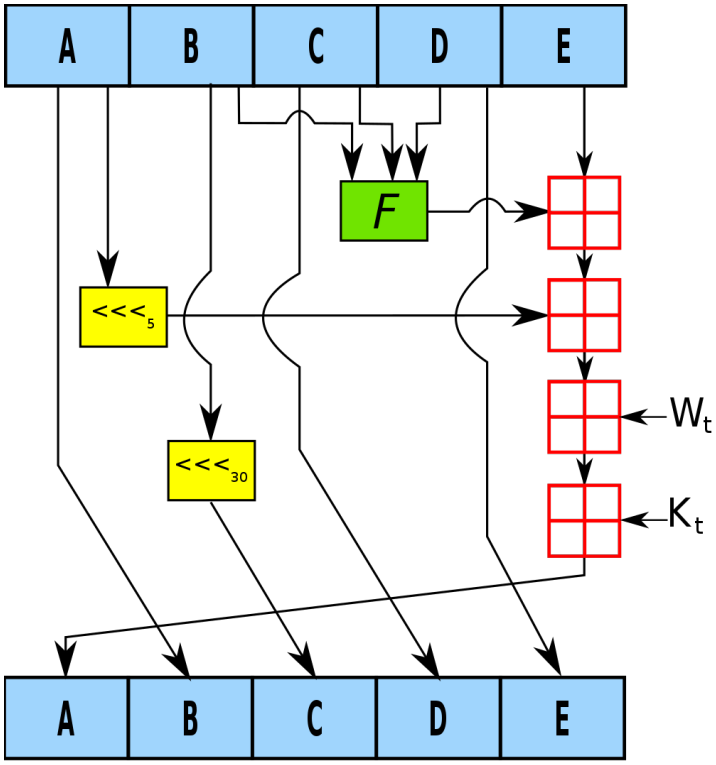

Fig. 2.8.: Coeur de l'algorithme SHA-1

Source : https ://en.wikipedia.org/wiki/SHA-1

SHA-1

Conçu par la NSA (*National Security Agency*) en 1995, SHA-1 (pour *Secure Hash Algorithm*) produit une empreinte de 160 bits (c'est-à-dire 20 octets) avec des principes similaires à ceux de MD5.

Comme pour MD5, l'algorithme de SHA-1 traite des messages divisés en blocs de 512 bits, avec le dernier bloc en particulier de 448 bits pour y ajouter la longueur du message (représentée sur 64 bits). Le principe du bourrage est là aussi le même afin d'arriver à ces blocs. La logique sous-jacente est encore une fois celle de Merkle-Damgård illustrée dans la figure 2.5.

Le coeur de SHA-1, représenté sur la figure 2.8, utilise cette fois 80 rondes, avec, une fois encore, des fonctions spécifiques par groupes de rondes :

1. $F_1 = (B \wedge C) \vee ((\neg B) \wedge D), \forall 0 \leq r < 20$
2. $F_2 = B \oplus C \oplus D, \forall 20 \leq r < 40$
3. $F_3 = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D), \forall 40 \leq r < 60$
4. $F_4 = B \oplus C \oplus D, \forall 60 \leq r < 80$

où r représente le numéro de ronde.

En 2005, SHA-1 a été cassé avec une complexité algorithmique de 2^{69}, c'est-à-dire une complexité algorithmique inférieure à 2^{80} ($= 2^{\frac{160}{2}}$) correspondant à une attaque des anniversaires classiques [57]. Plus récemment, en 2013, Marc Stevens a fourni une méthode pour trouver une collision avec une complexité de 2^{61} tout en laissant des pistes pour baisser encore cette complexité dans les années à venir [50]. En appliquant l'équation 2.1, la vulnérabilité de SHA-1 est alors de :

$$V_{SHA1} = log_2 \frac{T_{fb}}{T_{cp}} = log_2 \frac{2^{80}}{2^{61}} = 19$$

Tout comme pour MD5, SHA-1 peut se prémunir de ces collisions en étant implémenté convenablement, avec un *salt* par exemple. SHA-1 est longtemps resté un standard utilisé dans les signatures électroniques, notamment dans les certificats SSL. Néanmoins, en 2017, les navigateurs web les plus importants cessèrent de les accepter.

Cette décision a été plus simplement acceptée à grande échelle après l'annonce de Google d'avoir calculé la première collision concrète, en collaboration avec une équipe de chercheurs du CWI [22], dont Marc Stevens. Ce *calcul polémique* [23], sans enterrer définitivement SHA-1, aura au moins permis de montrer qu'une vulnérabilité ne rend pas un algorithme non sécurisé au sens où la sécurité ne peut être dans tous les cas absolue. Au contraire, cet *exploit* montre que seule une puissance étatique ou une méga-corporation possède les moyens aujourd'hui d'exploiter ce genre de vulnérabilités [32]. Dans tous les cas, la migration vers SHA-2 était lancée.

SHA-2 (et un peu de SHA-3)

En 2002, SHA-2 (*Secure Hash Algorithm 2*) a également été conçu par la NSA justement pour succéder à SHA-1 en proposant plusieurs fonctions de hachage de différentes tailles d'empreinte : SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 et enfin SHA-512/256.

SHA-256 et SHA-512 sont les deux variantes de SHA-2 qui seront abordées dans ce document. De manière générale, SHA-2 diffère de SHA-1 par les fonctions uti-

22. *Centrum voor Wiskunde en Informatica* ou Centre pour les mathématiques et l'informatique, situé à Amsterdam aux Pays-Bas.

23. Ce terme est repris de [32] qui définit *polemic computation* comme un calcul à l'utilité discutable au regard des ressources engagées pour calculer pratiquement un résultat prouvé théoriquement sans considération pour les impacts environnementaux, sociétaux ou économiques. Il cite à titre d'exemple l'*exploit* de Google qui a effectué des calculs en parallèle sur des super-calculateurs pendant plusieurs jours pour aboutir à cette première collision de SHA-1. L'auteur s'interroge sur les intérêts scientifiques de Google dans cette démarche par rapport à ses intérêts économiques pour convaincre les acteurs d'Internet de se passer de SHA-1.

lisées, le nombre de rondes et les opérations bits à bits réalisées dans le coeur de l'algorithme.

SHA-512 propose une empreinte 2 fois plus grande que SHA-256 (512 bits contre 256 bits) mais occupe plus de mémoire lors des opérations. En effet, les opérations ont lieu sur des mots de 64 bits contre des mots de 32 bits, pour un bloc total de 1024 bits contre 512 bits pour SHA-256.

Si la famille SHA-2 n'a pas aujourd'hui d'attaques par collision plus efficaces que l'attaque des anniversaires, un nouveau type d'attaque est potentiellement exploitable, lié au schéma de construction Merkle-Damgård : l'attaque par extension de longueur.

L'attaque par extension de longueur consiste à concaténer au message actuel un dernier bloc lors de la dernière opération de Merkle-Damgård, permettant de forger un nouveau message valide.

Cette attaque nécessite de connaître exactement le taille du message (et du *salt* éventuellement utilisé). Il s'agit davantage d'un défaut de schéma et d'implémentation que d'une vulnérabilité invoquant un calcul algorithmique complexe. Cette attaque ne sera donc pas pris en compte dans la suite de ce document et les complexités d'attaque retenue pour SHA-256 et SHA-512 seront donc respectivement 2^{128} et 2^{256}. L'attaque par extension de longueur étant plutôt conceptuelle aujourd'hui, cela signifie que SHA-2 n'a pas d'attaques connues à la date où ce document est rédigé.

Toutefois, une solution à ce défaut a été proposée : changer le schéma de construction. La proposition a été concrétisée et implémentée dans SHA-3 en 2015 : les fonctions éponges. Ces fonctions permettent, dans une première phase, d'absorber un bloc à chaque itération puis, dans une seconde phase, d'extraire un morceau d'empreinte à chaque itération. Chaque morceau extrait correspondra à une partie de l'empreinte finale, et le nombre d'itérations de la deuxième phase permet donc de contrôler la taille de l'empreinte finale.

Il est possible, pour SHA-3, de choisir un paramètre correspondant au nombre de bits à absorber par itération (noté r). Le nombre minimum de bits pour un bloc (le nombre de bits à absorber) est de 576 bits, légèrement plus important que SHA-256 (512 bits par bloc) mais beaucoup moins important que pour les blocs de SHA-512 (1024 bits par blocs). Selon les valeurs de r (1152, 1088, 832, ou 576 bits), la taille de l'empreinte change (224, 256, 384 et 512 bits).

SHA-3 a été proposé comme alternative aux algorithmes de hachage usuels en vue de se prémunir de l'attaque par extension de longueur. Or, aucune exploitation concrète

de cette faille n'est connue à ce jour, faisant de SHA-2 un algorithme de hachage considéré comme sûr.

2.2.3 Confidentialité et algorithmes de chiffrement

Il existe deux grandes familles d'algorithmes de chiffrement qui se distinguent par l'utilisation de clés symétriques ou asymétriques.

L'utilisation de clés symétriques signifie que la clé de chiffrement Ks_c et la clé de déchiffrement Ks_d sont identiques ou bien très facilement trouvables à partir de la première. Au contraire, l'utilisation de clés asymétriques signifie que la clé de chiffrement Ka_c et la clé de déchiffrement Ka_d sont différentes mais fortement liées par des propriétés mathématiques dépendantes de l'algorithme lui-même.

Dans un premier temps seront abordés les algorithmes de chiffrement symétrique.

DES

DES (*Data Encryption Standard*) est un algorithme de chiffrement symétrique par bloc proposé comme standard dés 1977, et c'est sa version 56/64 bits [24] qui sera finalement utilisée.

DES se base sur le principe des tournées de Feistel [25]. Les tournées de Feistel reposent sur N sous-clés K_0, K_1, ..., K_{N-1} générées à partir de la clé K de 64 bits et sur des blocs de 64 bits. Ces blocs, découpés en deux sous-blocs de 32 bits, sont utilisés en données d'entrées de chaque ronde. Enfin, les tournées de Feistel utilisent une fonction F appliquée sur la moitié d'un bloc à chaque itération.

DES implémente donc 16 itérations de tournées de Feistel avec une opération initiale et une permutation finale, comme récapitulé sur la figure 2.9.

Enfin, la fonction F utilisée par DES repose sur une série de 8 blocs de substitution (*S-boxes*) appliqué au résultat d'un XOR entre la moitié d'un bloc et une sous-clé. Une fois les *S-boxes* appliquées, une permutation est appliquée. La fonction F est représentée sur la figure 2.10.

24. La version 64 bits de DES utilise une clé de 56 bits auxquels sont ajoutés pour chaque octet un bit de parité, ce qui donne un total de 56 + 8 = 64 bits mais seulement 56 bits effectifs. À noter que l'algorithme initial conçu par IBM utilisait des clés de 112 bits mais la NSA a ramené la taille à 56 bits lors de la parution de DES en tant que standard.

25. également appelé réseau de Feistel nommé d'après son inventeur Horst Feistel, cryptologue chez IBM

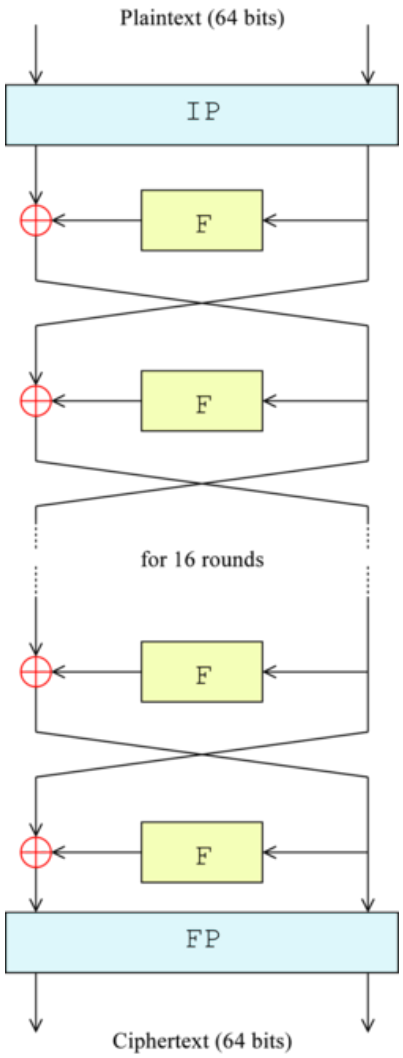

Fig. 2.9.: Tournées de Feistel dans DES

Source : https ://en.wikipedia.org/wiki/Data_Encryption_Standard

De part la taille effective des clés utilisées dans DES, l'attaque par force brute est d'une complexité de 2^{56}. L'attaque la plus efficace en terme de complexité est celle évaluée par [20] en 2^{41} et se base sur l'attaque trouvée par [31] en 1994. Cependant, cette complexité temporelle s'accompagne d'une complexité en mémoire assez grande car de nombreux textes chiffrés et clairs doivent être sauvegardés (2^{43}).

3DES

Face aux faiblesses évoquées précédemment, un nouveau standard basé sur DES émergea en 1995. Ce standard est 3DES (pour Triple DES) [22] qui, comme le laisse entendre le nom, triple la sécurité en utilisant une clé K de 168 bits. En fait, cette

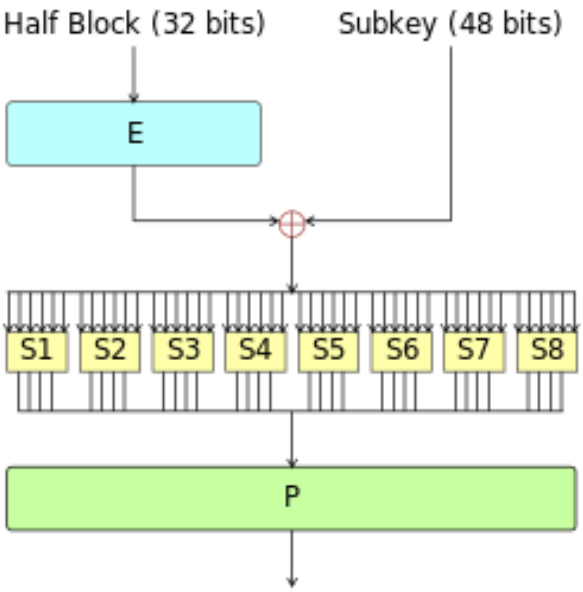

Fig. 2.10.: Fonction F DES dans les tournées de Feistel

Source : https ://en.wikipedia.org/wiki/Data_Encryption_Standard

clé K peut être décomposée en trois sous-clés K_1, K_2 et K_3 utilisées respectivement par les trois parties de 3DES.

En effet, 3DES opère trois opérations sur un message M à chiffrer :

1. Chiffrement DES de M avec K_1, ce qui donne C_1,
2. Opération de déchiffrement DES de C_1 **avec** K_2, ce qui donne C_2
3. Chiffrement DES de C_2 avec K_3, ce qui donne C

Résumé en une équation, 3DES donne : $C = E_{K_3}(D_{K_2}(E_{K_1}(M)))$.

Initialement, 3DES permettait de jouer sur les sous-clés K_1, K_2 et K_3 grâce à trois options :

1. Option 1 : les trois clés sont indépendantes, donnant une attaque par force brute de 2^{168}.
2. Option 2 : $K_1 = K_3$ et K_2 est indépendante, ce qui donne une attaque par force brute de 2^{112}.
3. Option 3 : les trois clés sont identiques, ce qui permet une rétro-compatibilité entre 3DES et DES.

Les options 2 et 3 étant aujourd'hui obsolètes, seule l'option 1 est utilisée. Cependant, les attaques de type *Meet-in-the-middle* réduisent la complexité temporelle à 2^{112} mais les coûts matériels pour casser 3DES restent très importants [39].

AES

Pour succéder à DES, le NIST sélectionna un nouvel algorithme de chiffrement pour son tout nouveau *Advanced Encryption Standard* (AES) en 2001 : l'algorithme de Rijndael. En particulier, AES-128, AES-192 et AES-256 furent sélectionnés. Il s'agit des versions acceptant respectivement des clés de 128, 192 et 256 bits pour des blocs de 128 bits.

L'algorithme AES peut être décomposé en 4 parties principales :

1. Expansion de clé : à partir de la clé d'origine, des sous-clés sont générées.
2. Étape initiale de XOR de la clé avec les octets du bloc.
3. Selon la taille de la clé (128, 192 ou 256 bits), enchaîner 9, 11 ou 13 rondes des fonctions suivantes :
   — Substitution des octets du bloc grâce à une table de substitution.
   — Permutation circulaire des lignes d'un bloc.
   — Mélange des bits d'une colonne.
   — XOR avec la sous-clé courante.
4. Une ronde finale composée uniquement des fonctions suivantes :
   — Substitution des octets du bloc grâce à une table de substitution.
   — Permutation circulaire des lignes d'un bloc.
   — XOR avec la sous-clé courante.

Les attaques par force brute sur AES dépendent de la taille de la clé utilisée (et donc du nombre de rondes utilisées dans l'algorithme). AES-128 possède donc une complexité d'attaque par force brute de 2^{128}, AES-192 une complexité de 2^{192} et enfin AES-256 une complexité de 2^{256}.

Les attaques proposées par [52] proposent une complexité de cassage en 2^{126} (pour AES-128), $2^{189.9}$ (pour AES-192) et $2^{254.3}$ (pour AES-256) et donc des facteurs de vulnérabilités proches de zéro.

RSA

RSA est un algorithme de chiffrement asymétrique (également dit à clé publique) créé par Ron Rivest, Adi Shamir et Léonard Adleman (d'où le nom R-S-A de l'algorithme) en 1977. Contrairement aux autres algorithmes vus jusqu'alors, RSA utilise donc deux clés : l'une pour le chiffrement (publique, notée K_c) et l'autre pour le déchiffrement (privée notée K_d).

L'intérêt d'un tel système est qu'il est mathématiquement complexe de trouver K_d en connaissant K_c. Cela est utile si K_c est connue publiquement : chacun peut alors utiliser K_c pour chiffrer ses messages, et seul le propriétaire de K_d pourra les déchiffrer, comme schématisé sur la figure 2.11. Pour RSA en particulier, trouver K_d à l'aide de K_c revient à résoudre le problème de factorisation des nombres premiers.

Génération des clés par Alice (qui souhaite que Bob puisse lui envoyer un texte chiffré) :

1. Choix de p et q, deux nombres premiers.
2. Calcul de $n = p * q$ et de $\Phi(n) = (p-1) * (q-1)$ avec Φ la fonction indicatrice d'Euler.
3. Choix de K_c, nombre premier avec $(p-1)$ et $(q-1)$.
4. Calcul de K_d $(= K_c^{-1})$, inverse de K_c dans $\mathbb{Z}/\Phi(n)\mathbb{Z}$.

Chiffrement RSA :
Le chiffrement RSA d'un message M par Bob (qui connaît K_c et n) s'effectue en appliquant la formule 2.2.

$$C = M^{K_c} \mod n \tag{2.2}$$

Déchiffrement RSA :
Le déchiffrement RSA d'un message chiffré C par Alice (qui connaît K_d et n) s'effectue en appliquant la formule 2.3.

$$M = C^{K_d} \mod n \tag{2.3}$$

RSA étant déterministe (c'est-à-dire qu'il ne dépend pas d'éléments aléatoires), l'algorithme est vulnérable à certaines attaques, notamment l'attaque par rejeu. C'est pourquoi le standard d'implémentation OAEP (introduction d'aléa et de bourrage) et le standard PKCS#1 (optimisation lors du déchiffrement) sont recommandés pour une utilisation sûre et optimisée de l'algorithme RSA [19].

Aujourd'hui, RSA est utilisé principalement avec un mode 1024 et 2048 bits. donnant ainsi des complexités de l'ordre de $2^{1024/2} = 2^{512}$ et $2^{2048/2} = 2^{1024}$. Le mode 4096 bits est également disponible, mais la lourdeur des calculs et l'espace mémoire nécessaire pour les réaliser [26] le rendent peu pratique voir impossible sur certains supports (systèmes embarqués, serveurs avec de nombreuses connexions simultanées...). En considérant l'algorithme correctement implémenté, les complexités d'attaques par force brute sur RSA sont de l'ordre de la taille de la clé divisée par 2 noté $\mathcal{O}(\sqrt{N})$

26. Par exemple, avec un exposant de 4096 bits, certains calculs intermédiaires nécessitent un stockage dans des variables de 8192 bits.

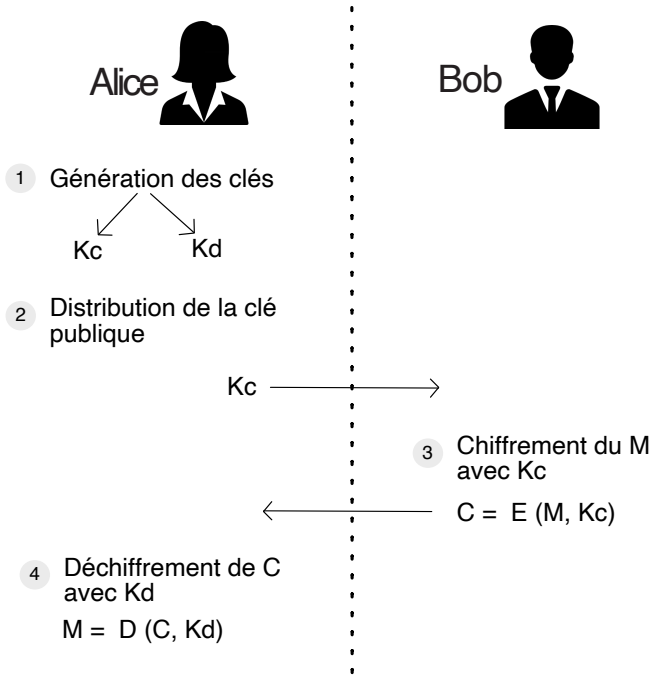

Fig. 2.11.: Chiffrement asymétrique

avec N la taille de la clé. Cependant, une meilleure attaque a été trouvée, reposant sur la factorisation de n. En effet, trouver K_d revient à calculer $K_c^{-1} \mod \Phi(n)$. $\Phi(n) = (p-1) * (q-1)$ donc en connaissant p et q, il est possible de recalculer plus facilement K_d.

Le meilleur algorithme de factorisation de nombres premiers possède une complexité de $\mathcal{O}(e^{(64/9)^{1/3} * ln(2^n)^{1/3} * ln(ln(2^n))^{2/3}})$ [36] (avec n la taille de la clé) et a été utilisé sur RSA-768 qui n'est plus recommandé aujourd'hui [26]. Les complexités d'attaques pour RSA-1024 et RSA-2048 sont donc respectivement de :

— $e^{(64/9)^{1/3}} * ln(2^1024)^{1/3} * ln(ln(2^1024))^{2/3} \approx 2^{87}$

— $e^{(64/9)^{1/3}} * ln(2^2048)^{1/3} * ln(ln(2^2048))^{2/3} \approx 2^{117}$

De même que l'attaque des anniversaires est l'attaque privilégiée pour casser les algorithmes de hachage (de part leur nature), l'attaque par factorisation de n est l'attaque privilégiée pour le système RSA et les résultats précédents serviront donc de référence pour les attaques RSA.

Un autre type d'attaque est possible mais volontairement omis de l'étude : les attaques par synchronisation. Ces dernières reposent sur l'analyse du temps utilisé par le processeur pour réaliser le calcul de l'équation 2.3. Cela peut donner des indices sur l'exposant lui-même (nombres de bits à 1 par exemple). Ce type d'attaques est omis car il implique un accès au processeur, ce qui est hors du champ de l'étude.

2.2.4 Authentification et algorithmes de signature

L'authentification implique des mécanismes mathématiques similaires à ceux déjà évoqués, mais le but est désormais de prouver l'authenticité d'un message. En particulier, il faut pouvoir certifier l'envoyeur du message et le message lui-même.

RSA

Le chiffrement RSA expliqué précédemment utilise un système de chiffrement à clé publique : K_c est utilisée publiquement pour chiffrer, et K_d est gardée privée pour le déchiffrement.

Dans le cas de l'authentification, les opérations de chiffrement et déchiffrement sont inversées. Ainsi K_c est gardé privée pour *chiffrer* un message et K_d est rendue publique pour que chacun puisse déchiffrer le message.

Si le message est correctement déchiffré avec K_d, seul le propriétaire de K_c a pu le chiffrer. L'authenticité de l'envoyeur est donc vérifiée. Pour vérifier si un message est correctement déchiffré, ce dernier est haché par un algorithme de hachage tel que ceux décrits précédemment. C'est l'ensemble message et empreinte qui est signé à l'aide de RSA. Afin de s'assurer de l'unicité de la signature, les standards OAEP et PKCS#1 doivent être implémentés également.

Les vulnérabilités sont donc les mêmes que pour le chiffrement RSA.

ECC

ECC, pour *Elliptic-curve cryptography* ou encore cryptographie basée sur des courbes elliptiques est un système asymétrique de chiffrement et de signatures. Il repose sur un problème mathématique différent de RSA (qui était la factorisation de nombres premiers) : le logarithme discret.

Le logarithme discret est déjà utilisé pour des algorithmes d'échanges de clés (comme Diffie-Hellman) ou pour des algorithmes de chiffrement et signatures comme El-Gamal mais nécessite un corps de nombre assez grand (et donc des clés et calculs intermédiaires sur des tailles relativement grandes). Toutefois, les mécanismes de ces exemples reposent sur le calcul modulaire (d'où le nom de logarithme discret).

Dans le cas d'ECC, le problème de logarithme discret doit être résolu dans un autre contexte mathématique avec une algèbre distincte : sur les courbes elliptiques munies

notamment d'une addition géométrique de points [27]. Les propriétés mathématiques d'ECC permettent d'utiliser des tailles de clés plus petites pour un niveau de sécurité équivalent à celui obtenu en utilisant RSA. Par exemple, pour une taille de clé de 256 bits, il faudrait une clé de 3072 bits pour RSA [13].

Tout comme RSA, ECC étant asymétrique peut être utilisé pour le chiffrement asymétrique ou pour la signature.

HMAC

Les HMAC pour *Keyed-Hash Message Authentication Code* sont des fonctions permettant d'associer les MAC (*Message Authentication Code*) aux fonctions de hachage comme celles citées précédemment.

Pour expliquer un MAC, le schéma 2.4 peut être repris et modifié pour y ajouter un secret préalablement partagé par les deux entités : la clé K. Cette clé est utilisée en entrée de la fonction de hachage avec le message. Ensuite, comme illustré sur la figure 2.12, seule l'empreinte et le message sont envoyés, la clé reste en possession de l'émetteur et du destinataire.

Un MAC peut être vu comme une application symétrique des signatures : en effet, la clé K est partagée au préalable par les deux entités communicantes.

HMAC est une forme de MAC dépendant d'une fonction de hachage (MD5, SHA-1,...). Le choix de cette fonction conditionne la taille du MAC qui sera égale à la taille de l'empreinte de la fonction de hachage choisie.

Comme expliqué dans la RFC 2104, la fonction de hachage est utilisée deux fois dans HMAC [28] :

$$h_K(m) = H(K \oplus o\_pad, H(K \oplus i\_pad, m)) \tag{2.4}$$

où *o_pad* et *i_pad* (pour respectivement *outer* et *inned*) sont deux constantes de taille B bits d'un bloc. Si la taille de K est inférieure à B, alors des zéros sont ajoutés à la suite de K pour compléter sa taille jusqu'à B.

27. Ces propriétés ne seront pas abordés ici. Pour avoir une idée globale cependant, il est possible de considérer ce problème mathématique comme deux points liés sur une courbe paramétrique par un facteur k. Les paramètres de la courbe et le point de référence G font parti de la clé publique tandis que le facteur k correspond à la clé publique.

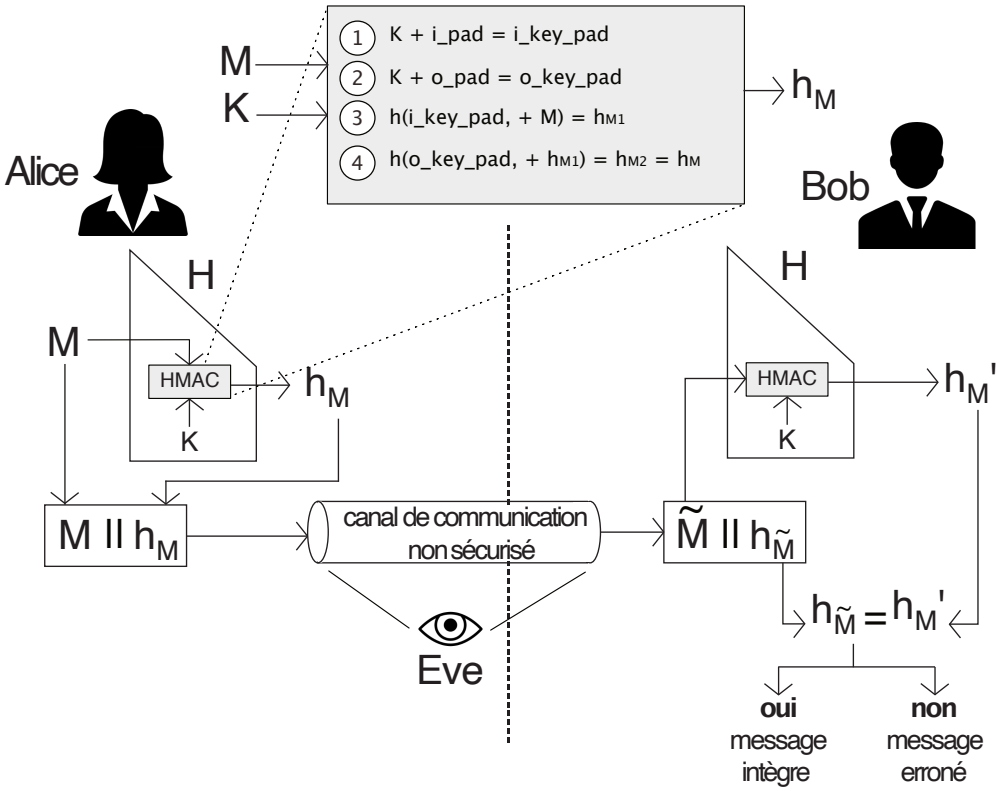

Fig. 2.12.: HMAC

Les vulnérabilités des HMAC dépendent donc de la fonction de hachage utilisée et sont récapitulées dans le tableau 2.2 ci-après.

2.2.5 Les modes d'opération

En reprenant l'exemple d'AES sur des messages assez conséquents, de longueur supérieure à 128 bits (la taille d'un bloc AES), les blocs sont chaînés afin de mieux *diffuser* le chiffrement. La manière dont sont chaînés les blocs correspond au mode d'opération. Certains modes, comme CCM, introduisent l'authentification en plus du chiffrement.

Les modes de chiffrement n'introduisent pas de vulnérabilité particulière et dépend donc de l'algorithme de chiffrement utilisé (AES-128, 192 ou 256 par exemple).

CBC

Le mode CBC chiffre un bloc grâce à l'algorithme de chiffrement associé. Le résultat est ensuite *xoré* avec le bloc suivant avant le chiffrement de ce dernier. Le premier bloc est *xoré* avec un vecteur d'initialisation.

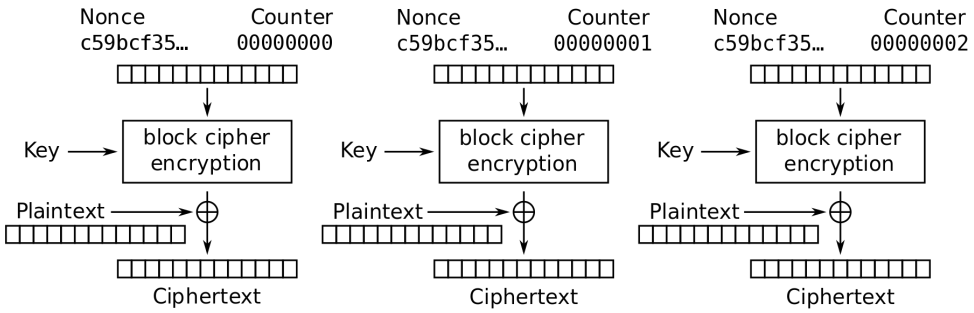

Fig. 2.13.: Chiffrement par bloc - Counter mode

Source : https ://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

CTR

Le mode CTR (pour *Counter mode*) chiffre un vecteur d'initialisation incrémenté dont le résultat est *xoré* au bloc clair 2.13.

Il est à noter que la figure 2.13 peut être vue comme un chiffrement de flux où le texte clair est *xoré* avec un bloc de la taille de ce texte, ce bloc étant généré à partir de la clé et du compteur.

CCM

Le mode CCM est un dérivé du mode CTR qui introduit un MAC pour authentifier la donnée en plus de la chiffrer. Ce mode peut nécessiter plus de ressources car deux opérations ont lieu ici (chiffrement et hachage).

2.2.6 Récapitulatif des algorithmes et de leur vulnérabilité

Le tableau 2.2 récapitule les vulnérabilités de chaque algorithme par familles algorithmiques. Les algorithmes dans la catégorie signature peuvent également être considérés comme des algorithmes de chiffrement et déchiffrement asymétriques.

Concernant ECC, une clé de 163 bits donne un équivalent de complexité de l'algorithme RSA sur une clé de 1024 bits [13].

Cette liste non exhaustive d'algorithmes permet donc, avec une certaine complexité algorithmique et une certaine vulnérabilité d'attaque, de protéger des messages échangés entre deux entités. Les entités en question devront répondre aux pré-

Familles	Algorithme	Complexité force brute	Complexité meilleure attaque	Vulnérabilité
Hachage	MD5	2^{64}	2^{33}	31
	SHA-1	2^{80}	2^{61}	19
	SHA-2 256	2^{128}	2^{128}	0
	SHA-2 512	2^{256}	2^{256}	0
Symétrique	DES	2^{56}	2^{41}	15
	3DES	2^{168}	2^{112}	44
	AES-128	2^{128}	2^{126}	2
	AES-192	2^{192}	$2^{189.9}$	2.1
	AES-256	2^{256}	$2^{254.3}$	1.7
Asymétrique	RSA-1024	2^{87}	2^{87}	0
	RSA-2048	2^{117}	2^{117}	0
	ECC-163	eq 2^{87}	eq 2^{87}	0

Tab. 2.2.: Vulnérabilités des algorithmes usuels

requis de ces algorithmes (connaissance de la clé, puissance de calcul...) afin de les appliquer à leurs messages. D'autres algorithmes auraient pu être sélectionnés pour cette étude mais cela donne déjà un aperçu global de la notion de sécurité et vulnérabilité. Voici cependant quelques algorithmes qui pourraient s'ajouter à cette liste déjà longue.

Mentions honorables

— IDEA : chiffrement symétrique par bloc breveté en 1991 par la société suisse *Mediacrypt*. Les blocs ont une taille de 64 bits et la clé une taille de 128 bits. Il s'agit d'une suite d'opérations à base de XOR, d'additions et multiplications modulaires.

— ElGamal : système de chiffrement (et signature) asymétrique. Il ne se base pas sur le problème de factorisation en nombres premiers comme RSA mais plutôt sur le problème du logarithme discret. Cet algorithme a inventé par Taher Elgamal en 1984 sans faire l'objet d'un brevet. En chiffrant, il a le désavantage de doubler la longueur du message à envoyer mais peut être très utile sur de petits messages.

— Whirlpool : datant de 2000, il s'agit d'un algorithme de hachage dérivé d'une fonction de chiffrement bien connue désormais : AES. Son co-créateur (Vincent Rijmen avec Paulo Barreto) a en effet participé à la création d'AES. Whirlpool permet d'obtenir des empreintes de 512 bits.

Résumé de la partie

L'approche historique a pu montrer que la synergie entre les cryptographes et cryptanalystes a permis des avancées mathématiques et technologiques donnant une complexité de plus en plus grande à la sécurisation des données. L'augmentation du nombre de messages à traiter (et leur importance parfois stratégique) a commencé à faire apparaître quelques éléments du problème : avoir un algorithme facile à utiliser mais difficile à casser, tout en permettant une circulation rapide de l'information (déchiffrement compris).
En passant par les différentes familles d'algorithmes plus modernes (hachage, cryptographie à clé publique, cryptographie symétrique), cette partie a permis d'aborder la notion de temps de cassage d'un algorithme par rapport aux vulnérabilités connues. En croisant les différentes études menées sur les algorithmes les plus courants, il en résulte finalement un tableau récapitulant en terme de complexité algorithmique la durée pendant laquelle une donnée peut être protégée (selon l'intégrité, l'authenticité ou la confidentialité).

3 L'environnement d'étude

> *Quelle chose magnifique que la technique moderne !... Une simple manette, et hop ! à des centaines de kilomètres, un moteur se remet en marche !... C'est prodigieux !...*
>
> **— Haddock**
> Objectif Lune, écrit par Hergé

Si l'historique de la partie introductive a mis en avant plusieurs problématiques de sécurité informatique, il convient à présent d'approfondir ces sujets dans le cas particulier des systèmes embarqués qui participent grandement aux *Wireless Sensor Networks*.

3.1 Wireless Sensor Networks (WSN)

L'expression *Wireless Sensor Networks* (WSN), pour réseaux de capteurs sans fil, regroupe plusieurs termes et notions sur lesquels il s'agit de s'attarder avant d'aller plus loin.

3.1.1 Les capteurs et les actionneurs

Les capteurs

Les capteurs permettent de mesurer une grandeur physique pour la transformer en information utile. Par exemple, un thermomètre est un capteur de température. Chaque capteur est adapté à sa propre mesure. Aussi, le capteur peut être contrôlé par une carte électronique équipée d'un processeur. Dans le cas des systèmes embarqués, le processeur est intégré au sein d'un micro-contrôleur.

Par abus de langage, le capteur [1] représente à la fois l'entité électronique permettant de mesurer la grandeur physique et l'équipement électronique permettant de contrôler ce dernier et de traiter la grandeur recueillie.

Les actionneurs

Les actionneurs, bien que ne mesurant pas de grandeurs physiques, participent aux réseaux de capteurs. Ils peuvent eux-mêmes être associés localement à un capteur relevant une grandeur physique ou jouer le rôle de récepteur attendant une donnée d'un autre capteur pour effectuer une action [47].

L'actionneur doit rendre compte du résultat de son action (succès ou échec) et devient alors un émetteur de données au même titre qu'un capteur au sein d'un WSN. C'est pourquoi, afin de généraliser l'ensemble des cas, les capteurs et actionneurs seront traités de la même manière, c'est-à-dire comme une entité d'un WSN permettant de recueillir et traiter une mesure.

3.1.2 La mesure d'une grandeur physique et son utilisation

La mesure

Pour les capteurs abordés dans cette étude, une mesure est souvent effectuée grâce à un dispositif permettant de transformer une grandeur physique en tension électrique puis de mesurer cette tension.

Le mesure de la grandeur physique peut s'effectuer de différentes manières :

1. Périodique : la grandeur est relevée à intervalles de temps réguliers.
2. Interruptive : la grandeur est envoyée lorsqu'un changement est détecté par le capteur.
3. Mixte : la grandeur est envoyée lorsque cette dernière change ou bien relevée lorsqu'aucun changement n'a été relevé pendant une durée fixe (il s'agit alors de signe de vie du capteur).

Le traitement de la donnée

Pour être traitée numériquement, cette grandeur passe par un Convertisseur Analogique Numérique (CAN) et est désormais considérée comme une donnée prête à

1. Il s'agit ici des capteurs associés à un micro-contrôleur, seuls capteurs abordés dans ce document. En effet, il existe également des capteurs biologiques, tels que les yeux, mais les données y sont rarement sécurisées et le présent document ne se prête guère à la sécurisation du canal de communication qu'est le nerf optique.

être utilisée. Il est toutefois possible de traiter la mesure différemment : c'est le cas de l'instrument analogique qui se contente de transformer la mesure en une autre grandeur [47] (signal électrique, barre de mercure pour le thermomètre classique, etc). Les instruments analogiques ne seront pas abordés ici.

Le cas des systèmes numériques (dont les systèmes embarqués) implique un codage binaire de la mesure (c'est le rôle du CAN), qui aura une taille adaptée à sa capacité d'information. Par exemple, une mesure pouvant prendre 256 valeurs peut être codée sur 8 bits, soit un octet. Au contraire, une mesure très précise pouvant varier sur de grandes gammes de valeurs nécessitera plutôt un *double* : une donnée codée sur 64 bits (8 octets) [2].

La donnée, une fois convertie par le CAN, peut être utilisée localement : afficher l'information sur un écran local, générer une alarme sur une sortie (une LED, une sortie audio...), commander l'arrêt d'une machine, etc.

Au-delà du traitement local, l'intérêt des réseaux de capteurs est la supervision des données et le contrôle des machines à distance. De plus, le traitement de la donnée peut être distant. Il s'agit alors de communiquer la donnée.

L'envoi de la donnée

Une fois codée, la mesure peut être intégrée à un message qui comportera notamment l'identité du capteur. D'autres informations peuvent y être ajoutées. Par exemple, pour vérifier l'intégrité de la donnée, un CRC (*Cyclic redundancy check*) ou mieux encore, une empreinte provenant d'une fonction de hachage peuvent être utilisés. Le destinataire du message refera le même calcul (à partir du message reçu) et si le résultat est identique, le message est intègre. Cette notion de contrôle d'intégrité grâce à des fonctions de hachage est détaillée dans la partie 2.2. De plus, un anti-rejeu (sous forme de compteur ou d'aléa) permet d'éviter la réutilisation d'un message par un attaquant potentiel et une signature peut authentifier le message.

Tout ou partie du message ainsi formé peut également être chiffré pour rendre confidentielles les données qui seront envoyées. La figure 3.1 donne un exemple d'un tel message type avec signature et chiffrement des données.

2. Le principe du capteur reste toutefois le même : une grandeur est convertie en tension évaluée par le capteur mais ce dernier prépare la donnée sous forme de *double* pour être envoyée au micro-contrôleur.

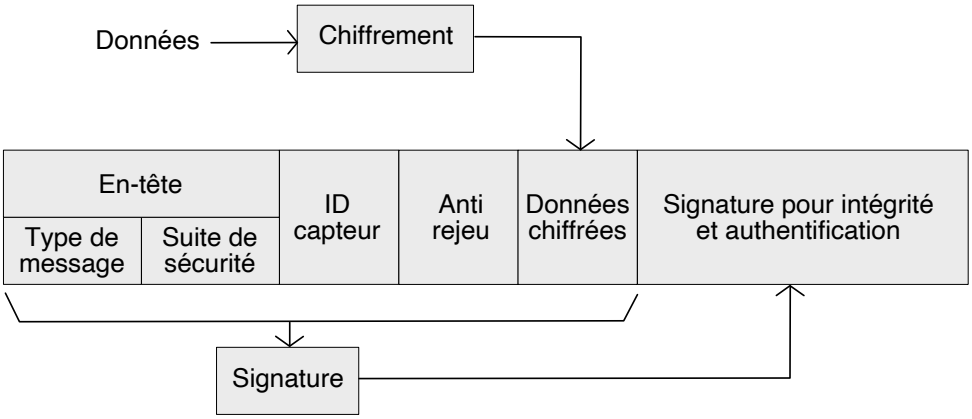

Fig. 3.1.: Exemple de message type

Le support de communication

Le support qui permet l'envoi des données peut différer selon le contexte. En voici une liste non exhaustive [3] :

— Filaire (électrique ou optique) : à travers un courant électrique représentant les bits du message ou une fibre optique.

— Radio : sur les bandes ISM (réservées pour les applications Industrielles, Scientifiques et Médicales) adaptées par exemple.

— Optique (hors fibre) : grâce à un code couleur (rouge = 1, bleu = 2...) ou encore grâce à des images représentant les données. Ces images ou codes couleurs peuvent ensuite être récupérés par un capteur d'images adapté.

Ces supports nécessitent un codage de l'information afin de l'adapter au mieux au support qui la transporte. Cela ne la cache et ne l'authentifie aucunement car ce codage nécessite d'être connu publiquement afin que les récepteurs puissent recevoir convenablement le message de façon normée [4].

Les capteurs filaires ont l'avantage - dans la majorité des cas - de pouvoir recevoir une source d'alimentation stable et de longue durée (potentiellement de durée supérieure à la durée de vie du capteur). En revanche, les systèmes sans fil doivent avoir leur propre source d'énergie et la gérer avec parcimonie, demandant alors souvent de diminuer les envois de messages.

3. Pour compléter la liste, le lecteur de Lucky Luke pensera aux signaux de fumées des indiens, tandis que les nostalgiques des années 80 et 90 se rappelleront des échanges de disquettes.

4. Il est également possible de créer un support et protocole de communication spécifique, dont l'encodage et le décodage ne sont pas publiques. Cependant, si le but est d'empêcher un attaquant potentiel d'accéder à l'information, il faut envisager le temps nécessaire pour réaliser une rétro-ingénierie de cette technologie. Si ce temps est supérieur au temps nécessaire pour casser la chaîne de sécurité mise en place, il peut être envisagé de créer cette nouvelle méthode de communication, et de la garder secrète. Il est alors important de considérer les aspects financiers d'un tel développement en interne face à l'utilisation d'un algorithme éprouvé par toute une communauté

3.1.3 Les capteurs sans fil

Derrière le terme *sans fil* se cachent de nombreuses technologies qui se basent sur la modulation d'ondes sur certaines bandes ISM : 5,8 Ghz, 2,4 GHz, 868 MHz ou encore 433MHz.

Les capteurs sans fil utilisent donc ces bandes ISM comme support de transmission et ont pour spectre d'utilisation des applications assez diverses de part leurs avantages. Les technologies sans fil permettent par exemple d'installer des capteurs sur des objets ou dans des lieux où le filaire rend l'installation compliquée. De plus, l'installation de fils augmentent souvent les coûts d'installation et nécessitent parfois des aménagements particuliers [24].

Néanmoins, sans câble, impossible de rattacher une source d'alimentation externe continue au capteur. Ce dernier est alors équipé d'une batterie permettant de répondre aux besoins énergétiques du micro-contrôleur pour contrôler les différents périphériques.

3.1.4 Les WSN au sein de l'IoT

Les WSN sont composés de ces capteurs sans fil associés à des émetteurs et récepteurs. Une passerelle, faisant donc office de récepteur, permet de connecter ces entités à internet. Ainsi, les WSN forment des sous-réseaux de l'internet des objets (IoT) et permettent de connecter les capteurs aux serveurs, ordinateurs, smartphones et autres entités présentes sur internet.

Il est alors possible d'accéder à un WSN grâce à des applications adaptées. Comme schématisés sur la figure 3.2, ces accès peuvent se faire à distance (cas A), sur site (cas B), ou de manière directe (cas C).

Une telle architecture permet un grand nombre d'usages dans des contextes très différents.

3.2 Applications des WSN

Plusieurs secteurs se sont accaparés les WSN. En voici quatre principaux qui permettent de mettre en avant les caractéristiques principales des WSN : l'industrie, la domotique, la traçabilité et le médical [64]. Le concepteur de solution a le choix parmi une multitude de technologies WSN, et certaines technologies sont plus

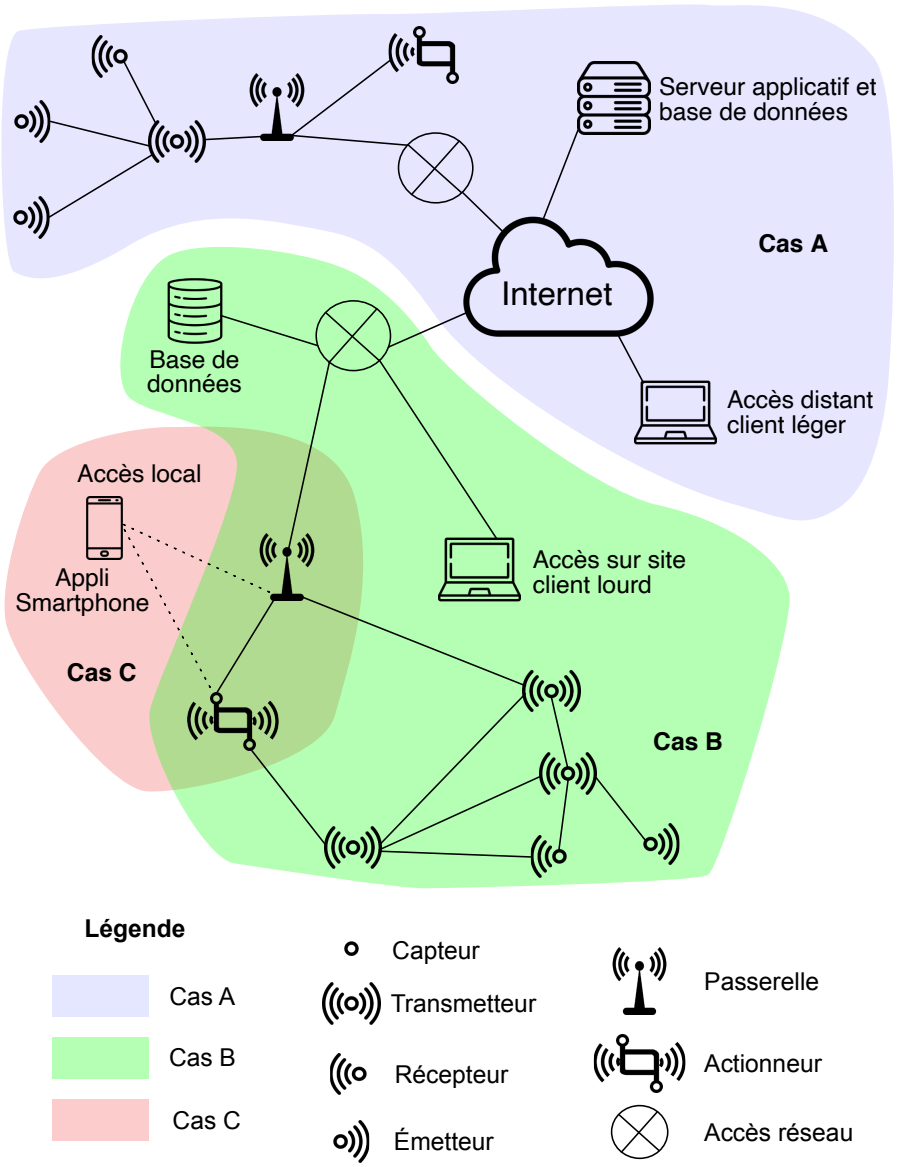

Fig. 3.2.: Exemple de réseaux WSN dans l'internet des objets

adaptées à certains domaines qu'à d'autres [58]. Une revue de ces domaines et technologies est présentée ci-après afin d'observer les choix de conception de protocoles standards, notamment en matière de sécurité, et leurs cas d'usage.

3.2.1 L'industrie

Contexte applicatif

Trois scénarios sont identifiés par [3] sur le développement de l'industrie, en rapport avec le contexte économique global :

1. S1 : Apparition de nouvelles technologies permettant l'accès à de nouvelles valeurs utilisables.
2. S2 : Organisations circulaires multi-échelles.
3. S3 : Néo-industrialisation dans le cadre d'un nouveau contrat social.

Le passé industriel de l'Europe a conduit celle-ci sur la voie de l'innovation et de l'automatisation de l'industrie. Les termes Usine du futur et Industrie 4.0 montrent l'importance de la technologie au sein des scénarios de développement et la donnée est désormais au coeur de ce modèle : valoriser cette donnée pour atteindre des performances inégalées. Les sites de gestion et de supervision étant parfois géographiquement (très) distants, les données et leur traitement sont décentralisés, ce qui implique une supervision et une prise de décision distante (et parfois rapide), comme schématisés sur la figure 3.2.

Dans le contexte du scénario S1, afin de remonter les données à valoriser, les réseaux de capteurs sont la base de l'édifice et se sont multipliés : pour avoir une vision complète de l'usine sans y être, il est nécessaire de remonter le plus de données possibles. Les actionneurs permettant désormais d'activer ou désactiver des machines à distance, la gestion éloignée s'est elle aussi démocratisée[24].

Cependant, les crises écologiques en cours [16] rendent le scénario S2 de plus en plus pressant, tandis que les dernières crises économiques et sanitaires rendent le scénario S3 de plus en plus pertinent. Les directions déjà engagées par les différentes institutions conduiront probablement à un mélange de ces scénarios, où les WSN auront quoiqu'il arrive leur place.

En effet, les WSN sont aujourd'hui utilisés dans l'industrie pour produire plus. Cependant, rien n'empêche les décisionnaires de glisser vers un scénario S2 et S3 pour produire mieux. Les WSN permettent, entre autres choses, d'optimiser les flux de matières ou de détecter des défaillances plus rapidement. Avec l'analyse de l'ensemble

des données collectées, il est même possible de prévoir les défaillances (maintenances prédictives) et donc d'utiliser plus efficacement machines et matières[45]. Il est courant d'utiliser le terme I-IoT pour parler de l'internet des objets industriel.

Toutefois, un WSN mal sécurisé au sein d'une industrie peut être totalement contre-productif dans un contexte géopolitique et économique compétitif. Si les WSN semblent avoir leur place dans les années à venir, il convient donc de les sécuriser convenablement.

Quelques technologies dans ce contexte

Historiquement, l'industrie repose sur les systèmes SCADA (*Supervisory Control And Data Acquisition*), qui permettent de contrôler l'usine et les chaînes de production à l'aide d'automates, de systèmes de communication, et d'interfaces hommes-machines.

Le protocole Modbus est apparu dans un contexte où les SCADA n'étaient pas reliés aux réseaux internet, ce qui explique son grand manque de sécurité : aucun contrôle d'intégrité, aucun chiffrement et aucun moyen d'authentification n'étaient présents. Ceci peut s'expliquer également par le contrôle du support de communication. Ce dernier était filaire (RS232 ou RS-485) et les câbles étaient déployés principalement dans l'usine [35].

Ce protocole s'est aujourd'hui modernisé avec une version *Modbus over TCP* pouvant être complétée par la suite de sécurité TLS venant palier les défauts de sécurité évoqués. Cela permet de connecter les systèmes SCADA déjà en place. Une sécurisation de la couche physique reste néanmoins nécessaire pour protéger efficacement les SCADA. Une politique de sécurité plus large s'impose alors au niveau de l'entreprise [53].

Les WSN industriels cohabitent aujourd'hui dans cet éco-système SCADA, avec ses failles et ses faiblesses. Ces WSN doivent donc être particulièrement bien conçus pour ne pas introduire de nouvelles brèches. Si la Wifi peut être implémentée sans difficulté sur les systèmes SCADA (il s'agit alors d'un réseau Wifi dédié et contrôlé), les nombreux capteurs venant compléter ces systèmes ne sont pas toujours assez puissants pour exploiter la Wifi, et des protocoles comme ZigBee semblent plus pertinents.

Le protocole ZigBee est un standard défini par l'alliance ZigBee et qui repose sur le protocole IEEE 802.15.4. Si ZigBee gère l'applicatif et l'adressage, IEEE 802.15.4 gère l'accès à la couche de liaison (MAC) et à la couche physique[54].

Plusieurs architectures réseaux sont possibles avec ZigBee comme le mode étoile ou le mode *mesh*. ZigBee définit trois entités :

1. Le coordinateur. Sa mission est de créer et gérer le réseau. Il peut être vu comme un équivalent d'un point d'accès unique.
2. Le routeur. Sa mission est de relayer les messages à un noeud suivant jusqu'au coordinateur. Un routeur peut également générer ses propres messages.
3. Le *end-device*. Cette entité utilise des fonctionnalités réduites par rapport aux deux autres. Il ne peut qu'envoyer ou recevoir les messages.

Bien qu'expliqué ici dans un contexte industriel, le protocole ZigBee pourrait également s'intégrer pour d'autres cas applicatifs, comme les villes intelligentes par exemple.

3.2.2 Les villes intelligentes

Contexte applicatif

L'utilisation des WSN dans les *Smart Cities* (villes intelligentes) passe par des infrastructures publiques ou privées ayant pour objectif d'améliorer l'usage de la ville. Par exemple, les transports publiques s'équipent de capteurs communiquant avec un centre logistique pour améliorer la fluidité de la circulation. Le déploiement de boites à colis connectés, d'appareils de vidéo-surveillance ou de gestion de l'énergie sont autant d'outils nécessitant la mises en place de WSN.

Certaines applications, comme les drones, posent la question de la vie privée [55]. Si la politique d'accès à ces données et de leur utilisation est hors champ, il est néanmoins légitime de poser la question de la sécurisation de celles-ci. En effet, si les données circulent librement sur un réseau, toute personne connectée à ce réseau peut potentiellement y accéder.

La domotique s'invite également dans les villes intelligentes. Les applications résidentielles proposent de connecter les services locaux (pressing, commerces de proximités, conciergeries...) aux résidents, et ces mêmes applications peuvent s'inviter jusqu'aux objets connectés présents chez ces derniers. Ainsi, un réfrigérateur vide pourra envoyer une notification au résident, lui proposant de commander l'aliment manquant auprès du commerçant le plus proche pour que ce dernier le lui livre dans sa boite à colis connectée [5].

5. Si les évènements de l'année 2020 ont permis de mettre en avant les avantages de ce types de solution, il peut être pertinent de se poser la question du lien social dans un tel contexte et aussi de l'utilisation des matières et ressources pour ces objets

Toute cette chaîne de domotique doit être sécurisée pour que le résident puisse avoir confiance dans son environnement désormais connecté.

Quelques protocoles sans fil dans ce contexte

La technologie *EnOcean* propose d'équiper les habitations de capteurs *sans fil et sans batterie*. L'énergie est alors récupérée à partir d'interaction avec le capteur comme une pression (la fermeture d'une porte ou d'une fenêtre, un appui sur un interrupteur...). Cela suffit pour une brève communication, généralement non acquittée car une fois le message envoyé, le système embarqué n'a plus l'énergie nécessaire pour passer en mode *récepteur* en attente d'une confirmation de réception.

En n'utilisant pas d'acquittement, *EnOcean* repose sur des répétitions de messages afin d'éviter une perte d'information. Il est ainsi possible de dimensionner un réseau *EnOcean* en fonction du nombre de capteurs et de messages générés face au nombre d'erreurs remontées [41].

Si *EnOcean* offre des échanges de données très limités, ce n'est pas le cas du Wifi ou du Bluetooth, proposant des débits plus importants. En particulier, le Bluetooth [6] permet de communiquer avec des objets connectés grâce à des équipements adaptés.

Si le Bluetooth était d'abord utilisé pour des casques audio par la société Ericsson dans les années 1990, la technologie est aujourd'hui utilisée sur des équipements très divers, allant des imprimantes aux consoles, en passant par les véhicules. La domotique s'est également appropriée la technologie pour permettre de configurer plus facilement les appareils de la maison, notamment à l'aide du smartphone.

L'avantage de cette technologie est son développement orienté basse consommation, ce qui implique quelques compromis, notamment une faible puissance d'émission imposant une certaine proximité pour un débit pouvant atteindre 24 Mb/s. Le protocole Bluetooth permet d'associer des appareils entre eux afin de les faire communiquer. La procédure d'association est traitée de manière différente par rapport aux communications plus classiques (échanges de données).

6. L'origine de ce nom est lié à son lieu d'origine : Lund, en Suède. Ce nom fait référence au roi Harald à la dent bleue (*Harald Bluetooth*) qui a unifié les tribus danoises. L'initiateur de cette technologie, Nils Rydbeck, proposait alors une analogie où le Bluetooth pourrait unifier les communications entre équipements connectés.

3.2.3 Traçabilité et mobilité

Contexte applicatif

La traçabilité intervient entre autre dans des contextes de suivis logistiques. Plusieurs approches sont possibles pour gérer ces problématiques. La première, historique, est d'identifier les produits avec un code. Par exemple, le code barre permet d'associer une information sur un produit, mais ce dernier nécessite d'être visible par un lecteur. Des technologies plus efficaces, comme le RFID (*Radio-frequency identification*) décrit ci-après, permet d'envoyer cette information par radio.

Cependant, pour la traçabilité des produits ou des équipements sur de grandes distances en prenant en compte leur mobilité, des technologies différentes entrent en jeu. Ainsi, le suivi de véhicules, notamment autonomes, peut reposer sur les technologies GSM permettant des communications longues distances, sous réserve de couverture réseau (4G, LTE, bientôt 5G...).

Certains objets peuvent également donner leur position GPS avec les réseaux LoRa et SigFox : une très grande portée pour un très bas débit (messages peu courants et très courts). Il s'agit ici d'un compromis entre le débit, la portée et l'énergie consommée par l'envoi d'un message.

Une information codée de manière non sécurisée permettrait ici à un attaquant d'accéder à des données qui ne lui sont pas destinées ou encore de réécrire ces données pour falsifier l'information (code barre, étiquette RFID). De même, une information transportée de manière non sécurisée par radio (GSM) induit des problématiques similaires. Il est alors évident qu'un système de traçabilité de confiance nécessite des fonctions de sécurité efficaces.

Quelques technologies sans fil dans ce contexte

La technologie RFID évoquée précédemment permet d'envoyer, sous sa forme passive, un identifiant par ondes radio lorsqu'une sollicitation extérieure est reçue (par un lecteur RFID par exemple) [9]. Dans ce contexte, il est courant de parler d'étiquettes RFID ou encore tags RFID. Un tel tag est composé d'une unité de calcul et d'une antenne ainsi que d'une mémoire contenant l'identifiant.

La forme active du RFID consiste en un tag RFID équipée d'une batterie permettant d'effectuer des calculs en plus de l'envoi d'un message. Les tags RFID actifs peuvent être associés à un micro-contrôleur pour gérer d'autres actions. Dans ce cas leurs

applications peuvent aller bien au-delà de la traçabilité en les associant à des capteurs pour faire remonter des mesures en plus d'un identifiant par exemple

Les communications GSM (2G, 3G, 4G, LTE et bientôt 5G) et ses équivalents bas débits : LoRa et SigFox permettent des envois des données longues distances. Sous réserve de couverture réseau, l'objet peut alors communiquer sa position et d'autres données (pression, température, alarme...).

3.2.4 Médical

Contexte applicatif

L'utilisation de l'internet des objets dans le milieu médical a permis l'émergence d'un nouveau domaine : l'*eHealth*. Celui-ci permet de collecter et analyser (localement ou à distance) des données de patients. Récolter ces données dans un cadre médical permet des applications directes (sur un patient ou un ensemble de patient) comme réguler le taux de sucre, superviser un pacemaker... mais permet également de rendre des services comme une assistance médicale personnalisée [38].

La collecte et l'utilisation de telles données soulèvent évidemment des inquiétude quant à leur sécurisation, en particulier si le système embarqué dans le corps peut agir sur ce dernier...

Quelques technologies sans fil dans ce contexte

La variété des technologies de l'*eHealth* peut regrouper l'ensemble des technologies et protocoles évoqués précédemment. Il faut alors que chaque sous-réseau et chaque interaction entre équipements soient correctement sécurisés. Cependant, la gestion des clés de chiffrement, d'une part, mais également les évolutions constantes des protocoles et des nouvelles technologies d'autre part rendent difficile une sécurisation très longue durée de ces équipements qui ne peuvent pas être modifiés facilement.

Face à ce constat, différentes approches de la sécurité sont apparues. Certaines études par exemple propose une approche par la théorie du jeu ([14]) afin d'adapter les politiques de sécurité plus efficacement face à un changement de règles (quand une nouvelle faille est connue par exemple). Les stratégies possibles sont d'ailleurs basées sur des compromis entre la sécurité et l'énergie des systèmes. D'autres approches abordent le problème d'un point de vue systémique où chaque élément de l'IoT a une complexité de sécurité et une surface de vulnérabilité variables ([43]) : un état

de l'art constant des failles et avancées technologiques doit avoir lieu pour mettre à jour ces éléments pour garder un niveau de sécurité acceptable au sein du WSN.

Il peut être alors intéressant de voir quelles bonnes pratiques sont déjà utilisées dans certains protocoles de WSN déjà évoqués et dans la conception de systèmes embarqués.

3.3 Sécurité d'un WSN

Si de nombreux axes de sécurisation de l'internet des objets et leurs utilisations peuvent être envisagés (l'accès à l'application de supervision, l'accès à la base de données, la protection des serveurs...) seule la sécurisation des WSN sera abordée.

En particulier, les implications du choix d'une suite de sécurité au sein d'un WSN sur la disponibilité et l'autonomie des équipements seront approfondies. Afin d'abstraire la problématique à l'ensemble des WSN, les protocoles précédemment évoqués seront abordés du point de vue de la sécurité.

Les problématiques de sécurisation liées au système embarqué lui-même seront rapidement évoquées car mettre en place des échanges sécurisés est vain si la porte d'entrée que sont le firmware et le hardware reste ouverte.

3.3.1 Sécurisation du protocole

Plusieurs mécanismes protocolaires peuvent entrer en jeu, comme pour fiabiliser la communication en plus de la sécuriser :

- Le contrôle d'intégrité du message.
- Le chiffrement des données[7].
- L'authentification du message.
- L'acquittement d'un message : cela implique à l'émetteur de passer en mode *récepteur* un bref instant (à estimer) pour recevoir un message d'acquittement du récepteur. En cas de non réception de cet acquittement, l'émetteur peut passer à la tache suivante ou bien tenter à nouveau un envoi.
- La répétition d'un message : parfois préférable à l'acquittement, la répétition d'un message permet d'augmenter la probabilité de réception dans un environnement potentiellement perturbé. Cependant, une fois les messages répétés,

7. Tout le message n'est pas forcément chiffré. En effet, les en-têtes sont souvent utiles pour un récepteur avant analyse du message afin de réaliser un premier pré-traitement et éventuellement filtrer ce message s'il n'est pas destiné à l'entité qui le reçoit.

l'émetteur n'a aucune certitude sur la bonne réception du message et passe à la tâche suivante.

— L'utilisation d'un anti-rejeu : il s'agit d'un compteur dont la valeur est intégrée à chaque message pour être utilisée côté récepteur afin d'ignorer les messages déjà reçus. Ce compteur permet en particulier de ce prémunir des attaques de rejeu de messages.

— Un mécanisme de CSMA-CA/CD (*Carrier Sense Multiple Access-Collision Avoidance/Collision Detection*) : cela permet d'optimiser l'utilisation du canal de communication en diminuant la probabilité que deux messages arrivent en même temps au récepteur (CA) voir à une écoute préliminaire du canal pour vérifier sa disponibilité (CD). Ce dernier point est plus difficile à mettre en oeuvre dans le cas des systèmes embarqués communicants autonomes.

La revue de protocoles proposée ci-après les comparera de ce point de vue en indiquant quels mécanismes entrent en jeu. Il sera précisé également à quels types de WSN ces protocoles sont principalement adaptés en terme d'autonomie, débit, portée...

ZigBee

ZigBee intègre différents mécanismes vus précédemment comme un anti-rejeu, du CSMA/CA, une gestion de clés de chiffrement symétrique ainsi qu'un système de chiffrement et d'authentification (associée à du contrôle d'intégrité). Ce système est basé sur AES-128 en mode CCM (qui permet donc de gérer l'authentification avec AES - voir partie 2.2). Ces mécanismes sont proposés avec l'option de ne faire que l'authentification (et le contrôle d'intégrité), que le chiffrement, ou les deux.

Toutefois, certaines failles ont été trouvées [56]. Elles se basent notamment sur les mécanismes d'échange de clés (interception de l'échange et usurpation de la clé) et sur l'envoi de signaux provoquant un réveil des *end-devices*, perturbant donc leur fonctionnement.

Ce protocole est adapté à la moyenne portée (jusqu'à une centaine de mètres en extérieur) mais la couche protocolaire assez lourde de ZigBee demande souvent l'utilisation d'un RTOS (*Real-Time Operating System* [8]) et consomme beaucoup plus d'énergie qu'un programme classique.

8. Un RTOS est un système d'exploitation très léger adapté à certains micro-contrôleurs. Il permet de gérer des problématiques de tâches parallèles en allouant du temps processeur aux différentes tâches grâce à un système d'interruption. Particulièrement adapté à la gestion des couches protocolaires comme Zigbee, le nombre d'interruptions provoquées par l'utilisation du protocole (et toutes ses sous-tâches) impacte directement l'utilisation de la batterie. L'usage des RTOS sort du cadre de cette étude.

EnOcean

Afin de sécuriser ses échanges, *EnOcean* utilise un système de chiffrement symétrique, AES-128 [9], associé à un CMAC pour vérifier l'intégrité des données.

CMAC est un algorithme de contrôle d'intégrité et d'authentification qui utilise les mêmes fonctions de base qu'un algorithme de chiffrement plutôt que de devoir passer par un autre algorithme différent comme ceux associés à HMAC. Cela s'explique par la cible de l'alliance EnOcean, en charge du développement du protocole : des objets connectés très peu encombrants et ultra basse consommation. Réduire l'espace mémoire du programme permet d'utiliser des micro-contrôleurs plus petits et donc moins encombrants et éventuellement moins gourmands en énergie.

De plus, un RLC (*Rolling Code*) joue le rôle d'anti-rejeu pour éviter à un attaquant éventuel de rejouer un message. Ce RLC est incrémenté à chaque message, ce qui permet au récepteur d'ignorer les RLC inférieurs au dernier reçu et validé [10].

Un mécanisme de CSMA/CA est mis en place à travers l'utilisation d'un aléa qui, associé à un *timer*, réduit la probabilité que deux messages soient envoyés et reçus en même temps (il faut alors que la probabilité que deux aléas identiques soient choisis par deux émetteurs différents au même instant soit faible).

Bluetooth

Plusieurs modes de sécurité existent pour le Bluetooth :

— Mode 1 : pas de sécurité.

— Mode 2 : une communication sécurisée est établie une fois l'association entre équipements établie (les échanges au moment de l'association ne sont pas sécurisés).

— Mode 3 : la communication est établie avant même l'association.

De nombreux jeux de clés de chiffrement et authentification interviennent selon les phases du protocole (association, échange de clés, échange de données...). Un signal d'horloge associé à un nombre aléatoire joue le rôle d'anti-rejeu. Ce type d'anti-rejeu nécessite une synchronisation parfaite entre l'émetteur et le récepteur pour que ce dernier le détermine de la même manière et déchiffre le message. Les algorithmes de chiffrement et d'authentification utilisés par le Bluetooth sont des algorithmes propriétaires.

9. EnOcean utilise une version VAES (pour Variale AES), et a déprécié AES-CBC précédemment utilisé dans leur protocole.

Plusieurs failles de sécurité ont été mises à jour concernant le protocole Bluetooth (référencées de manière non exhaustive dans [51]), comme la fuite d'information (*InformationTheft* et *BluePrinting*), l'utilisation d'équipements non autorisés (*ServiceTheft*) ou encore la prise de commande d'un appareil ou de données (*BlueJacking*).

L'analyse des vulnérabilités protocolaires et applicatives sort du cadre de cette étude. Il semble toutefois pertinent de mettre en évidence que la connaissance de telles failles n'impacte pas l'utilisation du protocole, notamment dans le domaine du grand public (toujours d'après l'étude [51]). En effet, le protocole Bluetooth est de plus en plus populaire aujourd'hui et arrive même dans certains produits industriels et médicaux, malgré les failles et la surface d'attaque assez large introduite avec celui-ci.

RFID

Les technologies RFID, notamment passives, manquent de puissance de calcul pour gérer des mécanismes de sécurité complexes comme l'échange de clés. Les clés sont souvent préalablement définies et gardées secrètes. Des travaux proposent néanmoins des méthodes pour échanger une clé à la volée lors de l'établissement de la communication [15]. La clé ainsi générée permet de sécuriser les échanges et potentiellement les données stockées sur le tag RFID.

Seules les formes passives de RFID seront considérées pour la suite. Dans ce contexte où les communications sont relativement proches, le manque d'anti-rejeu, d'acquittement et de répétition de messages est compensé par un réseau très courte portée et en théorie de meilleure qualité, permettant une réception des données plus aisée pour le lecteur RFID.

Le contrôle d'intégrité, le chiffrement et l'authentification sont possibles mais de façon passive : les données ont été chiffrées, signées et hachées au préalable, et au moment de la lecture de ces données, le lecteur réalise les vérifications nécessaires. La puissance cryptographique est externalisée du tag RFID.

Dans le cas où la clé est gérée par le lecteur de tag, un vol de lecteur (voir d'un équipement capable d'écrire sur la mémoire RFID) compromet la sécurité des objets RFID. Il en va de même si la clé fuite et est utilisée par un attaquant pour créer un équipement ad-hoc pouvant écrire et lire dans les objets RFID.

Protocole	Contrôle d'intégrité	Chiffrement	Auth	ACK ou REP	Anti-rejeu	CSMA
ZigBee	Oui	Oui	Oui	ACK	Oui	CA
EnOcean	Oui	Oui	Oui	REP	Oui	CA
Bluetooth	Oui	Oui	Oui	ACK	Oui	CA
RFID	Oui	Oui	Oui	$\emptyset$	$\emptyset$	$\emptyset$
LoRa	Oui	Oui	Oui	REP	Oui	CA
SigFox	Oui	Oui	Oui	REP	Oui	CA

Tab. 3.1.: Mécanismes protocolaires pour les WSN

LoRa et SigFox

Ces deux technologies [10] reposent sur du bas débit et de la très basse consommation et implémentent un chiffrement symétrique AES-128 avec des clés de chiffrement pré-partagées [6].

Les deux protocoles utilisent également des principes d'anti-rejeu, d'envois multiples de messages sur différents canaux (fréquences) de diffusion pour éviter les collisions et les risques d'usurpation.

La différence entre LoRa et SigFox repose principalement sur l'ouverture du système LoRa par rapport au système SigFox : en effet, sur SigFox, un message est envoyé à l'antenne puis sur un serveur SigFox par leur réseau privé et est ensuite accessible grâce à leurs serveurs tandis qu'avec LoRa, il est possible de gérer soi-même la plateforme de réception (l'antenne) pour envoyer sur ses propres serveurs le message.

Récapitulatif des protocoles

Le tableau 3.1 récapitule les mécanismes de sécurité des protocoles abordés précédemment. La présence de contrôle d'intégrité, de chiffrement ou d'authentification n'indique pas que ces mécanismes sont toujours là mais qu'il est possible de les activer. Dans le cas particulier des technologies RFID passives, cela implique d'externaliser ce traitement.

Si la majorité des protocoles de communication de l'IoT intègrent aujourd'hui la sécurité dés la conception (*by design*), il convient également de la penser au moment de l'élaboration du système embarqué et lors de sa programmation.

10. Seules les communications avec les antennes relais sont considérées. La suite de la communication avec les serveurs est sécurisée grâce à une communication SSL et un VPN pour SigFox par exemple.

3.3.2 Sécurisation d'un système embarqué

Une fois la clé échangée (au moment de la programmation ou par un protocole d'échange), cette dernière sera stockée. Le stockage peut se faire sur une mémoire morte pour une réutilisation future, ou en RAM pour une utilisation immédiate. Le stockage en mémoire morte peut être interne au micro-contrôleur ou externe dans un module mémoire.

Dans le cas où la clé est stockée dans la mémoire du micro-contrôleur, il est nécessaire de s'assurer que la mémoire est protégée en lecture. C'est-à-dire que pour accéder à la mémoire du micro-contrôleur depuis l'extérieur de ce dernier, il faut l'effacer. Ceci permet également de protéger l'accès au code lui-même afin de se prémunir d'une éventuelle rétro-ingénierie.

Toute donnée sensible nécessitant un stockage, en particulier dans une mémoire externe (hors du micro-contrôleur), doit être chiffrée pour ne pas être lue et même signée pour être authentifiée. En effet, un simple CRC ou une empreinte provenant d'une fonction de hachage peut prévenir d'une erreur, mais pas nécessairement d'une attaque volontaire.

Il est toutefois possible d'accéder à cette clé par des attaques de dépassement de mémoire. Si les *buffer overflow* permettent d'accéder à d'autres valeurs de la RAM, il est également possible d'accéder à d'autres zones d'exécution du programme, en particulier la zone protégée du micro-contrôleur où se trouve la clé. Pour éviter ces dépassements, il faut vérifier et valider toutes les entrées du programmes, et également penser au *typage* des caractères : un caractère UNICODE prend plus d'espace mémoire qu'un caractère ASCII par exemple.

Pour éviter toute fuite, il faut également penser à désactiver les interfaces de *debug* et les *bootloaders*. Et enfin, pour éviter la prédictibilité de l'aléatoire à l'origine de certains mécanismes de chiffrement, il convient de trouver une vraie valeur aléatoire, en se basant sur une mesure physique imprédictible par exemple. L'utilisation d'une même graine pour un générateur de nombre pseudo-aléatoire rend ce dernier prédictible, surtout quand il est directement utilisé pour un système de chiffrement [37].

Cette liste n'est pas exhaustive et doit s'étendre à l'ensemble des attaques physiques. Il est toutefois important de garder ces notions en tête lorsqu'il s'agira de considérer la sécurisation des communications. De même qu'une clé privée sur un serveur web doit être sécurisée, la clé utilisée par un système embarqué doit être protégée.

Pour finir sur ces sécurisations, il est intéressant de noter l'existence de modules électroniques d'outils cryptographiques. Les HSM, pour *Hardware Security Module*, permettent de stocker et protéger des clés de chiffrements de manière sécurisée et viennent ainsi en support au micro-contrôleur avec lequel ils communiquent (au sein d'une même carte électronique).

Les PUF, pour *Physically Unclonable Function* peuvent être utilisés pour générer des clés grâce à des couples de challenge/réponse uniques à chaque composant [23].

Enfin certains modules d'accélérations matérielles sont conçus spécifiquement pour certaines opérations de sécurité, comme pour le chiffrement AES par exemple et peuvent donc soulager le micro-contrôleur d'opérations relativement lourdes et coûteuses.

3.3.3 Quel attaquant ?

Depuis le début du document, le mot attaquant est utilisé pour définir une personne souhaitant perturber le fonctionnement initial du système. Ce dernier possède plusieurs outils et techniques pour y arriver.

L'attaquant peut occuper un canal de communication en brouillant ce dernier, ou en bombardant le récepteur de messages illégitimes, ou encore rejouer un message authentique. Les parades ont déjà été évoquées et correspondent à la mise en place de mécanismes protocolaires plus ou moins complexes (CSMA-CA/CD, anti-rejeu, répétition de messages, etc).

La suite de cette étude considère un attaquant qui récupère un message et cherche à perturber le WSN d'une manière ou d'une autre. En supposant un protocole correctement développé pour gérer les problématiques évoquées précédemment, l'attaquant peut alors :

1. Intercepter un message pour accéder aux données : il faut alors les protéger avec un mécanisme de chiffrement.
2. Modifier un message : il faut alors utiliser une signature ou un contrôle d'intégrité.
3. Créer un message de toute pièce : il faut alors authentifier l'émetteur.

Si l'attaquant réussi l'une de ces trois actions, c'est qu'il a pu casser l'algorithme de sécurité mis en place. Il a été vu qu'il était impossible de se prémunir indéfiniment d'un attaquant mais il peut être intéressant de s'en prémunir le plus longtemps possible (au delà de sa durée de vie serait déjà pas mal !).

Cependant, mettre une sécurité trop importante peut rendre le système embarqué inutilisable car trop lent ou trop gourmand en énergie. Des travaux ont cherché à s'attaquer à ce problème multi-critère notamment avec des questionnements d'optimisations.

3.4 État de l'art sur les travaux d'optimisation

Aujourd'hui, améliorer les performances des réseaux IoT (consommation d'énergie, réactivité et disponibilité) peut passer par la conception de nouvelles batteries, de composants moins énergivores, ou encore par des optimisations sur l'utilisation des objets connectés.

Ces optimisations peuvent intéresser le concepteur de systèmes embarqués par deux approches :

— Économique : un système embarqué plus performant peut mieux se vendre.

— Environnemental : un système embarqué qui consomme moins nécessite moins d'énergie. Si l'énergie consommée pour un capteur peut paraître dérisoire, un nombre important de systèmes embarqués peut consommer une énergie non négligeable à grande échelle [11].

3.4.1 La disponibilité et la réactivité

Pour rappel, les systèmes embarqués basés sur un micro-contrôleur ne peuvent pas réaliser plusieurs tâches en même temps : il s'agit de systèmes à un seul *thread*. L'algorithme de sécurité, basé sur des calculs lourds, occupe entièrement le processeur et les autres tâches doivent attendre.

Réduire le nombre de messages à envoyer permet de réduire les communications et donc le temps nécessaire à la sécurisation de celles-ci. Une partie du traitement de la donnée peut être gérée localement par le système embarqué avant d'être envoyée. Notamment, si le traitement ne peut être fait localement ou bien nécessite des agrégats de données de plusieurs capteurs, il est possible de mettre en place des

11. Attention cependant au paradoxe de Jevons, également appelé *effet rebond*. Ce dernier apparaît lorsqu'une amélioration en terme d'utilisation d'une ressource entraîne finalement une plus grande consommation de celle-ci (ici l'énergie).

infrastructures *Edge Computing* (ou encore *Edge Cloud* [29]) permettant de gérer ces calculs sur un site proche du système embarqué et pas forcément sur un serveur très distant du *Cloud*.

L'utilisation de telles infrastructures permet d'améliorer la qualité de service et la réactivité d'un système dans sa globalité en externalisant le traitement lourd de données (les flux vidéos est d'ailleurs pris en exemple dans [29]), mais la consommation énergétique devient plus importante pour les serveurs. Cette décentralisation du traitement de la donnée permet néanmoins d'augmenter l'autonomie du système embarqué, critère très important pour le concepteur et l'utilisateur final. Ces améliorations peuvent être moins évidentes pour des traitements plus léger et des envois réguliers de petites quantités de données.

3.4.2 La consommation d'énergie

L'autonomie d'un système embarqué sans fil dépend de la quantité de données à envoyer, de la portée souhaitée du signal (puissance du signal) et du canal choisis pour la communication radio. Avec ces trois paramètres, il est possible d'utiliser des techniques d'optimisation basées sur des algorithmes évolutionnistes pour trouver la configuration multi-critères optimale [62].

Les cycles de veille et d'activité d'un système embarqué font également partis des paramètres à prendre en compte pour optimiser la consommation d'un WSN tout en réduisant le nombre de collisions potentielles sur le réseau : un capteur endormi n'occupe pas le canal et permet à un autre capteur d'envoyer correctement son message. Cette problématique est abordée dans [8] où l'étude considère un WSN où chaque noeud peut communiquer avec ses voisins qui peuvent être en veille ou actifs.

La gestion de la veille d'un noeud critique, c'est-à-dire d'un noeud demandant une réactivité importante doit se faire de manière particulière. C'est pourquoi dans la suite de cette thèse, de tels noeuds ne seront pas mis en veille mais plutôt soumis à une alimentation continue.

3.4.3 La sécurité algorithmique

La sécurité est nécessaire pour assurer un fonctionnement serein et efficace du WSN. C'est d'ailleurs la motivation du projet PACLIDO, regroupant industriels et collectivités autour de la sécurisation de l'internet des objets [12].

12. Le projet PACLIDO est détaillé sur le site `https://paclido.fr/`

Le choix de l'algorithme de sécurité est un paramètre tout aussi important pouvant être vu comme la vulnérabilité maximale tolérée. Les études de [42] et [4] mettent en avant le lien entre performance et choix des algorithmes de sécurité et de leurs paramètres. Notamment, la notion de vulnérabilité de l'équation 2.1 est intéressante pour évaluer l'impact d'un paramètre (taille de la clé par exemple) au sein d'un même algorithme (ou famille d'algorithme).

Toutefois, afin de comparer les algorithmes entre eux, la complexité de cassage de l'attaque la plus efficace sera préférée ; autrement dit : le temps minimum avant cassage de la sécurité (donnée lue ou corrompue, signature falsifiée...). Ainsi les données du tableau 2.2 seront utilisées comme paramètre de sécurité par rapport aux algorithmes choisis.

L'énergie consommée par les algorithmes de sécurité est dépendante de façon linéaire à la durée d'exécution de ces algorithmes [17]. Cette étude mesure la consommation d'énergie et la performance de différents algorithmes sur des systèmes embarqués, en considérant leurs contraintes temps réel. En faisant varier la taille des données à chiffrer, l'impact sur la vitesse d'exécution et l'énergie consommée peut être observée.

Une méthode similaire sera utilisée dans cette thèse (partie 4) avec les algorithmes présentés dans la partie 2.2 afin de mettre en évidence ces effets sur un émetteur à base de micro-contrôleur. Il s'agit cependant d'aller plus loin en prenant en compte le nombre de messages à envoyer.

3.4.4 Le bon compromis

Trouver le bon compromis entre ces trois grandeurs (sécurité, autonomie et disponibilité) n'est pas aisé et dépend des cas d'applications, plus précisément de l'environnement de mesure. Des études modélisent partiellement ce problème, notamment à l'aide de réseaux de Petri.

Les réseaux de Petri sont des outils de modélisation de systèmes parallèles et distribués. Ils doivent leur nom à leur créateur : Carl Adam Petri, informaticien et mathématicien allemand qui formalisa la notion de réseaux de Petri dans sa thèse de doctorat *Kommunikation mit Automaten* (Communication avec les automates) [40].

Ces réseaux de Petri seront détaillés et formalisés dans la partie 5 mais d'autres travaux ont déjà utilisé cet outil de modélisation puissant en l'associant à des ressources

physiques (énergie, disponibilité) et à des contraintes temporelles, donnant ainsi des RBTPN (*Resources Based Time Petri Nets*) [60] et [61].

Un tel système de représentation est idéal pour schématiser des réseaux distribués temps réel de systèmes embarqués. Si ces études se concentrent sur les problèmes de synchronisations et de planifications, il manque l'aspect sécurité. Ces modèles peuvent néanmoins être une base pour une approche impliquant la sécurité et les mesures de grandeurs physiques ; cette approche est d'ailleurs présentée et modélisée dans la partie 5.

L'étude [18], en se focalisant sur un seul système embarqué, va plus loin en intégrant la sécurité et la qualité de service à l'aide de réseaux de Petri RBTPN avec des réplications de tâches pour s'assurer de leur bonne exécution. Cette méthode est intéressante au sein d'un système embarqué à plusieurs composants (qui doivent communiquer entre eux). Le parallèle peut être fait avec un WSN composé de plusieurs émetteurs et d'un récepteur.

La suite de cette thèse se concentre justement sur un tel réseau de systèmes embarqués, où le meilleur compromis est recherché pour résoudre ce dilemme en fonction des contraintes du concepteur et de l'environnement où le WSN sera utilisé.

Résumé de la partie

Les réseaux de capteurs sans fil (WSN) s'imposent de plus en plus à travers les différents secteurs d'activité (industrie, médical, domotique...) et s'interconnectent entre eux et des serveurs pour former l'internet des objets (IoT).
Les systèmes embarqués d'une telle diversité d'applications sont forcément différents et s'adaptent à leurs contraintes fonctionnelles. Des protocoles de communication ont émergés pour répondre à ces contraintes et montrent la pratique désormais usuel d'intégrer la sécurité dés la conception du système, ce qui n'empêche pas la découverte régulière de nouvelles failles. Si sécuriser un WSN est aujourd'hui plus aisé (nombreuses librairies et documentations à disposition), il est plus difficile de mesurer l'impact de la sécurité sur les performances d'un WSN.
Des projets et papiers scientifiques proposent des approches pour modéliser et simuler le comportement de réseaux de systèmes embarqués mais sans considérer ces trois aspects ensemble : autonomie (pour l'émetteur), sécurité (pour la communication) et qualité de service (ou disponibilité du récepteur).

4

Le dilemme autonomie, disponibilité et sécurité

> *Approuveriez-vous l'idée de rédiger une question qui poserait d'emblée un choix binaire plutôt que d'en rédiger une qui comporte en son sein deux sous-questions, dont chacune est le parfait oxymore de l'autre ?*
>
> **— Agent Calot**
> Au service de la France, réalisé par Alexandre Courtès et Alexis Charrier

En prenant l'exemple particulièrement parlant d'un pacemaker communicant, il est aisé de comprendre ce dilemme. Communiquer avec un pacemaker permet de vérifier son état de manière plus efficace et d'adapter sa configuration de façon plus aisée pour le patient qu'une opération chirurgicale, seule solution pour y accéder physiquement.

Une telle communication non sécurisée permettrait une attaque pouvant littéralement tuer la personne. Il est donc évident ici que sécuriser ces communications est essentiel.

D'autre part, un pacemaker doit être le plus autonome possible, et le moins encombrant possible. Cela implique des contraintes fortes sur la batterie (généralement une pile en lithium d'une durée de vie de 10 à 15 ans). Si l'algorithme de sécurité ici augmente la consommation d'énergie, ce n'est pas le principal problème. En effet, un algorithme de sécurité mal choisi et mal paramétré risque d'occuper le pacemaker aux opérations de sécurité pendant un temps relativement long, bloquant le pacemaker et l'empêchant de revenir à ses tâches vitales de base. Autrement dit, l'algorithme de sécurité compromet la disponibilité et, de façon secondaire, l'autonomie du pacemaker.

L'objectif de cette partie est de mettre en avant les paramètres et variables sur lesquels le concepteur de systèmes embarqués pourra agir pour répondre à ses

critères. Il s'agit alors pour ce dernier de trouver la meilleure configuration qui respectent ses conditions de départ comme par exemple :

— L'autonomie des émetteurs supérieure à une durée T_{vie}

— Une sécurité des messages pour une durée supérieure à T_{secu}

— Une disponibilité du récepteur (ou qualité de service) supérieure à Q (exprimée en pourcentage)

Il se peut que des critères trop contraignants par le concepteur de systèmes embarqués ne permettent pas de trouver de solutions. C'est le cas par exemple si T_{vie} est fixée à 1 million d'années, ou si T_{secu} doit être infinie.

Si T_{secu} peut être évaluée en fonction du tableau 2.2 selon les algorithmes choisis et leur complexité de meilleure attaque, T_{vie} et Q dépendent de l'implémentation matérielle qu'il convient d'évaluer.

Pour ce faire, cette partie est décomposée en trois sous-parties. Une première sous-partie donne des mesures concrètes de performances des algorithmes de sécurité usuels. Ces mesures seront utiles pour observer l'influence du choix des algorithmes de sécurité sur l'émetteur et le récepteur. Justement, la deuxième sous-partie se place du point de vue de l'émetteur pour dimensionner sa durée de vie tandis que la troisième explicite les paramètres qui entrent en jeu du côté du récepteur, notamment avec plusieurs émetteurs.

4.1 Implémentations matérielles et algorithmiques

Afin d'avoir des ordres de grandeurs pertinents concernant le temps occupé par un algorithme de sécurité sur un système embarqué, un environnement matériel spécifique a été mis en place. Les durées mesurées dépendent donc du micro-contrôleur mais l'étude reste valable pour n'importe quel autre support matériel. Il conviendra en effet de réaliser ces mêmes mesures sur d'autres micro-contrôleurs. La définition des variables reste indépendante du support.

Le micro-contrôleur utilisé est le PIC32MX795F512L de Microchip. Associé à l'environnement de développement MPLab X et au *framework* Harmony 2.0, ce processeur possède les librairies nécessaires pour exécuter les algorithmes décrits dans la partie 2.2 (MD5, SHA-1, SHA-2, AES-CBC, AES-CTR, RSA, ECC).

Le choix d'utiliser ces librairies est motivé par l'optimisation de ces dernières sur la plate-forme cible en terme de temps et d'espace occupé. Surtout, une solution éprouvée et testée par une communauté est généralement plus sécurisée qu'une

solution faite soi-même [1]. De plus, l'utilisation de telles librairies rapproche cette étude de la réalité industrielle des systèmes embarqués.

Enfin, pour le lecteur curieux de reproduire le montage décrit ci-après, ces librairies sont directement accessibles sur le *framework* Harmony 2.0 de MPLab X.

4.1.1 Montage expérimental

La figure 4.1 propose un schéma du montage. Le micro-contrôleur PIC32MX795F512L est soudé sur un circuit imprimé. L'une des sorties du micro-contrôleur est dédiée à un signal possédant deux états :

— 1 (pour actif) lorsqu'un algorithme de sécurité est en cours d'exécution.

— 0 (pour inactif) lorsqu'aucun algorithme de sécurité n'est en cours d'exécution.

Un oscilloscope mesure cette sortie (CH2 sur la figure 4.1) afin d'obtenir la durée d'exécution de l'algorithme de sécurité testé. La sortie en question est contrôlée au niveau du code du *firmware* : l'instruction d'activation est exécutée juste avant l'algorithme de sécurité, et l'instruction de désactivation est exécutée juste après.

Trois résistances de 1 Ohm (pour un total de 3 Ohms donc) au niveau de l'alimentation du circuit imprimé permettent de brancher une deuxième sonde de l'oscilloscope et de mesurer le voltage (CH1 sur la figure 4.1). Grâce à l'équation $I = U/R$ il est possible d'en déduire l'intensité du courant d'entrée.

La mesure de consommation lors du signal actif est de 33.3 mA. Cette consommation est constante peu importe l'algorithme utilisé. Cela s'explique par la propriété *mono-thread* du micro-contrôleur : pour un algorithme de sécurité, le processeur est utilisé à pleine puissance en continu.

Des mesures de consommation différentes peuvent être obtenues en ajoutant des modules de sécurité sur le circuit imprimé, comme un module d'accélération matérielle par exemple. La durée de certains algorithmes pourraient être réduite mais la consommation totale du système embarqué plus importante car un nouveau

1. Plus précisément, l'auteur de ce document n'a pas la prétention de considérer son propre travail de quelques mois comme de meilleure qualité que les travaux d'une troupe d'ingénieurs, techniciens et testeurs sur plusieurs années. Il ne faut pas non plus croire qu'une solution open-source est totalement fiable. Des failles ou des portes dérobées existent, comme l'ont montré les affaires Crypto AG (porte dérobée sur des équipements de chiffrements privés) et HeartBleed (faille sur la librairie OpenSSL). Ces deux affaires montrent davantage l'intérêt des communautés *open source* et le manque de moyens de ces dernières, d'où l'initiative du secteur privée de les soutenir avec le projet Core Infrastructure Initiative auquel prennent part Microsoft, Google, Amazon Web Service, Facebook...

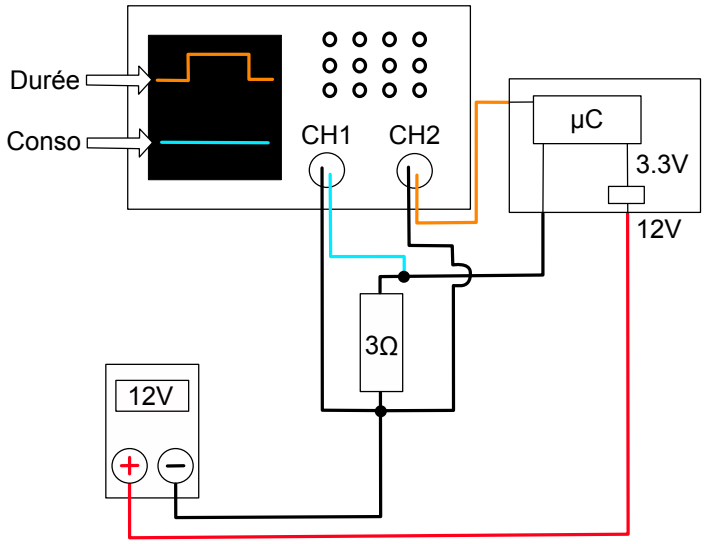

Fig. 4.1.: Schéma du montage expérimental

composant est utilisé. Cette piste a cependant été ignorée volontairement pour se concentrer sur les différences entre algorithmes et non sur les différences entre supports matériels.

À noter que le micro-contrôleur choisi (PIC32MX795F512L) est relativement puissant dans les gammes proposées par Microchip. Les résultats mesurés et présentés sont donc à considérer d'un point de vue *optimiste* : les temps d'exécution pourront être moins rapides sur des micro-contrôleurs de gammes ou caractéristiques inférieures.

4.1.2 Performances temporelles des algorithmes

Les mesures présentées de manière graphique dans cette partie sont détaillées dans l'annexe A.2 sous forme de tableaux. Afin de mettre en évidence l'influence de la taille d'un message sur la performance de l'algorithme de sécurité, différentes tailles de données à chiffrer ont été choisis :

— 3 octets : cela correspond à une très petite donnée. Les informations comme l'en-tête protocolaire, l'anti-rejeu ou un identifiant seraient alors ajoutées non chiffrées, hachées ou signées dans le message. Cette valeur peut servir de témoin (il aurait été superflu de gérer un message vide de zéro octet).

— 32 octets : cela peut correspondre à une charge utile classique pour un simple capteur (identifiant du capteur, niveau de batterie, grandeur mesurée...)

— 64 octets : comme précédemment, mais cette taille peut mettre en avant certaines propriétés *escaliers* de quelques protocoles (voir ci-après).

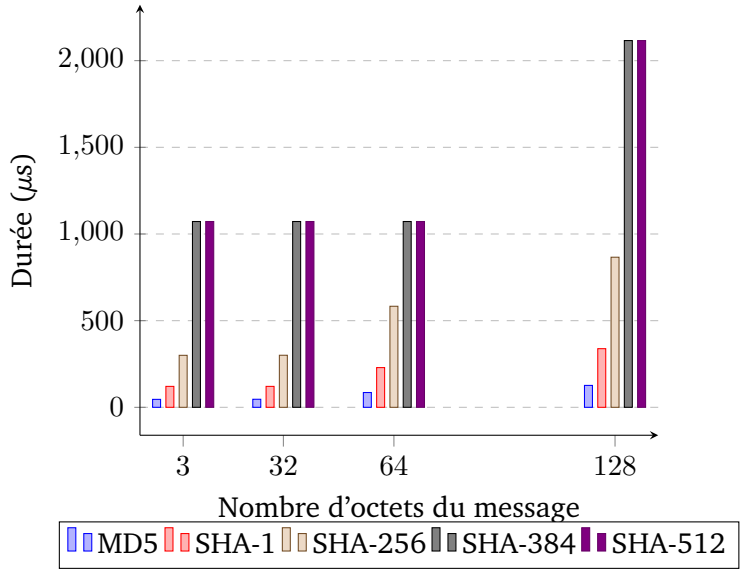

Fig. 4.2.: Mesures de performance - Hachage

— 128 octets : cette dernière valeur est souvent vue comme une limite pour de nombreux protocoles et au sein de certains algorithmes. Au-delà, cela mène souvent à un découpage en plusieurs sous-messages.

Des capteurs, notamment audio et vidéo, peuvent envoyer beaucoup plus de données mais ils ne sont pas considérés dans cette étude. Le raisonnement reste cependant le même et il suffirait d'augmenter la taille des messages et de refaire ces expériences.

Les algorithmes de hachage

Les algorithmes et variantes MD5, SHA1, SHA256, SHA384 et SHA512 donnent des temps d'exécution croissants en fonction du nombre d'octets à hacher, comme le montre la figure 4.2. Le détail des mesures est présenté dans l'annexe A.2, tableau A.1.

Pour une même longueur de messages à hacher, MD5 est plus rapide, tandis que SHA384 et SHA512 ont des durées identiques. Il faut cependant considérer les complexités du meilleur algorithme de cassage pour chacun des algorithmes de hachage. Ces complexités sont récapitulées dans la première partie dans le tableau 2.2. SHA512, qui a une plus grande complexité de cassage 2^{256} est également le plus long à être exécuté.

Ces mesures permettent également de mettre en avant certaines propriétés des algorithmes de hachage. C'est le cas en particulier pour les valeurs palières des algorithmes.

MD5, SHA1 et SHA256 utilisent des blocs de 512 bits (64 octets). Le dernier bloc utilise les 64 derniers bits pour coder la longueur du message. Théoriquement, à partir de 448 bits (56 octets), un deuxième bloc est nécessaire pour hacher le message. C'est pourquoi, pour un de ces algorithmes, la durée pour hacher 3 ou 32 octets est la même et pour 64 octets la durée est environ deux fois plus grande.

La logique est la même pour SHA384/512 car ces versions utilisent des blocs de 1024 bits (128 octets) avec la taille codée sur les derniers bits du dernier bloc. La valeur seuil apparaît donc sur le graphe pour 128 octets.

DES et 3DES

Les mesures effectuées sur le temps d'exécution des algorithmes DES et 3DES, présentés sur la figure 4.3, montrent des durées de chiffrement plus importantes pour 3DES (qui a une complexité d'attaque de 2^{112}) que DES (qui a une complexité d'attaque de 2^{41}).

À noter que pour un message de 3 octets, un bourrage est ajouté pour atteindre une taille minimale de 8 octets. La performance est donc identique pour une taille allant de 1 à 8 octets.

Encore une fois, ces mesures permettent de mettre en avant les propriétés des algorithmes DES et 3DES : le rapport de temps pour une taille équivalente de données à chiffrer est d'environ x3. Cela s'explique par la structure de 3DES qui exécute trois fois l'algorithme DES avec des clés différentes, comme expliqué dans la partie 2.2.

Les clés et vecteurs d'initialisation utilisés pour ces mesures sont décrits dans l' annexes A.1.1.

AES-CBC et AES-CTR

AES-128, AES-192 et AES-256 en mode d'opération CBC a des performances assez similaires qu'en mode d'opération CTR, bien que le mode CTR semble légèrement plus rapide (voir tableau A.2 de l'annexe A.2).

Les clés et vecteurs d'initialisation utilisés sont référencés dans l'annexe A.1.2.

Bien qu'étant plus robustes que les algorithmes de chiffrement précédents (2^{112} pour 3DES contre 2^{126} pour AES-128), la figure 4.3 montre que les familles d'algorithmes AES sont globalement plus efficaces. Le détail des mesures est présenté dans l'annexe

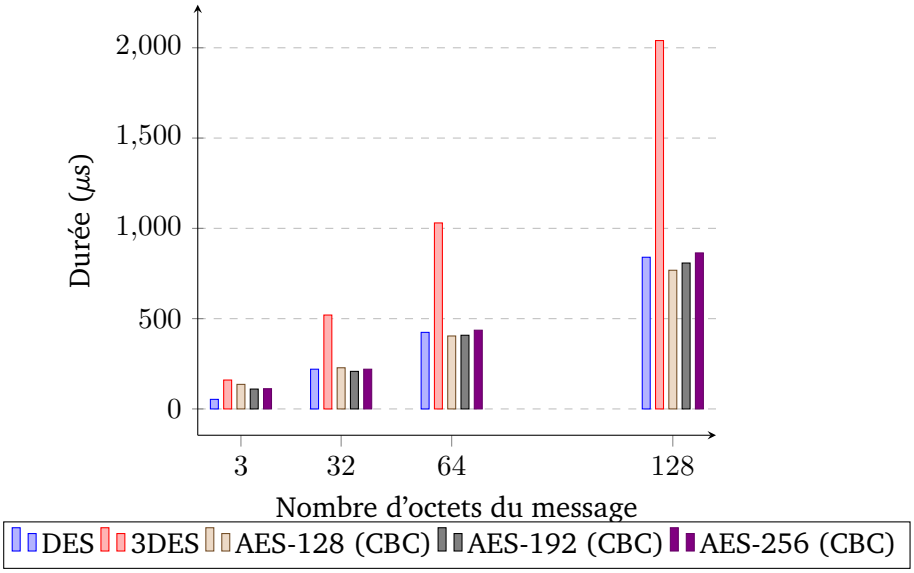

Fig. 4.3.: Mesures de performance - Chiffrement symétrique

A.2, tableau A.2. La règle intuitive qui semblait se dessiner qu'un algorithme plus robuste est plus long à exécuter n'est donc pas valide.

Cela peut s'expliquer par les propriétés mathématiques des algorithmes mais également par une optimisation plus performante de la librairie AES sur la plate-forme cible. De plus, AES, validé en tant que standard en 2001 contre 1977 pour DES a pu profiter de ces 24 années de recul de la communauté scientifique pour améliorer les méthodes de chiffrement [2].

Comme pour DES, un bourrage est effectué, mais cette fois pour atteindre 16 octets (taille d'un bloc utilisé dans AES). Les performances sont donc identiques pour des messages de 1 à 16 octets. De la même manière, des paliers de performances apparaîtraient tous les 16 octets, reflétant à travers les mesures la structure d'AES.

HMAC

Les mesures réalisées sur les algorithmes HMAC montrent, sur la figure 4.4, des évolutions de durées très similaires à la figure 4.2 sur les algorithmes de hachage. Le détail des mesures est présenté dans l'annexe A.2, tableau A.3 tandis que les clés utilisées sont décrites dans l'annexe A.1.3.

Ces tendances similaires s'expliquent naturellement en considérant que les HMAC utilisent les fonctions de hachage. Plus particulièrement, HMAC utilise deux fois

2. Les critères pour l'adoption du nouveau standard de chiffrement en 2001 comprenaient notamment une grande vitesse d'exécution.

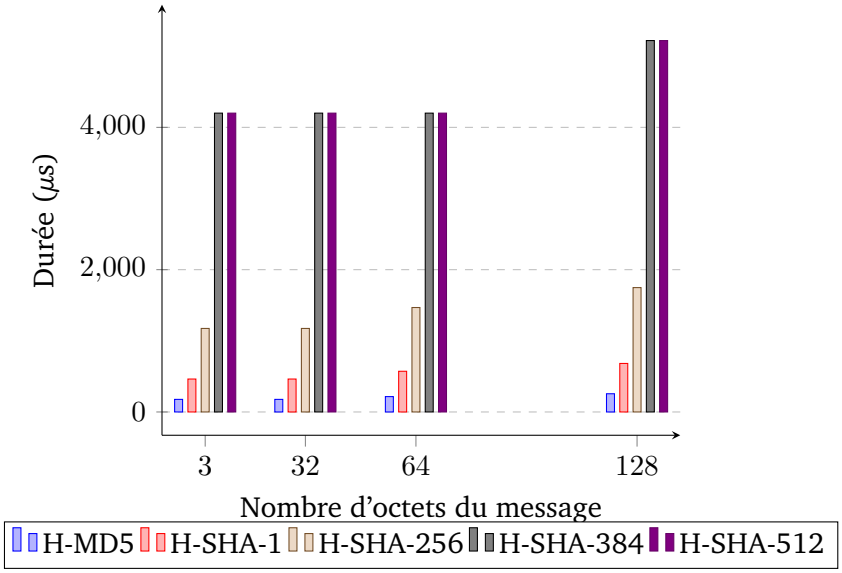

Fig. 4.4.: Mesures de performance - HMAC

une fonction de hachage : une première hachant le message, une deuxième hachant l'empreinte de la première fonction.

Or, comme observé sur les algorithmes de hachage, le temps d'exécution du calcul d'empreinte évolue selon la taille du message (avec un effet escalier lié à la taille des blocs).

En considérant N le nombre de blocs générés par le message m. La fonction HMAC va effectuer le hachage de $N + 3$ blocs (en supposant que K est inférieur à la taille d'un bloc) :

— N + 1 blocs générés par $(K \oplus i\_pad)||m$.

— 2 blocs générés par $(K \oplus o\_pad)||h$ (avec h l'empreinte précédente).

Cela donne un rapport de $(N + 3)/N$ entre les durées HMAC et les durées de fonctions de hachage. En particulier, les durées HMAC sont quatre fois plus lentes que les durées de hachage lorsque la taille du message est inférieure à la taille d'un bloc (4 bloc contre 1).

Autre exemple, pour un message m de 64 octets avec MD5 (dont la taille d'un bloc est de 64 octets), m va générer deux blocs [3]. La première étape HMAC va hacher 3 blocs et la deuxième 2 blocs.

3. La démonstration de calcul du rapport est ici simplifiée, mais il faut prendre un compte la longueur de l'encodage de la longueur du message placée en fin du dernier bloc). Cette longueur est ce qui provoque 2 blocs pour un message de 64 octets avec MD5 alors que la taille d'un bloc pour cet algorithme est de 64 octets

— MD5 : deux blocs, ce qui donne 85,4μs, environ deux fois plus de temps que pour un seul bloc (45,9μs).

— HMAC-MD5 : 5 blocs, ce qui donne 216μs (environ 5 fois plus de temps que le hachage MD5 d'un seul bloc)

Dans ce cas le rapport est bien $5/2$.

Enfin, dernier exemple avec un message de 128 octets, 3 blocs sont générés pour MD5 (126,4 μs) et 6 pour HMAC-MD5 (256 μs) donnant un rapport $(N+3)/N =$ x2.

Chiffrement et signature asymétriques

Les clés utilisées pour ECC sont détaillées en annexe A.1.5 tandis que les clés RSA (1024 et 2048) sont affichées en hexadécimal dans l'annexe A.1.4 pour permettre de refaire l'expérience si nécessaire.

Les solutions de chiffrement asymétrique ont été mesurées de la même manière, mais les résultats ont été naturellement différents pour le chiffrement et le déchiffrement comme le montre la figure 4.5. Le détail des mesures est présenté dans l'annexe A.2, à travers les tableaux A.4 (chiffrement) et A.5 (déchiffrement).

En effet, bien que l'opération soit la même, les clés pour le chiffrement (clé publique) et pour le déchiffrement (clé privée) sont différentes[4]. En particulier, bien que codées sur le même nombre de bits, les clés n'ont pas la même valeur. Une clé de valeur plus petite pour le chiffrement entraînera un nombre de calcul moindre que la valeur de la clé associée pour le déchiffrement, cette dernière pouvant être beaucoup plus grande.

Contrairement aux autres systèmes de sécurité, une telle différence causée par la clé implique une attention particulière sur le choix et l'utilisation des clés de chiffrement et de déchiffrement.

Il est courant que la clé utilisée pour le chiffrement sur un système embarqué soit plus petite que la clé de déchiffrement utilisée sur un système plus puissant (serveur, PC industriel...). Cela permet d'améliorer la complexité calculatoire pour le système embarqué au détriment du récepteur (qui peut lui même laisser le soin de la vérification à un serveur).

4. Pour la signature, la clé privée est celle utilisée pour signer, tandis que la clé utiliser pour vérifier la signature est publique car tous les destinataires doivent pouvoir vérifier l'authenticité du message. La clé privée reste donc plus grande et la charge de calcul est alors portée par l'émetteur.

Cependant, cette pratique donne une indication sur la clé côté système embarqué : il est probable qu'elle soit de *petite* valeur. C'est pourquoi cette clé de petite valeur est souvent utilisée en guise de clé publique car plus facilement trouvable, tandis que la clé utilisée pour le déchiffrement est gardée privée : seul le détenteur de cette dernière pourra lire le message envoyé et chiffré avec la clé publique.

Ces mesures permettent de donner des ordres de grandeur sur le temps que prend un système embarqué avec le chiffrement asymétrique. En l'occurrence, RSA-1024 approche 160 ms pour le chiffrement et 1300 ms pour le déchiffrement, tandis que RSA-2048 est autour des 600 ms pour le chiffrement et 9560ms pour le déchiffrement.

Ces temps de calculs sont d'un tout autre ordre de grandeur par rapport aux algorithmes de chiffrement symétriques ou aux algorithmes de hachage. Ils sont difficilement utilisables en pratique comme le montrera la partie 5, surtout pour chiffrer l'ensemble des données. En revanche, une utilisation hybride peut être intéressante pour un émetteur qui peut également passer en mode *récepteur* : un premier échange grâce à un algorithme asymétrique permet d'échanger une *clé de session symétrique* qui sera utilisée pour chiffrer les prochains messages.

Cependant, les émetteurs ne seront pas considérés en mode *récepteur* pour le reste de l'étude, l'occasion de se focaliser sur le problème de l'autonomie de façon simplifiée.

4.2 Le dilemme du point de vue de l'émetteur

Les mesures de consommation précédemment présentées permettent d'associer une valeur temporelle aux algorithmes de sécurité et donc indirectement de consommation d'énergie. Cette valeur dépend également de la quantité d'information à sécuriser mais pour la suite de l'étude, seulement des messages de 128 octets seront considérés.

En considérant les capteurs sans fil, la sécurisation d'une donnée intervient lorsque celle-ci doit être envoyée. Cela correspond à un évènement.

4.2.1 La notion d'évènement

Pour les mesures interruptives (le déclenchement de la mesure est appelé **interruption**), la précision du capteur est cruciale mais pas toujours configurable. Si le

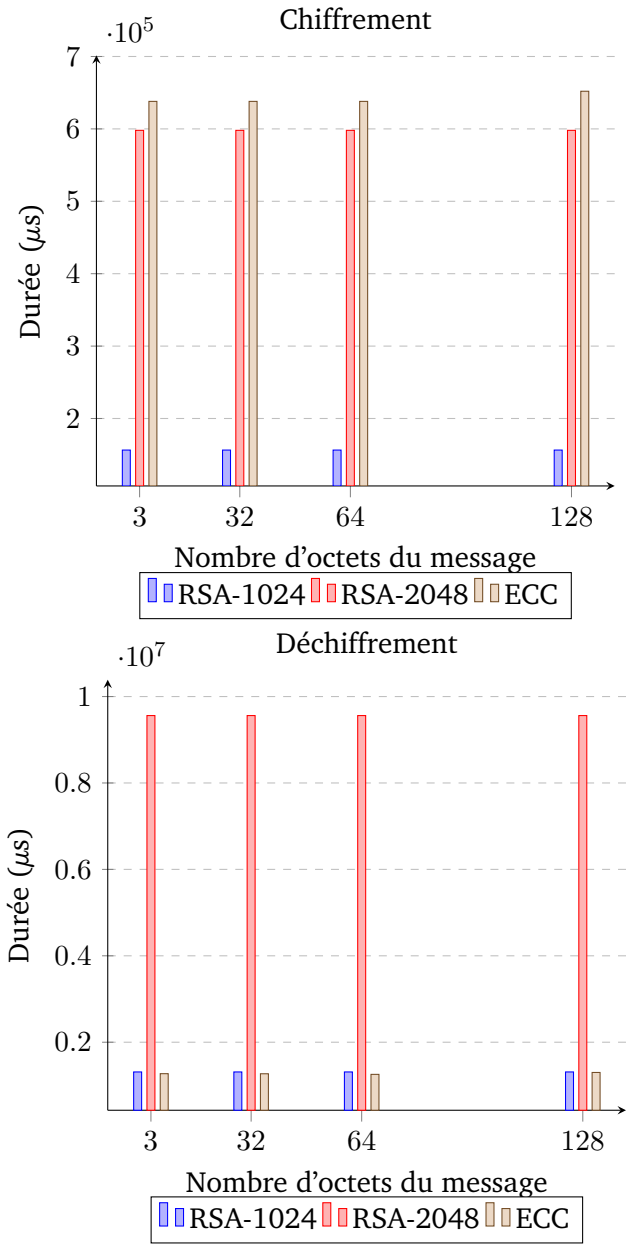

Fig. 4.5.: Mesures de performance - Chiffrement et déchiffrement asymétriques

concepteur souhaite mesurer une température, avec quelle précision souhaite-t-il l'avoir ?

La précision du capteur doit être au moins aussi grande que celle attendue par le concepteur du système. Dans ce cas, le concepteur peut choisir un seuil noté S_c.

Par exemple $S_c = 0.5$°C implique qu'une valeur sera traitée seulement si une différence de 0.5°C est détectée par rapport à une valeur de référence X_{ref} (la valeur précédente par exemple). Il s'agit de la notion d'évènement.

Afin de simplifier le reste de l'étude, la précision du capteur est considérée comme infinie, ce qui permet de se focaliser sur le paramètre contrôlé par le concepteur de système embarqué : S_c.

Un capteur peut également mesurer une variable discrète, comme un interrupteur qui donne deux valeurs possibles : 1 ou 0 pour fermé ou ouvert. Il s'agit là aussi d'un évènement. Le cas d'une variable discrète peut se généraliser puisqu'ici, le seuil serait alors $S_c = 1$. Dans les deux cas, il est donc possible de parler d'évènement déclenché selon le seuil choisi par l'utilisateur.

Une fois le seuil S_c dépassé entre une valeur de référence initiale X_{ref} et la mesure X_k, la mesure X_k est traitée (le traitement sera explicité plus tard). X_k devient alors la nouvelle valeur de référence X_{ref} pour la comparer aux prochaines mesures. Cette gestion du seuil apporte plusieurs avantages par rapport à des mesures continues :

— Réduction du nombre de traitements et d'envois de valeurs.

— Réduction de l'utilisation de la batterie du capteur si celui-ci est sur batterie.

— Évitement d'un ballottage de valeur en détectant une nouvelle valeur seulement quand un seuil a été franchi.

En considérant ce mode de mesure, comment déterminer le nombre de fois qu'un traitement aura lieu, ou encore, combien de fois le seuil sera franchi ?

L'équation 4.1 explicite ce dernier point en définissant λ comme la moyenne des probabilités qu'une valeur X_k soit différente de la valeur de référence X_{ref}, avec K valeurs différentes sur la période t. Cette différence doit être supérieure ou égale à S_c, seuil choisi par l'utilisateur.

$$\lambda = E[|X_k - X_{ref}| \geq S_c], \forall k \in [1, K] \tag{4.1}$$

L'indice k correspond à l'indice de la mesure entre 1 et K, X_1 étant la première mesure effectuée après X_{ref} (valeur de référence) et X_K étant la dernière mesure évaluée à la fin d'un intervalle de temps défini t.

Pour généraliser l'étude, une mesure mixte est considérée. C'est-à-dire qu'une interruption peut provoquer la récupération d'une mesure mais qu'une récupération forcée périodiquement permet de générer un signal de vie et s'assurer que le capteur fonctionne toujours même si la grandeur mesurée ne change que très peu.

Dans ce cas, t est égal à la durée T_{sigvie} entre deux envois forcés de la mesure (signal de vie) et K dépend de l'échantillonnage de la valeur, c'est-à-dire le nombre de mesures possibles pendant la durée T_{sigvie}.

Voici un exemple concret :

— $T_{sigvie} = 10s$, c'est-à-dire qu'un signe de vie est envoyé toutes les 10s et la valeur initiale X_{ref} est désormais égale à cette nouvelle valeur envoyée.

— $K = 10$, c'est-à-dire que la grandeur a changé 10 fois sur l'intervalle défini, soit un changement toutes les secondes.

— La grandeur mesurée est une température, et $X_{ref} = 22,5$°C.

— Le seuil choisi par l'utilisateur est $S_c = 0,2$°C.

λ représente le nombre moyen de dépassements de seuil détectés sur la période $t = T_{sigvie}$. Soient les grandeurs mesurées [22,5 ; 22,7* ; 22,6 ; 22,8* ; 22,5* ; 22,2* ; 22,3 ; 22,2 ; 22,3 ; 22,2]. Seules les mesures avec un astérisque provoquent une interruption car le seuil S_c a été dépassé. La valeur x_{ref} est alors remplacée par cette nouvelle mesure. Il y a donc eu dans cet exemple 4 interruptions.

L'utilisateur a une influence sur λ en jouant sur le seuil S_c, mais λ dépendra également de la volatilité de la grandeur mesurée et dépend donc d'une grandeur aléatoire X.

4.2.2 Les tâches du capteur sans fil

Récupération de la mesure

La récupération de la mesure a lieu selon la description de l'évènement faite précédemment. La consommation énergétique et la durée de lecture d'un capteur dépend donc du matériel choisi. Cette consommation sera notée C_{mes}, la durée sera notée T_{mes} et elles dépendent du choix du matériel.

Le seuil du capteur S_c dépend du choix du concepteur et a pour valeur limite la précision du capteur.

La durée entre deux signaux de vie T_{sigvie} dépend du choix du concepteur. Sauf interruption, le système est considéré comme en veille pendant T_{sigvie}.

La volatilité de la grandeur mesurée dépend de l'environnement où fonctionnera le capteur. λ, défini dans l'équation 4.1 dépend donc d'une variable aléatoire influencée par le choix du seuil S_c du concepteur et l'environnement du capteur. Pour la suite, cette valeur sera choisie arbitrairement mais il s'agira d'affiner cette valeur en fonction de la volatilité de la grandeur mesurée dans son environnement.

Traitement de la grandeur mesurée

Localement, c'est-à-dire sur le système embarqué, la grandeur mesurée peut subir un traitement permettant :

— d'interagir avec l'utilisateur (affichage sur un écran, clignotement d'une LED, déclenchement d'une alarme...),

— d'effectuer une action (actionner une machine dans le cas d'un actionneur, changer la position d'un interrupteur...),

— créer une donnée supplémentaire à partir de la grandeur mesurée (moyenne des dernières données...).

Ce traitement consomme du temps noté T_{trait} et de l'énergie consommée notée C_{trait}.

Protection de la donnée

La donnée traitée ainsi que des données additionnelles peuvent ensuite être protégées en vu d'un envoi. Pour ce faire, le concepteur peut choisir de :

— Hacher ces données. Cela prendra une durée T_{hash} et consommera C_{hash}. Si le concepteur ne souhaite pas hacher ces données, alors $T_{hash} = 0$.

— Chiffrer ces données. Cela prendra une durée T_{chiff} et consommera C_{chiff}. Si le concepteur ne souhaite pas chiffrer ces données, alors $T_{chiff} = 0$.

— Authentifier ces données. Cela prendra une durée T_{auth} et consommera C_{auth}. Si le concepteur ne souhaite pas authentifier ces données, alors $T_{auth} = 0$.

Envoi de la donnée

L'envoi d'une donnée peut nécessiter une antenne intégrée ou un composant externe (antenne RF, carte GSM, carte Wifi...).

Le consommation C_{envoi} et la durée de l'envoi $Tenvoi$ peuvent donc varier en fonction de la technologie et du matériel utilisés.

Répétition de l'envoi du message

Selon la situation, principalement dans des communications non acquittées, plusieurs envois peuvent avoir lieu. Ce nombre de répétitions est choisi par le concepteur et sera noté Nb_{rep} [5].

La répétition a lieu après un intervalle de temps choisi T_{rep}, auquel un aléa est ajouté pour chaque répétition. Cela permet de réduire la probabilité d'avoir une collision si deux messages avaient été émis en même temps par deux émetteurs.

Passage en veille

Une fois le message envoyé et répété, le capteur peut passer en veille pour une durée T_{veille}. Le système embarqué passe ses périphériques et son micro-contrôleur en veille plus ou moins profonde [6].

La veille a l'avantage de consommer très peu d'énergie, notée C_{veille}. La veille dure T_{veille} où $T_{veille} = T_{sigvie}$ dans le cas où aucune interruption n'a lieu, mais l'interruption faisant sortir de veille le système embarqué, ce temps peut être plus court. Autrement dit : $0 < T_{veille} \leq T_{sigvie}$.

4.2.3 Quelques exemples

Un capteur permettant un relevé de mesure et une antenne permettant l'envoi sont associés au miro-contrôleur, formant ainsi le système embarqué. présents. L'envoi consomme davantage de courant : cela peut varier selon la puissance du signal d'émission, ici 0 dB. Ce paramètre est considéré comme fixe dans le cadre de l'étude [7]. L logique est identique pour la lecture d'un capteur : la consommation et la durée peuvent varier selon les caractéristiques du capteur.

5. Il a été vu par exemple que le protocole EnOcean utilise une communication non acquittée. La répétition des messages envoyés permet d'améliorer la qualité de service (QoS) du protocole.

6. Une veille trop profonde requiert d'éteindre les périphériques, y compris le capteur qui déclenche l'interruption et le réveil du micro-contrôleur. Le mode de veille choisi doit donc correspondre à un mode où le micro-contrôleur peut être réveillé par un périphérique et par un *timer* interne.

7. Plus précisément, changer la puissance d'émission peut être vu comme changer la portée du signal, dans les limites de la technologie utilisée. Une étude approfondie serait nécessaire pour observer comment la variation de puissance du signal influence la portée de ce dernier et l'énergie consommée pour l'émetteur.

Un cycle correspond à l'enchaînement des tâches décrites précédemment. Une grandeur fixe ne déclenchera jamais d'interruption et seul le signal de vie impactera l'autonomie du système embarqué.

Ne pas considérer les interruptions permet alors d'avoir les performances maximales de l'émetteur pour les paramètres choisis, à savoir T_{sigvie}, Nb_{rep} et le niveau de sécurité souhaité.

Pour mieux prendre en compte l'influence de ces éléments sur un cycle de fonctionnement d'un système embarqué, des exemples sont donnés en annexe B et permettent d'en déduire des informations utiles pour l'évaluation des performances du système embarqué.

Ces exemples sont donnés pour un et un seul évènement, avec un temps de veille correspondant donc à T_{sigvie}. A partir de ces informations, le concepteur peut dimensionner la batterie pour atteindre l'autonomie souhaitée.

C'est pourquoi, pour chaque exemple, la capacité de la batterie est également présentée dans les tableaux de l'annexe B. Cette capacité est notée E_{batt} et est fixée à 155 mAh. L'autonomie est calculée en heures et en jours à partir de la consommation d'un cycle et de la durée d'un cycle. Pour rappel : ces exemples ne considèrent pas les interruptions, donnant une valeur d'autonomie dans des conditions idéales.

Exemple 1

— Pas de sécurité

— $T_{sigvie} = 10000$ ms

— $Nb_{rep} = 0$

Avec une telle configuration, le système a une autonomie maximale de 354 jours (tableau B.1). L'émission radio représente 57% de a consommation d'un cycle, tandis que la lecture et le traitement de la donnée en représente 21% et la veille représente les 22% restant (figure B.1).

Le concepteur souhaitant améliorer les performances de son système embarqué pourra alors se concentrer sur le choix des composants ou sur la puissance d'émission.

Exemple 2

— Chiffrement (AES-256) et hachage avec authentification symétrique (HMAC-SHA-512)
— $T_{sigvie} = 10000$ ms
— $Nb_{rep} = 1$

Pour ce cas, le système a une autonomie maximale de 133 jours (tableau B.2). L'émission radio représente 22% de la consommation d'un cycle, tandis que la lecture et le traitement de la donnée en représentent 8% (idem pour la veille). La répétition consomme 21% car le message est renvoyé une fois, et la sécurité implémentée représente 41% de la consommation totale d'un cycle (figure B.2).

Ici, le concepteur souhaitant améliorer les performances de son système embarqué pourra avoir un impact plus important sur l'autonomie en se focalisant sur la sécurité ou l'émission en particulier (la répétition étant également impactée par l'émission).

Exemple 3

— Hachage (SHA-512), chiffrement (AES-256) et signature (RSA-1024)
— $T_{sigvie} = 1000$ ms
— $Nb_{rep} = 2$

Enfin, dans cet exemple, le système a une autonomie maximale de 36 heures (tableau B.3). L'émission radio représente 2% de la consommation d'un cycle, tandis que la lecture et le traitement de la donnée ainsi que la veille sont négligeables par rapport aux autres consommations. La répétition consomme 4% car le message est renvoyé deux fois, et la sécurité implémentée représente 94% de la consommation totale d'un cycle (figure B.3).

Ici, le concepteur souhaitant améliorer les performances de son système embarqué se focalisera donc sur les algorithmes de sécurité utilisés. La grande part associée à la sécurité dans ce cycle est liée principalement à RSA-1024 qui utilise la majorité du temps de calcul hors veille.

4.3 Le dilemme du point de vue du récepteur

Dans cette étude, le récepteur est également un système embarqué équipé du micro-contrôleur PIC32MX795F512L. Il arrive en effet que l'émetteur et le récepteur

partage une base électronique commune pour mutualiser les librairies et environnements de développements. Dans le cas extrême où la carte électronique est la même pour le récepteur et l'émetteur, une réduction des coûts lors de l'industrialisation des systèmes embarqués peut avoir lieu.

4.3.1 Les tâches du récepteur

Le récepteur doit réaliser une série d'actions lors de la réception d'un message. Ces actions peuvent être vues comme un reflet des actions de l'émetteur.

Réception d'un message

Le récepteur, équipé de son antenne interne ou d'un composant externe (antenne RF, carte GSM, carte Wifi...) réceptionne un message qui est ensuite récupéré par le micro-contrôleur. Cette opération prends un temps T_r et consomme C_r.

Déchiffrement et vérification du message

Une fois le message récupéré, selon les choix ayant été faits côté émetteur, il faut alors :

— Déchiffrer le message. Cela prendra une durée $T_{dechiff}$ et consommera $C_{dechiff}$. Si l'utilisateur ne souhaite pas déchiffrer ces données, alors $T_{dechiff} = 0$. Cela peut se justifier si le déchiffrement intervient sur un autre système par exemple ; autre système auquel le récepteur transmettra le message chiffré.

— Vérifier l'intégrité des données. Cela prendra une durée T_{hash} et consommera C_{hash}. Si l'utilisateur ne souhaite pas vérifier l'intégrité des données, alors $T_{hash} = 0$. T_{hash} et C_{hash} sont identiques à ceux de hachage côté émetteur car l'opération est identique pour la vérification de l'intégrité (voir schéma 2.4).

— Vérifier l'authenticité des données. Cela prendra une durée T_{vauth} et consommera C_{vauth}. Si l'utilisateur ne souhaite pas vérifier ces données, alors $T_{vauth} = 0$. Encore une fois, les vérifications peuvent avoir lieu sur un autre système, plus performant.

Traitement de la donnée

Une fois les données récupérées et vérifiées, le récepteur peut effectuer un traitement local de celles-ci, comme pour l'émetteur.

Ici, un récepteur peut être considéré comme un noeud, c'est-à-dire un récepteur qui doit envoyer le message à une autre entité du réseau pour atteindre une destination finale où la donnée sera utile.

Dans ce cas, le transfert du message à une autre entité peut être considéré comme le traitement de la donnée du point de vue du récepteur. Cela permet de généraliser l'étude à des réseaux *mesh* et pas forcément uniquement aux réseaux en étoile.

Le traitement de la donnée prend un temps T_{rtrait} pour une consommation C_{rtrait}.

Attente d'un nouveau message

Une fois le traitement terminé, le récepteur attend un nouveau message. Cette attente peut se faire dans une veille peu profonde, permettant une bonne réactivité du récepteur. Un récepteur étant souvent associé à une source d'alimentation stable, la veille du récepteur ne sera pas considérée.

Pendant l'attente, le récepteur consommera C_{ratt}. Le délai d'attente dépend de la fréquence d'arrivée des messages. Cette fréquence d'arrivée dépend de l'évènement mesuré (voir définition de λ sur l'équation 4.1) et du nombre de répétitions d'un message par l'émetteur.

Dans le cas d'un réseau de capteurs sans fil, il est fort probable que le récepteur gère plusieurs émetteurs.

4.3.2 Gestion de plusieurs émetteurs

Le nombre de capteurs influencera forcément le récepteur sur la charge de ce dernier. Ce nombre est noté Nb_{cap}.

Pour ne pas complexifier le problème, il a été convenu que le récepteur peut traiter un message (*dans* le micro-contrôleur) tandis qu'un message est *en attente* dans l'antenne. Si un message est en attente et qu'un nouveau message est reçu, alors ce dernier est considéré comme perdu.

Ce problème est un problème de file d'attente et peut être modélisé avec des chaînes de Markov mais ne sera pas abordé dans ce document. La configuration "un message dans le micro-contrôleur, un message dans l'antenne" sera considérée.

Enfin, la qualité de service (QoS), représente le pourcentage de messages correctement reçus et traités par rapport au nombre total de messages envoyés par les émetteurs. Cette valeur est utile pour évaluer les performances d'un réseau.

4.4 Récapitulatif des variables et paramètres

Cette section récapitule les paramètres vus jusqu'alors et discute quelques pistes pour prendre en compte d'autres facteurs. Le lecteur est invité à y revenir pendant la lecture de la partie 5.

Ces paramètres dépendent principalement des choix du concepteur : choix des composants (micro-contrôleur, capteur...), choix du niveau de sécurité, choix du délai entre deux signes de vie...

Toutefois, certains paramètres dépendent de caractéristiques fonctionnelles et environnementales. λ par exemple, comme expliqué précédemment, dépend de l'environnement où évoluera le capteur, même si le concepteur peut influencer ce λ avec le seuil S_c. Par simplification, seul λ sera pris en compte.

En revanche, la durée et la consommation liées au traitement de la donnée, que ce soit sur l'émetteur ou le récepteur, dépendront davantage des fonctionnalités du système embarqué. Par exemple, un affichage sur un écran LCD consommera beaucoup plus que l'allumage d'une LED. Indirectement. Ces deux paramètres dépendent donc aussi du choix des composants.

4.4.1 Paramètres et variable d'un émetteur

Des paramètres supplémentaires peuvent être pris en compte, comme l'encombrement des composants ou encore le prix de ceux-ci. Ces contraintes sont à prendre en compte pour le concepteur de systèmes embarqués au moment des choix des composants. Elles ne prennent pas une part directe au problème du dilemme abordé mais agissent comme un filtre pour le concepteur lors du choix des composants. L'encombrement et le prix ne seront pas pris en compte dans l'étude.

La capacité de la batterie est prise en compte en tant qu'énergie uniquement. Il faut cependant garder en tête qu'exiger une capacité importante impliquera probablement un encombrement important de cette dernière.

Les paramètres concernant un émetteur sont récapitulés dans le tableau 4.1.

Variables	Description
C_{mes}	Conso lecture capteur
T_{mes}	Durée lecture capteur
C_{trait}	Conso traitement
T_{trait}	Durée traitement
C_{hash}	Conso hash
T_{hash}	Durée hash
C_{chiff}	Conso chiffrement
T_{chiff}	Durée chiffrement
C_{auth}	Conso authentification
T_{auth}	Durée authentification
C_{envoi}	Conso émission
T_{envoi}	Durée émission
Nb_{rep}	Nombre de répétitions
T_{rep}	Durée moyenne entre chaque répétition
T_{sigvie}	Durée avant prochain cycle
C_{veille}	Conso veille
λ (et S_c)	Nombre moyen de mesures sur une durée t
E_{batt}	Énergie de la batterie
T_{vie}	Durée de vie de l'émetteur (autonomie)

Tab. 4.1.: Paramètres d'un émetteur

4.4.2 Paramètres et variable d'un récepteur

Le récepteur peut effectuer un traitement supplémentaire et différent de la donnée par rapport à l'émetteur. Il possède donc ses propres paramètres.

De plus, si le récepteur utilise une électronique (et particulièrement un micro-contrôleur) différent de l'émetteur, les consommations et durées des algorithmes de sécurité peuvent différer de l'émetteur. C'est pourquoi le récepteur a également ses propres paramètres pour les algorithmes de hachage, déchiffrement et vérification de l'authentification.

Les paramètres concernant un récepteur sont récapitulés dans le tableau 4.2.

4.4.3 Paramètres et variable d'un réseau de capteurs

Une fois l'émetteur et le récepteur définis, il convient alors de dimensionner le réseau de capteurs. La qualité de service (QoS) est une donnée propre au réseau, et sera le résultat des différents paramètres évoqués précédemment.

Variables	Description
T_r	Durée de réception
C_r	Conso de réception
C_{rtrait}	Conso traitement des données
T_{rtrait}	Durée traitement des données
C_{rhash}	Conso hash
T_{rhash}	Durée hash
$C_{dechiff}$	Conso chiffrement
$T_{dechiff}$	Durée chiffrement
C_{vauth}	Conso authentification
T_{vauth}	Durée authentification

Tab. 4.2.: Paramètres d'un récepteur

Variables	Description
Nb_c	Nombre de capteurs
QoS	Qualité de service
T_{secu}	Complexité meilleure attaque

Tab. 4.3.: Paramètres d'un réseau de capteurs

Par exemple, un seul récepteur pour un très grand nombre de capteurs envoyant de nombreux messages provoquera une grande perte de messages et donc une QoS assez faible.

Justement, le nombre de capteurs est un élément supplémentaire à prendre en compte. Les capteurs peuvent mesurer des grandeurs différentes (par nature ou à cause de l'environnement de mesure). Chaque capteur pouvant être différent de par la grandeur mesurée, il est possible aussi de considérer chaque capteur comme entièrement différent, y compris au niveau de leur électronique. Dans un tel cas, chaque émetteur du réseau aura son propre jeu de paramètre.

Enfin, un niveau de sécurité requis pour les données circulant sur ce WSN peut être exigé par le concepteur ou son client. Noté T_{secu}, ce niveau de sécurité représente une complexité minimale d'attaque. Autrement dit, T_{secu} est le temps minimum de sécurisation de la donnée. Ce *temps* est en fait une complexité algorithmique que le concepteur peut rapporter aux temps de calculs possibles sur le marché (ordinateur performant, serveur dédié, super-calculateur... selon les capacités d'un attaquant éventuel).

Les paramètres concernant un WSN sont récapitulés dans le tableau 4.3.

Il est désormais possible de modéliser un tel WSN et d'utiliser ces paramètres définis pour simuler le comportement de celui-ci et déterminer l'évolution de la QoS en fonction des données d'entrée.

Résumé de la partie

Afin d'avoir des ordres de grandeurs concrets sur les durées et la consommation des algorithmes de sécurité, un montage expérimental a été proposé pour les mesurer. Cela a permis de mettre en évidence le lien entre la taille de la donnée à chiffrer et la durée du chiffrement et de mettre en avant les différences de performances calculatoires entre algorithmes.
Par ailleurs, le rôle d'un émetteur et d'un récepteur ont été clairement définis afin de mettre en lumière certains paramètres qui influenceront le WSN : le comportement de la grandeur mesurée, le seuil de déclenchement de la mesure, les durées et consommations des différentes tâches pour le récepteur et l'émetteur, le nombre de répétitions d'un message, la durée de la veille (entre deux signes de vie), le niveau de batterie des émetteurs ou encore le nombre d'émetteurs.

5 Modélisation et simulations

> *La tour Eiffel... entièrement faite avec des allumettes, 346 422 exactement !*
>
> — **François Pignon**
> Le dîner de cons, réalisé par Francis Veber

5.1 Modélisation par réseaux de Petri

5.1.1 Rappels sur les réseaux de Petri

Bases

Pour représenter un système, les réseaux de Petri utilisent la notion de *places* (représentées par des cercles) et *transitions* (représentées par des rectangles). Les places et transitions sont connectées entre elles grâce à des arcs. Deux places ou deux transitions ne peuvent être connectées entre elles.

Une place peut contenir des jetons (*tokens*), représentant alors un nombre de ressources ou une condition à satisfaire. Ces conditions sont représentées par le poids de chaque arc, définis dans la fonction de valuation W.

Enfin, M est le vecteur de marquage des places à un instant t : il indique pour un instant donné le nombre de jetons dans chaque place. M_0 est alors appelé le marquage initial.

Voici un exemple de réseau de Petri $R = (P, T, W, M)$ avec :

- $P = \{p_1, p_2\}$
- $T = \{t_1, t_2\}$
- $W(p_1, t_1) = 1$, $W(t_1, p_2) = 1$, $W(p_2, t_2) = 2$, $W(t_2, p_2) = 2$
- $M_0(p_1, p_2) = (1, 1)$

Ce qui donne le réseau de Petri schématisé dans la figure 5.1 produite avec le logiciel GRIF 2011. À noter que pour alléger le schéma, les valuations égales à 1 sont omises

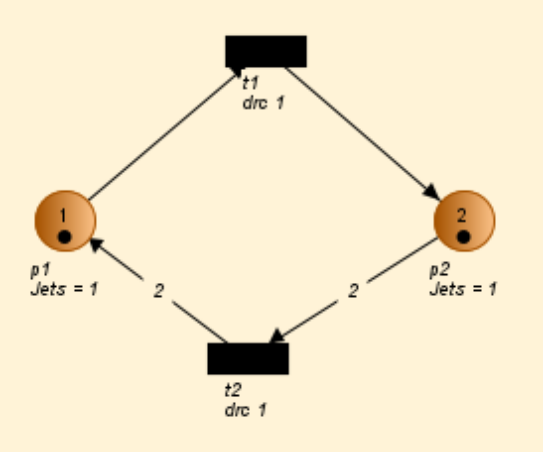

Fig. 5.1.: Exemple d'un réseau de Petri

(il s'agit d'une simple flèche en trait continu entre une place et une transition ou entre une transition et une place).

Une transition est effectuée lorsque ses conditions sont respectées. Par exemple, la transition t_2 demande 2 jetons pour s'activer et générera 2 jetons dans p_1. Dans l'exemple de la figure 5.1, chaque transition a une durée d'une unité de temps représentée par une fonction de Dirac.

Sources, puits et arcs inhibiteurs

Une source est une transition sans place en amont : elle produit des jetons pour la place suivante en aval. De manière analogue, un puits est une transition sans place en aval : elle consomme les jetons de la place en amont.

Un arc inhibiteur représente une règle qui spécifie que la transition t a lieu si et seulement si la place p associée à cet arc ne possède pas de jeton, autrement dit, si et seulement si $M(p) = 0$ et $W(p,t) = \emptyset$.

La figure 5.2 montre l'utilisation d'un tel arc. Dans ce schéma, l'arc inhibiteur empêche la source t_4 de créer un jeton supplémentaire si un jeton est déjà présent dans p_3. GRIF 2011 représente cet arc par un trait en pointillé associé à la valeur -1 [1]. Dans cette configuration, t_4 joue le rôle d'une source (ou d'un producteur : elle créé un jeton) et t_5 celui d'un puits (ou consommateur : elle consomme un jeton).

1. Une représentation plus usuelle d'un arc inhibiteur consiste en un trait plein avec un rond du côté de la place, indiquant une négation de l'arc.

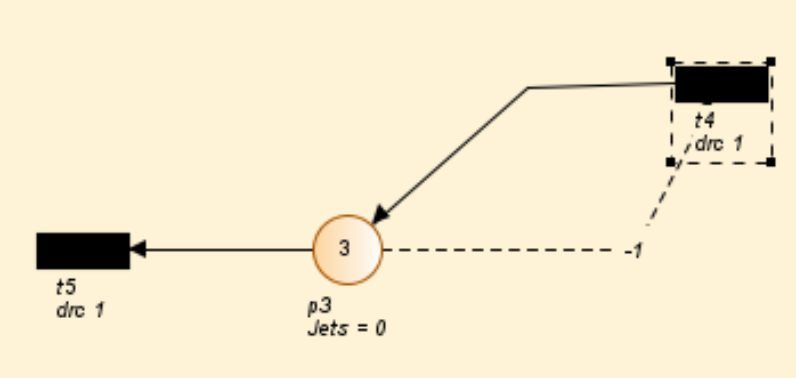

Fig. 5.2.: Arc inhibiteur

Réseaux de Petri temporisés

Dans l'exemple de la figure 5.2, la transition t_4 possède l'annotation *drc 1* qui symbolise une durée d'une unité de temps : il s'agit de la durée de la transition. Il s'agit ici de formaliser la temporalité au sein d'un réseau de Petri, tel que proposé dans [12]. GRIF 2011 permet de modéliser cela sous la forme mathématique d'une fonction de Dirac.

Il est également possible d'ajouter des *affectations* à des variables lors du passage d'une transition : cela est particulièrement pratique pour faire diminuer une valeur de batterie par exemple.

Toutefois, les transitions vues jusqu'alors sont déterministes, et cela implique donc que le réseau de Petri dans sa globalité est également déterministe : à partir du marquage initial, il est possible dans ce cas de déduire tous les marquages à chaque instant de la durée de la simulation du réseau de Petri.

Pour se rapprocher de la réalité où des évènements (la mesure d'une grandeur dans le cas de l'étude) sont aléatoires, une approche stochastique des réseaux de Petri est pertinente.

Réseaux de Petri stochastiques

Deux systèmes embarqués peuvent avoir des comportements différents selon les grandeurs mesurées, l'état des composants (et leurs défaillances potentielles) ou l'épuisement de la batterie.

Ces aspects dépendent de lois aléatoires et forment ensemble des processus stochastiques qui peuvent être modélisés dans le cadre des réseaux de Petri grâce à des transitions dépendant de telles lois tandis que d'autres transitions et places

dépendent de temporisations fixes ([11], [5]). Cependant, les défaillances et l'épuisement de la batterie (à force d'utilisation) ne seront pas abordés dans ce document, afin de simplifier le problème.

C'est pourquoi seule une transition représentant le déclenchement d'un évènement (réalisation d'une mesure) aura un comportement soumis à des aléas. Au moment de la simulation, un temps sera tiré selon une loi aléatoire, instanciant ainsi la durée de la transition.

En reprenant l'exemple 5.2, les transitions ont lieu de manière déterministe après une durée d'une unité de temps (représentée par une fonction de Dirac) mais t_4 pourrait également se déclencher aléatoirement en fonction de différentes lois aléatoires (loi exponentielle, loi uniforme...). Cet aspect sera notamment utilisé pour modéliser un capteur-émetteur.

Réseaux de Petri colorés

Un réseau de Petri coloré peut être utile lorsque plusieurs processus utilisent le même sous-processus. Au lieu d'un réseau de Petri composé de sous-réseaux répétitifs, il est possible de créer un seul sous-réseau réutilisé par les jetons des différents processus.

Afin de distinguer les jetons, ceux-ci sont différenciés (par forme, notation, ou couleur, d'où le nom de réseaux de Petri colorés).

Champs d'applications

La simplicité de modélisation des réseaux de Petri couplée à la puissance du formalisme proposée par leur niveau d'abstraction ont popularisé cette méthode dans plusieurs domaines, notamment des domaines où un partage de ressources associé à des processus concurrents a lieu. Sans être exhaustif voici une liste de domaines d'applications pour la modélisation par réseaux de Petri :

— Protocoles de communications : la modélisation de protocoles avec un réseau de Petri est utile pour mettre en avant les points de blocage.

— Contrôle de processus industriels : la gestion de ressources et d'approvisionnements de ceux-ci peut être représentée par des jetons générés de manière contrôlée par les transitions.

— Programmation concurrente : les problématiques de synchronisation sont aisément modélisables à l'aide des jetons et des transitions.

Pour la problématique concernant l'étude, les réseaux de Petri seront utilisés afin de modéliser le processus d'un émetteur et d'un récepteur. Une fois les modèles décrits, ceux-ci seront utilisés pour simuler le comportement d'un WSN selon les paramètres définis dans la partie 4.

Outils de modélisation

Le logiciel GRIF (Graphiques Interactifs pour la Fiabilité) développé par Satodev pour le compte de Total est exploité pour modéliser les réseaux de Petri décrits ci-après. En particulier, la version 2011 de GRIF est utilisée (licence accessible au sein du laboratoire où l'étude a été menée).

Le choix de ce logiciel a été motivé par la simplicité de construction d'un réseau de Petri. En plus d'un réseau de Petri classique, GRIF 2011 permet d'appliquer aux transitions des *affectations*, affectations qui seront utilisées pour prendre en compte la consommation énergétique des émetteurs par exemple. Mais il est également possible d'utiliser des *gardes* qui conditionnent l'application d'une transition : pour s'activer, une transition devra respecter le nombre de jetons nécessaires ainsi que les conditions définies dans les gardes de celle-ci.

De plus, GRIF 2011 permet de lancer des simulations de réseaux de Petri, ce qui est particulièrement intéressant lorsque des processus stochastiques entrent en jeu. Pour *lisser* l'aléatoire, l'outil de calcul Moca permet de lancer plusieurs *histoires* : il s'agit d'expériences avec une concrétisation aléatoire (potentiellement différente d'une histoire à une autre) donnant des évènements différents. Moca moyenne ces résultats sur l'ensemble des histoires pour donner le comportement moyen du système. Cet outil de calcul sera particulièrement utile pour exploiter les résultats de la simulation.

Cependant, GRIF utilise l'heure comme unité de mesure du temps. Cela s'explique par l'utilisation principale du logiciel : le but est de mettre en avant des problèmes de fiabilités avec des défaillances ayant lieu sur le long terme. Or, pour le problème étudié, l'unité de temps est la milliseconde, unité de temps trop faible pour être prise en compte par GRIF. C'est pourquoi, pour la suite de la modélisation, l'heure de GRIF est considérée comme une milliseconde. Ainsi, lorsque l'interface GRIF affiche une durée de simulation de 10 000 000 d'heures, il faut en fait comprendre que la simulation dure 10 000 000 ms (soit 2,78 h).

La gestion en mémoire des variables dans GRIF ne permet pas d'aller beaucoup plus loin dans la durée de la simulation. Il est donc impossible d'arriver à la fin de l'autonomie d'un émetteur. En effet, comme expliqué dans la partie 4.2.2, le

temps d'utilisation d'un émetteur peut aller de quelques heures dans le pire des cas, à plusieurs milliers d'heures dans le meilleur des cas. Toutefois, la tendance de consommation pourra déjà apparaître au bout de la durée de la simulation.

5.1.2 Modélisation de l'émetteur

Les tâches de l'émetteur ont été détaillées dans la partie 4.2.2 et se retrouvent dans les places et transitions du réseau de Petri de l'émetteur, schématisé sur la figure 5.3 à l'aide de GRIF. Chaque émetteur est numéroté, ainsi P_i représente les places du réseau de Petri de l'émetteur i.

La définition du réseau de Petri est alors :

- $P_i = \{p_{event_i}, p_{mesure_i}, p_{trait_i}, p_{secu_i}, p_{envoi_i}, p_{fin_i}, p_{veille_i}\}$
- $T_i = \{t_{ping_i}, t_{mesure_i}, t_{eveil_i}, t_{trait_i}, t_{secu_i}, t_{envoi_i}, t_{repet_i}, t_{out_i}, t_{veille_i}\}$
- W :
  - $W(t_{ping_i}, p_{event_i}) = 1$
  - $W(t_{mesure_i}, p_{event_i}) = 1$
  - $W(p_{event_i}, t_{eveil_i}) = 1$
  - $W(t_{eveil_i}, p_{mesure_i}) = 1$
  - $W(p_{mesure_i}, t_{trait_i}) = 1$
  - $W(t_{trait_i}, p_{trait_i}) = 1$
  - $W(p_{trait_i}, t_{secu_i}) = 1$
  - $W(t_{secu_i}, p_{secu_i}) = Nb_{rep}$
  - $W(p_{secu_i}, t_{envoi_i}) = 1$
  - $W(t_{envoi_i}, p_{envoi_i}) = 1$
  - $W(p_{envoi_i}, t_{envoi_i}) = \emptyset$
  - $W(p_{envoi_i}, t_{repet_i}) = 1$
  - $W(t_{repet_i}, p_{fin_i}) = 1$
  - $W(p_{fin_i}, t_{out_i}) = Nb_{rep}$
  - $W(p_{fin_i}, t_{veille_i}) = Nb_{rep}$
  - $W(t_{veille_i}, p_{veille_i}) = 1$
  - $W(p_{veille_i}, t_{eveil_i}) = 1$

Où chaque place correspond à un état du système :

- p_{event_i} : Dans cet état, un évènement a eu lieu suite à une interruption (t_{mesure_i}) ou suite à un *ping* (t_{ping_i}) et permettra à l'émetteur de se réveiller (t_{eveil_i}).

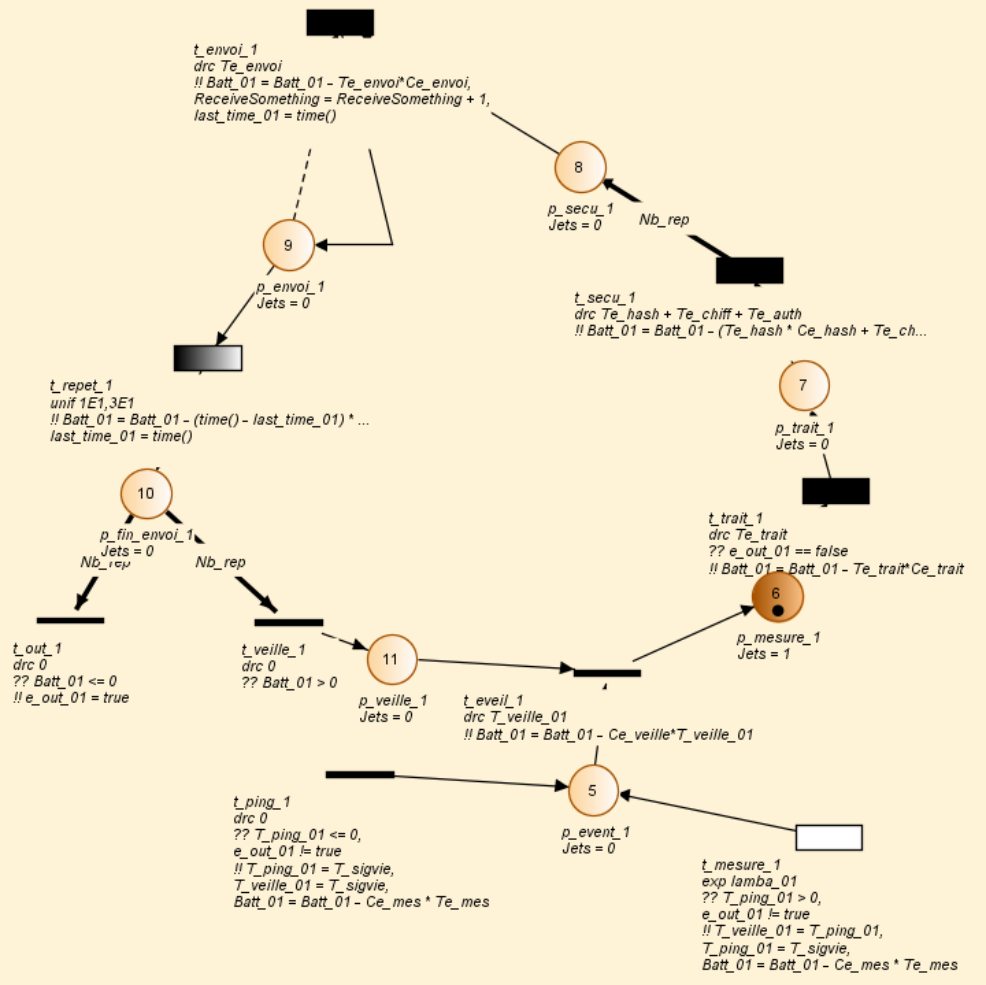

Fig. 5.3.: Modélisation de l'émetteur à l'aide des réseaux de Petri

— p_{mesure_i} : L'émetteur est réveillé, la mesure est récupérée et la donnée est prête à être traitée (t_{trait_i}).

— p_{trait_i} : La donnée a été traitée et le message est prêt à être sécurisé (t_{secu_i}).

— p_{secu_i} : Le message est sécurisé et est prêt à être envoyé Nb_{rep} fois.

— p_{envoi_i} : Un message a été envoyé et l'émetteur est en attente d'une temporisation (t_{repet_i}) avant la prochaine répétition.

— p_{fin_i} : Le message (une des répétition) est envoyé, l'émetteur attend la fin des envois des Nb_{rep} répétitions. S'il n'y a plus d'énergie dans la batterie, l'émetteur s'éteint (t_{out_i}) sinon il passe en veille (t_{veille_i}).

— p_{veille_i} : L'émetteur est en veille, et attend qu'un évènement ait eu lieu, c'est-à-dire qu'un jeton apparaisse dans la place p_{event_i}.

Et où chaque transition correspond à une action du système :

— t_{ping_i} : Cette transition produit un jeton pour la place p_{event_i} après la durée T_{sigvie} si aucune interruption n'a eu lieu (grâce à la transition t_{mesure_i}). Dans ce cas, la transition t_{eveil_i} durera T_{veille_i}. Cette variable sera mise à jour grâce à l'affectation $T_{veille_i} = T_{sigvie}$ qui a lieu de la transition t_{ping_i}. Une affectation supplémentaire a lieu et permet de prendre en compte la consommation générée par la mesure sur le capteur : $Batt_i = Batt_i - C_{mes} * T_{mes}$.

— t_{mesure_i} : Similaire à t_{ping_i}, cette transition génère un jeton dans p_{event_i} mais cette fois de manière non déterministe. En effet, un changement de la grandeur mesurée peut intervenir aléatoirement, et de façon plus ou moins récurrente selon la précision et le phénomène mesuré. Il s'agit du λ détaillé dans la partie 4.2.1, λ utilisé dans le cadre d'une loi exponentielle régissant la transition. Dans ce cas, la durée effective de la veille n'est pas égale à T_{sigvie} et une affectation a lieu pour que la transition t_{eveil_i} dure exactement T_{veille_i}. Chaque émetteur pouvant potentiellement mesurer des grandeurs et phénomènes différents, un indice est ajouté pour correspondre à l'émetteur donnant ainsi les paramètres λ_i.

— t_{eveil_i} : Transition permettant de réveiller le capteur après une durée de T_{veille} avec une affectation a lieu pour prendre la consommation de la veille : $Batt_i = Batt_i - C_{veille} * T_{veille_i}$.

— t_{trait_i} : Il s'agit de la transition permettant de traiter la donnée localement. Cela prend une durée T_{trait}. Une affectation a lieu afin de prendre en compte la consommation : $Batt_i = Batt_i - C_{trait} * T_{trait}$.

— t_{secu_i} : Cette transition permet d'intégrer aux messages les éléments de sécurité (intégrité, confidentialité, authentification). L'affectation dépend donc du temps de ces trois éléments de sécurité : $Batt_i = Batt_i - (C_{hash} * T_{hash} + C_{chiff} * T_{chiff} + C_{auth} * T_{auth})$. Cette transition donne Nb_{rep} jetons correspondant au nombre de répétitions à envoyer pour la place p_{secu_i}.

— t_{envoi_i} : Cette transition permet d'envoyer une répétition de messages. L'affectation suivante permet de prendre en compte la consommation lors de l'envoi : $Batt_i = Batt_i - C_{envoi} * T_{envoi}$.

— t_{repet_i} : Il s'agit du délai de temporisation (de durée T_{rep}) entre chaque message répété. Cette durée varie selon l'aléa de répétition : cet aléa permet, en cas de collision de messages avec un autre émetteur, de réduire les chances d'une nouvelle collision avec la répétition. Entre chaque répétition, l'émetteur est en veille d'où une consommation assez faible : $Batt_i = Batt_i - C_{veille} * T_{rep}$. T_{rep} est aléatoirement choisi dans un intervalle donné $[10, 30]$ ms, d'où la loi uniforme choisie pour modéliser cette transition [2].

— t_{out_i} : Cette transition permet d'activer une variable indiquant que l'émetteur n'a plus de batterie et est donc éteint. Avec ce *flag* à *true*, les transitions t_{ping_i} et t_{mesure_i} sont désactivées, symbolisant un émetteur inactif.

— t_{veille_i} : Cette dernière transition permet de faire entrer l'émetteur en veille si toutes les répétitions ont été envoyées.

5.1.3 Modélisation du récepteur

De façon similaire à l'émetteur, les tâches du récepteur ont été détaillées dans la partie 4.3 et se retrouvent dans les places et transitions du réseau de Petri du récepteur, schématisées sur la figure 5.4 à l'aide du logiciel GRIF.

Le récepteur étant souvent sur une source d'énergie stable et pour simplifier le problème, la consommation ne sera pas prise en compte ici.

La définition du réseau de Petri est alors :

— $P_r = \{pr_{event}, pr_{buffer}, pr_{secu}, p_{standby}\}$

— $T_r = \{tr_{event}, tr_{perdu}, tr_{reception}, tr_{secu}, tr_{trait}\}$

— W :

  — $W(tr_{event}, pr_{event}) = 1$

  — $W(pr_{event}, tr_{perdu}) = 1$

  — $W(pr_{event}, tr_{reception}) = 1$

  — $W(tr_{reception}, pr_{buffer}) = 1$

  — $W(pr_{buffer}, tr_{secu}) = 1$

  — $W(tr_{secu}, pr_{secu}) = 1$

  — $W(pr_{secu}, tr_{trait}) = 1$

2. Le concepteur est libre d'adapter les bornes et la loi utilisée pour être plus proche de sa conception de WSN.

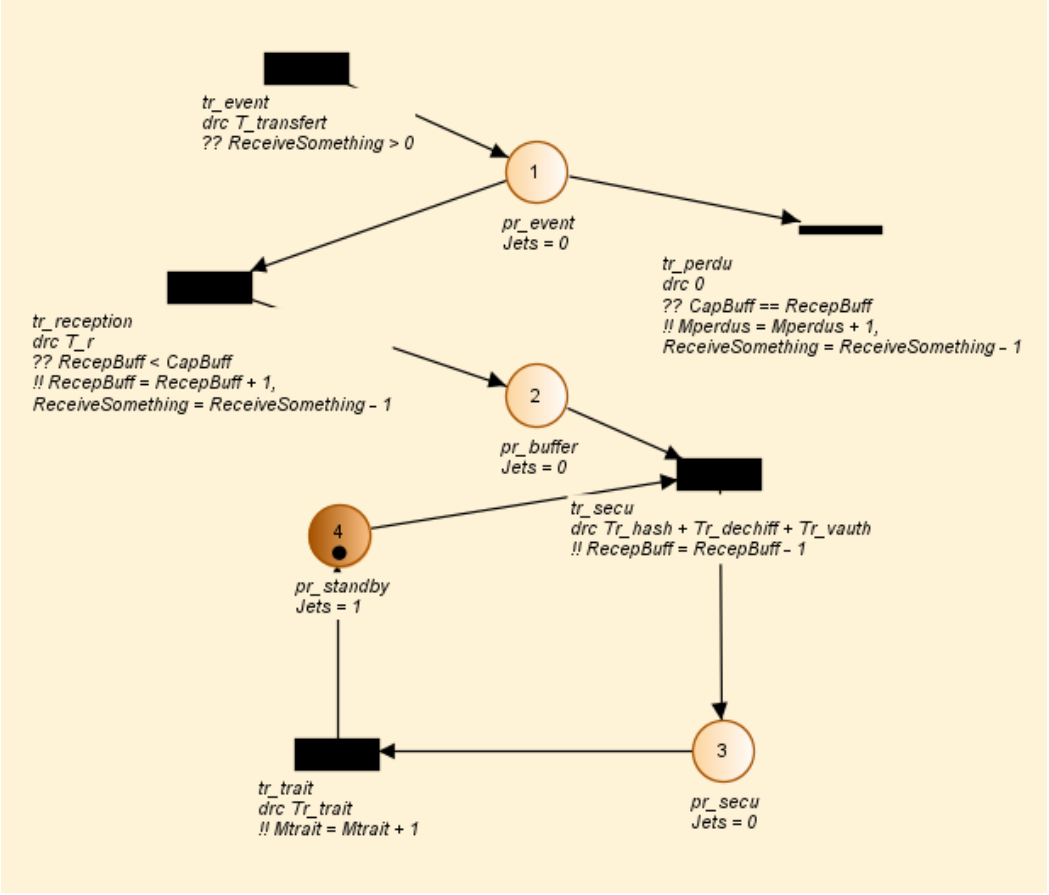

Fig. 5.4.: Modélisation du récepteur à l'aide des réseaux de Petri

— $W(tr_{trait}, pr_{standby}) = 1$

— $W(pr_{standby}, tr_{secu}) = 1$

Où chaque place correspond à un état du système :

— pr_{event} : Un message a été transmis et est en attente d'être traité. Si la capacité de la file d'attente $CapBuff$ n'est pas atteinte, la transition $tr_{reception}$ va concrétiser la réception, sinon le message sera considéré comme perdu par la transition tr_{perdu}.

— pr_{buffer} : La sécurité du message doit être traitée par la transition tr_{secu} dès que le récepteur sera disponible.

— pr_{secu} : La sécurité a été traitée. Il s'agit maintenant de traiter la donnée avec la transition tr_{trait}.

— $pr_{standby}$: Le récepteur est disponible, il attend qu'un message arrive, représenté par un jeton dans pr_{buffer}.

Et où chaque transition correspond à une action du système :

— tr_{event} : La transition a lieu lorsque la variable $ReceiveSomething$ est supérieure à zéro. Cette transition génère un jeton et représente la captation d'un message radio.

— tr_{perdu} : La transition indique que le message est perdu. Un message est considéré perdu si le récepteur ne peut pas stocker le message dans sa mémoire déjà occupée par d'autres messages. Le nombre de messages pouvant être stocké en mémoire est représenté par le paramètre $CapBuff$. Cette transition permet également de garder le compte du nombre de messages perdus grâce à l'affectation $M_{perdus} = M_{perdus} + 1$.

— $tr_{reception}$: Cette transition indique que le message est récupéré. Celle-ci a lieu si le nombre de messages prêts à être traités $RecepBuff$ est inférieur à la capacité de stockage en mémoire $CapBuff$.

— tr_{secu} : Cette transition a une durée dépendant du niveau de sécurité et des algorithmes utilisés. La durée de la transition est donc de $Tr_{hash} + Tr_{dechiff} + Tr_{vauth}$ pour respectivement : la durée de hachage, la durée de déchiffrement et la durée de vérification de l'authentification.

— tr_{trait} : Cette transition d'une durée arbitraire de $Tr_{trait} = 1$ ms représente un traitement local. Une fois le message traité, l'affectation $M_{trait} = M_{trait} + 1$ permet de garder le compte des messages traités.

Quelques remarques et améliorations

Ce modèle implique qu'un message envoyé est toujours capté de manière intelligible par l'antenne du récepteur. La réalité est bien différente, notamment dans un contexte industriel où des perturbations électromagnétiques venant des équipements proches peuvent brouiller le canal de communication. Cela entraîne une perte de message sur le canal de communication pouvant être représentée par une transition supplémentaire tr_{pertub} avec $W(pr_{event}, tr_{pertub}) = 1$, dépendant d'une loi exponentielle (dont le paramètre λ_{pertub} dépendrait de l'environnement électromagnétique).

Cet aspect a volontairement été omis de cette étude afin de focaliser celle-ci sur le dilemme entre la sécurité, l'autonomie (des émetteurs) et la disponibilité du récepteur. L'environnement dans lequel évolue le WSN est donc être considéré comme idéal, sans aucune perturbation.

La capacité de réception $CapBuff$ du récepteur peut influencer sur le pourcentage de perte de messages. Ce problème s'apparente davantage à un problème de fil d'attente, aisément modélisable à l'aide des chaînes de Markov, mais peut aussi être abordé dans ce contexte de réseau de Petri. Cependant, afin encore une fois de se focaliser sur le problème initial, $CapBuff$ sera fixé à 1 pour le reste de l'étude.

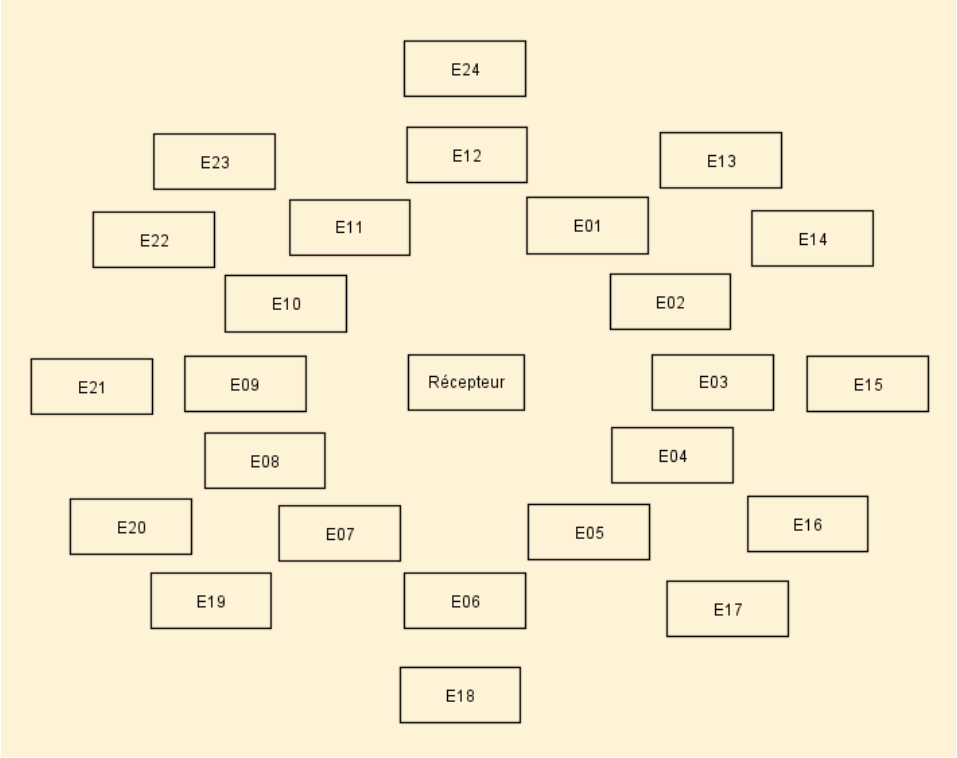

Fig. 5.5.: Modélisation en réseau de Petri d'un WSN

5.1.4 Un réseau de réseaux de Petri

Dans le modèle, représenter un WSN avec un réseaux de Petri revient à schématiser plusieurs émetteurs. GRIF 2011 ne permet pas de gérer des réseaux de Petri colorés, ce qui aurait grandement simplifié le schéma. Il a donc été décidé, à la place d'un réseau de Petri coloré, de dupliquer le réseau de l'émetteur 24 fois (pour faire varier le nombre d'émetteurs entre 1 et 24), d'où le schéma 5.5.

Les variables, paramètres, transitions et places possédant un indice i sont spécifiques à l'émetteur i, tandis que les autres paramètres (C_{veille}, T_{veille}...) sont communs à l'ensemble des émetteurs. Cela implique que tous les émetteurs sont identiques au niveau du matériel (processeur, fonctionnement...).

Pour faire varier le nombre d'émetteurs du réseau, des variables booléennes ont été définies, pour représenter leur état *actif* ou *inactif*.

Une valeur initiale de veille a été choisie pour chaque émetteur, permettant de simuler un branchement (ou allumage) différé des émetteurs. En effet, allumer les émetteurs en même temps générerait un message (répété $Nb_{rep} - 1$ fois) pour chaque émetteur et baisserait artificiellement la QoS. Pour éviter ce régime transitoire initial, chaque émetteur activé sera allumé à intervalle de 300ms.

Un fabricant de systèmes embarqués peut chercher à mutualiser son matériel pour réduire les coûts et temps de développement. C'est pourquoi les composants seront considérés comme identiques pour le récepteur et les émetteurs dans la suite de l'étude.

Il est toutefois pertinent de faire varier λ_i pour observer la qualité de service d'un WSN en fonction de la fréquence des évènements. Deux émetteurs peuvent surveiller une même grandeur (la température par exemple), mais selon l'endroit où la température est mesurée, λ_i peut différer d'un émetteur à l'autre. Bien que chaque émetteur possède son propre λ_i, deux λ_i identiques ne signifient pas que les émetteurs se réveilleront en même temps : cela signifie que les émetteurs mesurent tous les deux une grandeur suivant la même loi de probabilité (exponentielle en l'occurrence [3]) dépendante de λ_i.

Quelques remarques et améliorations

L'ensemble des messages échangés sont identiques en terme de structure, et d'une taille fixée à 128 octets, ce qui permet d'en déduire la durée de chiffrement grâce aux mesures effectuées dans la partie 4.

Dans ce modèle, un récepteur n'est que la destination finale d'un message. Il est cependant tout à fait envisageable de le transformer en noeud d'un réseau *mesh*. Pour ce faire, la transition tr_{trait} permettant le traitement d'un message côté récepteur peut être modifiée de la façon suivante : le traitement d'un message pour un noeud d'un réseau *mesh* correspond à la transmission du message à un noeud suivant jusqu'au récepteur final. Deux possibilités s'offrent alors au concepteur de systèmes : transmettre le message tel quel, ou bien le modifier (par exemple pour y ajouter l'identifiant du noeud actuel par lequel le message transite), le sécuriser et le renvoyer. Ces traitements supplémentaires consomment à nouveau du temps processeur et de l'énergie à prendre en compte. En cas d'alimentation continue stable, l'énergie n'est pas un problème pour ce noeud, mais le temps d'exécution de ce traitement peut impacter le fonctionnement global du réseau.

Adapter ce modèle avec cette amélioration reviendrait à ajouter des éléments (places et transitions) d'un émetteur au noeud-récepteur, le noeud étant une nouvelle entité du réseau de Petri par rapport à l'existant présenté précédemment.

3. La loi exponentielle est sans mémoire, c'est-à-dire que la probabilité d'occurrence d'un évènement est indépendante du précédent évènement. Cette loi est à choisir judicieusement selon la grandeur mesurée. Pour une température contrôlée, la loi exponentielle n'est pas forcément appropriée. La loi choisie est donc un paramètre au même titre que λ.

Sans être un noeud, le récepteur peut être connecté à un serveur distant et transférer les données à ce dernier. La durée de traitement côté récepteur est donc choisi arbitrairement (elle est fixée à 1 ms) mais peut être modifiée pour correspondre au traitement effectivement réalisé au sein d'un WSN.

Le modèle étant expliqué, il est désormais possible d'utiliser les outils de simulation proposés par GRIF 2011 pour observer le comportement d'un WSN.

5.2 Exploitation du modèle

5.2.1 Simulations et Monte Carlo

Simuler un évènement

Le modèle présenté dans la partie précédente dépend d'aléas au niveau de chaque émetteur : l'apparition d'un évènement, c'est-à-dire d'un changement de la grandeur mesurée par le capteur.

Dans le cadre du modèle, il est intéressant de raisonner en terme de durée entre deux changements de valeur sur la grandeur mesurée pour considérer un évènement. C'est en effet ce changement de valeur de la grandeur mesurée qui provoque une interruption pour réveiller l'émetteur et envoyer un message. L'évènement est dit aléatoire car la grandeur peut varier aléatoirement en suivant certaines lois selon l'environnement et la grandeur elle-même.

Cette notion d'évènement aléatoire a été détaillée dans la partie 4.2.2 et il a été vu dans cette partie qu'une grandeur peut changer un certain nombre de fois sur une durée.

Le changement de la grandeur peut dépendre de différentes lois de probabilité selon le contexte. La loi exponentielle a été choisie dans cette étude. Cette loi est utile pour modéliser un évènement sans mémoire : la probabilité qu'un changement ait lieu à un instant t ne dépend pas du précédent changement. Cela peut être faux dans certains cas. Par exemple, si un changement de température a lieu dans un four à température contrôlée, il est fort possible que cela indique un démarrage et que la température va continuer à augmenter dans les prochains instants.

Le choix de la loi est laissé à la discrétion du concepteur. Pour cette étude cependant, seule la loi exponentielle est considérée, avec un paramètre λ_i propre à chaque

émetteur. λ_i peut être vu comme le nombre de changements ayant eu lieu sur un intervalle de temps t pour l'émetteur i.

Attention à la subtilité suivante par rapport à l'utilisation qui est faite de GRIF 2011 dans cette étude : l'unité de temps atomique est l'heure, mais comme expliqué précédemment, l'étude considère que cette unité de temps atomique est d'une ms afin de pouvoir exprimer convenablement les autres unités (consommation, durées des transitions...). Par cohérence donc, une variable changeant en moyenne 2 fois par minute (donc 120 fois par heure) correspond à en moyenne $3,33 * 10^{-5}$ changement par ms, ce qui donne la valeur du λ_i à utiliser pour une telle grandeur sur le modèle de GRIF 2011 défini précédemment.

Durée d'une simulation

Une simulation est caractérisée par sa durée. En l'occurrence, la durée d'une simulation est de 10 000 000 ms, soit environ 2,75h. Cette limite est causée par l'espace mémoire réservée à la variable de durée : un trop grand nombre ne peut pas être correctement interprété par le logiciel GRIF 2011. Cela est principalement dû à l'utilisation particulière du logiciel qui est faite ici, plutôt qu'à un défaut de développement du logiciel.

Toutefois, cette durée reste suffisante pour donner l'état du système lorsque tous les émetteurs sont en fonctionnement. Le seul défaut de cette méthode est l'utilisation de la batterie : il n'est pas possible de l'épuiser avec cette simulation car, comme vu dans l'annexe B, l'autonomie de l'émetteur peut aller jusqu'à plusieurs milliers d'heures. Cela n'empêche cependant pas d'extrapoler la courbe de consommation de la batterie à partir de ces *courtes* simulations.

Simulation d'un processus stochastique

La simulation dépendant d'un évènement aléatoire, lancer deux simulations peut donner des résultats différents selon les réalisations des évènements à chaque lancement. Il est alors courant de répéter les simulations un certain nombre de fois et de moyenner les résultats : c'est la méthode de Monte Carlo, utilisée par Moca. Moca est l'utilitaire de GRIF 2011 qui permet de lancer plusieurs *histoires* (simulations) et de moyenner les variables observées.

Ces variables observées sont :

— Le nombre de messages perdus M_{perdus}.

— Le nombre de messages traités M_{trait}.

— Le niveau de batterie des émetteurs $Batt_i$.

Il reste alors à choisir astucieusement les paramètres afin de refléter des cas concrets.

Remarque sur le nombre d'histoire

Le nombre d'histoires lancées par Moca a été limitée par la puissance de la machine sur laquelle tournait GRIF 2011. Au bout d'une certaine durée, le logiciel arrête la simulation. Le nombre d'histoires utilisées a donc été de 12.

Si ce nombre d'histoires peut paraître faible, les évènements apportant de l'aléatoire dans l'étude sont relativement stables pour les unités de temps concernées, permettant d'afficher les tendances présentées ci-après. Pour s'en convaincre, l'écart-type des valeurs moyennées sur les 12 histoires seront renseignés.

5.2.2 Autonomie de l'émetteur selon un évènement aléatoire

Un évènement aléatoire

La partie 4.2.2 a pu mettre en avant le lien entre la sécurité et l'autonomie de l'émetteur selon un processus déterministe : les messages sont envoyés de manière régulière. Il s'agissait alors de mesurer la consommation d'un cycle, consommations récapitulées en annexe B.

L'émetteur tel que modélisé dans le réseau de Petri de la figure 5.3 permet quant à lui d'exprimer cette notion de consommation énergétique en fonction des mêmes paramètres, mais cette fois en ajoutant un évènement aléatoire : le changement de la grandeur mesurée. Cet évènement étant aléatoire, l'outil Moca est utilisé pour lisser cet aléatoire. Bien qu'aléatoire, un évènement arrive plus ou moins fréquemment selon le λ_i comme expliqué précédemment.

Trois scénarios ont été définis afin de tester cinq valeurs de lambda :

— Pas d'interruption ($\lambda_i = \emptyset$) : seul le signe de vie est envoyé (car la donnée mesurée ne change pas). Il s'agit des courbes rouges.

— $\lambda_i = 3,3.10^{-5}$: cela correspond à 120 changements de valeur par heure en moyenne (2 par minute). Il s'agit des courbes bleues.

— $\lambda_i = 6,6.10^{-5}$: le double de la valeur précédente, c'est-à-dire 240 changements de valeur par heure en moyenne (4 par minute). Il s'agit des courbes violettes.

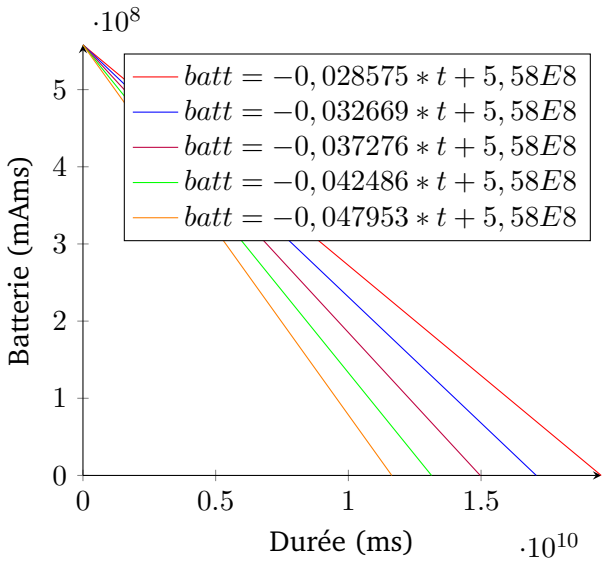

Fig. 5.6.: Variation de l'autonomie de l'émetteur selon λ_i - Scénario 1

— $\lambda_i = 9,9.10^{-5}$: 360 changements de valeur par heure en moyenne, (6 par minute). Il s'agit des courbes vertes.

— $\lambda_i = 13,2.10^{-5}$: 480 changements de valeur par heure en moyenne (8 par minute). Il s'agit des courbes orange.

Les émetteurs possédant les mêmes composants dans le modèle, l'analyse de l'autonomie par rapport à λ_i se fera sur un seul émetteur.

Scénario 1

Le premier jeu de paramètre répète une fois le message ($Nb_{rep} = 2$) et n'implémente pas d'algorithmes de sécurité ($Te_{hash} + Te_{chiff} + Te_{auth} = 0$). Cela donne les courbes représentées sur la figure 5.6 avec les cinq valeurs de λ_i définies précédemment.

Les courbes donnent une droite ayant pour origine le niveau de batterie au temps zéro : $5,58.10^8$. Une approximation linéaire pour chaque λ_i donne les équations suivantes :

— $batt = -0,028575 * t + 5,58.10^8$ lorsqu'il n'y a pas d'interruption. Cela donne un épuisement théorique de la batterie[4] au bout de $1,95.10^{10}$ ms, soient 5420 heures ou encore 226 jours.

— $batt = -0,032669 * t + 5,58.10^8$ lorsque $\lambda_i = 3,3.10^{-5}$. Cela donne un épuisement théorique de la batterie au bout de $1,71.10^{10}$ ms, soient 4750 heures ou encore 198 jours.

4. L'épuisement total de la batterie est considéré, c'est-à-dire que la valeur de la batterie arrive à zéro. En réalité, une marge est prise par le concepteur car une batterie à zéro se recharge moins efficacement : mieux vaut donc s'arrêter avant. Dans les calculs cela revient à diminuer la capacité de la batterie pour prendre en compte ce seuil.

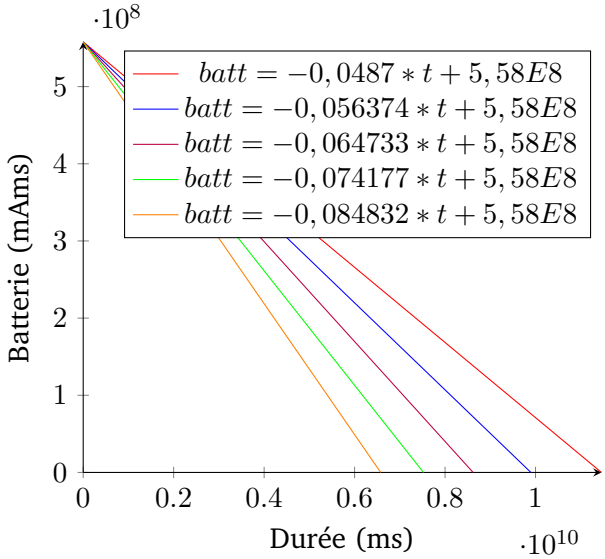

Fig. 5.7.: Variation de l'autonomie de l'émetteur selon λ_i - Scénario 2

— $batt = -0,037276 * t + 5,58.10^8$ lorsque $\lambda_i = 6.6.10^{-5}$. Cela donne un épuisement théorique de la batterie au bout de $1,50.10^{10}$ ms, soient 4170 heures ou encore 174 jours.

— $batt = -0,042486 * t + 5,58.10^8$ lorsque $\lambda_i = 9,9.10^{-5}$. Cela donne un épuisement théorique de la batterie au bout de $1,32.10^{10}$ ms, soient 3660 heures ou encore 152 jours.

— $batt = -0,047953 * t + 5,58.10^8$ lorsque $\lambda_i = 13,2.10^{-5}$. Cela donne un épuisement théorique de la batterie au bout de $1,17.10^{10}$ ms, soient 3240 heures ou encore 135 jours.

Scénario 2

Le deuxième jeu de paramètre répète une fois le message ($Nb_{rep} = 2$) et implémente le chiffrement AES ($Te_{chiff} = 0.864$ ms) et l'authentification/intégrité par HMAC-SHA512 ($Te_{hash} = 0$ ms car le hachage est directement intégré dans Te_{auth} et $Te_{auth} = 5.22$ ms). Cela donne les courbes représentées sur la figure 5.7 avec les cinq valeurs de λ_i définies précédemment.

Là aussi, une approximation linéaire pour chaque λ_i donne les équations suivantes :

— $batt = -0,0487 * t + 5,58.10^8$ lorsqu'il n'y a pas d'interruption. Cela donne un épuisement théorique de la batterie au bout de $1,15.10^{10}$ ms, soient 3180 heures ou encore 133 jours.

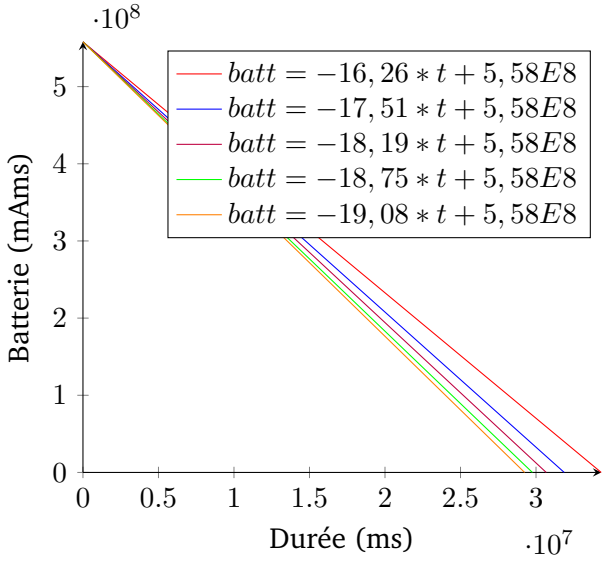

Fig. 5.8.: Variation de l'autonomie de l'émetteur selon λ_i - Scénario 3

— $batt = -0,056374 * t + 5,58.10^8$ lorsque $\lambda_i = 3,3.10^{-5}$. Cela donne un épuisement théorique de la batterie au bout de $9,91.10^9$ ms, soient 2750 heures ou encore 115 jours.

— $batt = -0,064733 * t + 5,58.10^8$ lorsque $\lambda_i = 6,6.10^{-5}$. Cela donne un épuisement théorique de la batterie au bout de $8,63.10^9$ ms, soient 2400 heures ou encore 99,8 jours.

— $batt = -0,074177 * t + 5,58.10^8$ lorsque $\lambda_i = 9,9.10^{-5}$. Cela donne un épuisement théorique de la batterie au bout de $7,54.10^9$ ms, soient 2090 heures ou encore 87,3 jours.

— $batt = -0,084832 * t + 5,58.10^8$ lorsque $\lambda_i = 13,2.10^{-5}$. Cela donne un épuisement théorique de la batterie au bout de $6,62.10^9$ ms, soient 1840 heures ou encore 76,6 jours.

Ce scénario sans interruption correspond à l'annexe B.2 qui met en évidence l'autonomie dans le cas d'un processus déterministe (l'autonomie trouvée était de 132,98 jours). Comme attendu, si davantage de mesures ont lieu sur la durée d'utilisation de l'émetteur, l'autonomie en pâtie.

Scénario 3

Le troisième jeu de paramètres répète une fois le message ($Nb_{rep} = 2$) et implémente le chiffrement AES ($Te_{chiff} = 0.864$ ms), le contrôle d'intégrité SHA512 ($Te_{hash} = 2.116$ ms) et une signature RSA-2048 ($Te_{auth} = 9560$ ms). Cela donne les courbes représentées sur la figure 5.8 avec les cinq valeurs de λ_i définies précédemment.

Une fois encore, une approximation linéaire pour chaque λ_i donne les équations suivantes :

— $batt = -16,26 * t + 5,58.10^8$ lorsqu'il n'y a pas d'interruption. Cela donne un épuisement théorique de la batterie au bout de $3,43.10^7$ ms, soient 9,54 heures.

— $batt = -17,51 * t + 5,58.10^8$ lorsque $\lambda_i = 3,3.10^{-5}$. Cela donne un épuisement théorique de la batterie au bout de $3,19.10^7$ ms, soient 8,85 heures.

— $batt = -18,19 * t + 5,58.10^8$ lorsque $\lambda_i = 6,6.10^{-5}$. Cela donne un épuisement théorique de la batterie au bout de $3,07.10^7$ ms, soient 8,52 heures.

— $batt = -18,75 * t + 5,58.10^8$ lorsque $\lambda_i = 9,9.10^{-5}$. Cela donne un épuisement théorique de la batterie au bout de $2,98.10^7$ ms, soient 8,27 heures.

— $batt = -19,08 * t + 5,58.10^8$ lorsque $\lambda_i = 13,2.10^{-5}$. Cela donne un épuisement théorique de la batterie au bout de $2,92.10^7$ ms, soient 8,12 heures.

Ces scénarios confirment et dimensionnent l'influence de la sécurité et de la fréquence de l'évènement sur l'autonomie de l'émetteur.

Mais si le récepteur n'a pas de problème d'autonomie, sa problématique est toute autre et concerne principalement la disponibilité de ce dernier.

5.2.3 La qualité de service (QoS) d'un WSN

Les trois scénarios précédents sont utilisés une nouvelle fois pour mettre en avant l'évolution de la Qualité de Service (QoS) en fonction :

— Du scénario choisi : plus précisément de la suite de sécurité choisie.

— Du nombre d'émetteurs N.

— De λ_i tel que défini précédemment. À noter que tous les émetteurs dans cette simulation mesureraient une même grandeur dans un même environnement car ils possèdent le même λ_i.

Dans ce contexte, la QoS du WSN est vue comme le nombre de messages traités par rapport au nombre de messages générés. Il s'agit donc d'un ratio défini par l'équation 5.1.

$$QoS = \frac{M_{trait}}{M_{trait} + M_{perdus}} \tag{5.1}$$

Où :

— M_{trait} : le nombre de messages correctement reçus et traités par le récepteur

— M_{perdu} : le nombre de messages perdus car le récepteur était indisponible (occupé avec d'autres messages).

Les durées des différentes tâches d'un récepteurs sont récapitulées en annexe C et correspondent aux paramètres du réseau de Petri de la figure 5.4 pour les transitions. L'électronique du récepteur est considérée comme identique à l'émetteur. Le traitement des algorithmes symétriques étant donc le même, leur durée sera aussi la même.

Scénario 1

Le cas du scénario 1 (sans sécurité) avec un seul émetteur peut valoir de témoin. Le nombre de messages générés et le nombre de messages traités par le récepteur en fonction de λ_i sont récapitulés dans le tableau 5.1. À noter qu'un $\lambda_i = \emptyset$ représente l'absence d'interruption : seul le signal de vie génère une mesure et un message.

Pour rappel, dire que 3564.67 messages ont été générés ne signifie pas que 0.67 message a été envoyé. Ce nombre est la moyenne du nombre de messages générés sur l'ensemble des *histoires* générées par Moca, d'où un résultat à virgule. L'écart-type de ces moyennes est renseignée entre parenthèse (σ).

Il est intéressant de noter que pour $\lambda = \emptyset$, l'écart-type est de 0, signifiant que le nombre de messages générés sur chaque histoire est le même. Cela s'explique par le caractère déterministe d'un tel WSN : il ne dépend pas des interruptions provoquées par les changements de valeur de la grandeur mesurée, mais uniquement des signaux de vie.

De manière générale, le scénario 1 peut aussi servir de témoin. Plus précisément, il donne une idée du nombre de messages générés sur un WSN pour différents λ_i, et il se trouve que le récepteur peut récupérer et traiter l'ensemble de ces messages, d'où une QoS de 100 %.

Scénario 2

Le scénario 2 également décrit précédemment utilise un chiffrement symétrique AES 256, et un HMAC-SHA512 permettant de faire l'authentification en plus du contrôle d'intégrité. Le tableau 5.2 montre que la baisse de la qualité de service est évidente dès 6 émetteurs.

De plus, il est possible de voir que le niveau de sécurité impacte différemment la QoS selon le nombre d'émetteur et la valeur de λ_i. Par exemple, entre le scénario 1

N	λ_i	M_{perdu} (σ)	M_{trait} (σ)	QoS
1	$\emptyset$	0 (0)	1990 (0)	100%
	$3,3.10^{-5}$	0 (0)	2349,33 (20,46)	100%
	$6,6.10^{-5}$	0 (0)	2725,33 (34,54)	100%
	$9,9.10^{-5}$	0 (0)	3138,33 (32,15)	100%
	$13,2.10^{-5}$	0 (0)	3564,67 (42,99)	100%
6	$\emptyset$	0 (0)	11880 (0)	100%
	$3,3.10^{-5}$	0 (0)	14039,7 (40,04)	100%
	$6,6.10^{-5}$	0 (0)	16324,3 (45,07)	100%
	$9,9.10^{-5}$	0 (0)	18806 (134,80)	100%
	$13,2.10^{-5}$	0 (0)	21331,7 (178,75)	100%
12	$\emptyset$	0 (0)	23750 (0)	100%
	$3,3.10^{-5}$	0 (0)	28119,1 (86,55)	100%
	$6,6.10^{-5}$	0 (0)	32717,8 (150,42)	100%
	$9,9.10^{-5}$	0 (0)	37636,3 (171,62)	100%
	$13,2.10^{-5}$	0 (0)	42759 (259,64)	100%
18	$\emptyset$	0 (0)	35612 (0)	100%
	$3,3.10^{-5}$	0 (0)	42167,8 (61,36)	100%
	$6,6.10^{-5}$	0 (0)	49084,3 (132,27)	100%
	$9,9.10^{-5}$	0 (0)	56446,3 (216,11)	100%
	$13,2.10^{-5}$	0 (0)	64046,7 (200,65)	100%
24	$\emptyset$	0 (0)	47468 (0)	100%
	$3,3.10^{-5}$	0 (0)	56258,5 (85,40)	100%
	$6,6.10^{-5}$	0 (0)	65422,3 (151,50)	100%
	$9,9.10^{-5}$	0 (0)	75337,3 (212,83)	100%
	$13,2.10^{-5}$	0 (0)	85427,2 (303,47)	100%

Tab. 5.1.: Évolution de la QoS - Scénario 1

N	λ_i	M_{perdu} (σ)	M_{trait} (σ)	QoS
1	$\emptyset$	0 (0)	1988	100%
	$3,3.10^{-5}$	0 (0)	2343,17	100%
	$6,6.10^{-5}$	0 (0)	2719,83	100%
	$9,9.10^{-5}$	0 (0)	3131,83	100%
	$13,2.10^{-5}$	0 (0)	3579	100%
6	$\emptyset$	31,08 (4,80)	11836,9 (4,80)	99,74%
	$3,3.10^{-5}$	106,67 (15,42)	13947,1 (41,94)	99,24%
	$6,6.10^{-5}$	144,33 (16,54)	16173,7 (81,14)	99,12%
	$9,9.10^{-5}$	180,91 (14,90)	18639,7 (117,14)	99,04%
	$13,2.10^{-5}$	244,42 (15,53)	21062,6 (173,68)	98,85%
12	$\emptyset$	70 (5,38)	23656 (5,38)	99,70%
	$3,3.10^{-5}$	476,83 (36,90)	27645,9 (63,71)	98,30%
	$6,6.10^{-5}$	609,08 (39,85)	32110,3 (119,43)	98,14%
	$9,9.10^{-5}$	802,04 (41,14)	36852,5 (126,44)	97,86%
	$13,2.10^{-5}$	1034,67 (33,08)	41662,1 (204,71)	97,58%
18	$\emptyset$	109,25 (6,94)	35470,7 (6,94)	99,69%
	$3,3.10^{-5}$	1042,33 (63,03)	41146,3 (80,06)	97,53%
	$6,6.10^{-5}$	1401,17 (55,88)	47664,6 (156,80)	97,14%
	$9,9.10^{-5}$	1840,67 (61,71)	54612,9 (216,426)	96,74%
	$13,2.10^{-5}$	2373,42 (48,00)	61651,3 (241,59)	96,29%
24	$\emptyset$	151,5 (14,90)	47284,5 (14,90)	99,68%
	$3,3.10^{-5}$	1869,5 (81,46)	54380,6 (116,00)	96,68%
	$6,6.10^{-5}$	2485,08 (69,64)	62917,7 (196,66)	96,20%
	$9,9.10^{-5}$	3251,17 (65,89)	72058 (208,00)	95,68%
	$13,2.10^{-5}$	4195 (72,22)	81188,4 (238,68)	95,09%

Tab. 5.2.: Évolution de la QoS - Scénario 2

N	λ_i	M_{perdu} (σ)	M_{trait} (σ)	QoS
1	$\emptyset$	510 (0)	510 (0)	50,00%
	$3,3.10^{-5}$	549,25 (3,49)	549,25 (3,49)	50,00%
	$6,6.10^{-5}$	570,83 (4,67)	570,75 (4,69)	50,00%
	$9,9.10^{-5}$	588,25 (3,44)	588,17 (3,46)	50,00%
	$13,2.10^{-5}$	598,58 (2,87)	598,42 (2,94)	49,99%
6	$\emptyset$	4595,58 (0,79)	1504,42 (0,79)	24,66%
	$3,3.10^{-5}$	3739,08 (28,62)	2812,83 (27,96)	42,93%
	$6,6.10^{-5}$	3910,92 (25,22)	2936,17 (23,42)	42,88%
	$9,9.10^{-5}$	4023,92 (30,37)	3023,92 (15,55)	42,91%
	$13,2.10^{-5}$	4092,75 (21,32)	3056,00 (21,42)	42,75%
12	$\emptyset$	9226,33 (1,15)	2971,67 (1,15)	24,36%
	$3,3.10^{-5}$	8213,92 (55,54)	4878,58 (52,99)	37,26%
	$6,6.10^{-5}$	8642,33 (43,59)	5045,17 (31,08)	36,86%
	$9,9.10^{-5}$	8925,33 (23,27)	5146,42 (20,92)	36,57%
	$13,2.10^{-5}$	9094,58 (31,30)	5208,25 (22,82)	36,41%
18	$\emptyset$	14321,5 (1,09)	3970,5 (1,09)	21,71%
	$3,3.10^{-5}$	13180,00 (53,09)	6446,50 (46,60)	32,85%
	$6,6.10^{-5}$	13922,60 (58,66)	6622,25 (49,75)	32,23%
	$9,9.10^{-5}$	14402,10 (59,04)	6731,83 (25,05)	31,85%
	$13,2.10^{-5}$	14621,10 (43,59)	6807,00 (20,40)	31,77%
24	$\emptyset$	18943,80 (1,64)	5444,17 (1,64)	22,32%
	$3,3.10^{-5}$	18501,00 (36,61)	7666,00 (40,63)	29,30%
	$6,6.10^{-5}$	19549,30 (42,60)	7843,83 (35,14)	28,63%
	$9,9.10^{-5}$	20174,80 (36,85)	7987,17 (24,77)	28,36%
	$13,2.10^{-5}$	20530,20 (32,93)	8039,33 (29,63)	28,14%

Tab. 5.3.: Évolution de la QoS - Scénario 3

et le scénario 2 pour 6 émetteurs et $\lambda_i = 9,9.10^{-5}$, la différence de QoS est de 0,96 %, tandis que pour 24 émetteurs et $\lambda_i = 9,9.10^{-5}$, la différence est de 4,32 %.

Scénario 3

Ce scénario reprend les paramètres énoncés dans le scénario 3 déjà défini plus haut dans ce document et les résultats sont présentés dans le tableau 5.3. Il convient cependant d'expliciter la vérification RSA côté récepteur. En effet, le système étant asymétrique, la durée de l'opération diffère entre l'émetteur et le récepteur : la durée T_{vauth} est de 598ms pour une durée de signature Te_{auth} de 9560ms côté émetteur.

La qualité de service très faible sur ce scénario s'explique par l'utilisation de RSA : la durée des algorithmes de sécurisation sur le récepteur occupe ce dernier pour la majorité du temps, l'empêchant ainsi de traiter les autres messages qui arrivent. Ceci est particulièrement vrai pour la répétition du message qui est émise en moyenne 20ms après le premier message. Cela signifie que dans ce cas, au mieux, le récepteur

ne peut traiter qu'un message reçu sur deux, d'où le plafonnement à 50% pour un seul émetteur.

Enfin, les conditions initiales du réseau de Petri sont particulièrement importantes ici car afin de simuler un branchement différé de chaque émetteur, le premier émetteur est activé tout de suite, mais le suivant est activé 300ms plus tard, et ainsi de suite pour les émetteurs suivants. Les messages envoyés par ces émetteurs seront perdus car le récepteur va mettre $T_{vauth} = 9560ms$ pour traiter le premier message. Le premier message émis à plus de 9560ms pourra ensuite être traité.

Cela explique la QoS plus faible pour $\lambda_i = \emptyset$ dans ce scénario que pour les autres λ_i : l'aléatoire fait disparaître cet état déterministe initial et distribue les évènements dans le temps de façon plus aléatoire, réduisant le nombre de messages perdus côté récepteur.

Enfin, il est intéressant de remarquer que le nombre de messages générés est plus faible ici que sur les autres scénarios. C'est parce que l'émetteur est occupé à signer le message et ne peut pas relever l'évènement quand la grandeur a changé.

L'utilisation de RSA pour un tel WSN est donc à éviter.

5.3 Analyse des résultats

5.3.1 Gestion de la sécurité

En reprenant le tableau 2.2 récapitulant la complexité de la meilleure attaque des algorithmes, le concepteur de systèmes embarqués peut estimer le temps que prendrait un attaquant à casser ses communications. À la fin de ce temps, la clé de chiffrement est considérée comme connue de l'attaquant. Cela implique donc de renouveler la clé avant la fin de cette durée.

Changer la clé de chiffrement peut s'avérer impossible dans certains cas (pas de mode de réception possible sur le capteur, puissance de calcul trop faible pour gérer l'échange de clés...). Ce temps de cassage pour de tels appareils peut potentiellement être vue comme la durée de vie de ceux-ci.

Cette durée peut être fixée pour répondre à un besoin de compétitivité commerciale et est donc une caractéristique fixée. Par exemple, AES, par sa robustesse est souvent choisi aujourd'hui, ce qui donne le scénario 2 présenté précédemment.

Différents cas peuvent se poser permettant de ne pas forcément prendre en compte la vulnérabilité du hachage. Par exemple, si le message est haché et que l'ensemble (message et empreinte) sont chiffrés, l'empreinte n'apparaîtra pas et ne sera pas utilisable sans avoir cassé l'algorithme de chiffrement.

5.3.2 Gestion de la QoS

La qualité de service évolue selon le niveau de sécurité choisi mais également selon le nombre d'émetteurs qui envoient des messages au récepteur. Ce nombre de messages dépend de l'environnement de mesure (λ).

Le concepteur peut garantir une qualité de service jusqu'à un certain niveau de λ et d'émetteurs qu'il peut ensuite traduire en conseils d'utilisation de son appareil.

Dans le cas où l'environnement de mesure est clairement identifié (ainsi que le nombre d'émetteurs), si la QoS est trop basse, le concepteur peut alors réaliser des compromis sur la sécurité.

5.3.3 Gestion de l'autonomie

L'autonomie est directement influencée par le nombre de messages échangés (répétition des messages et évènement de mesure déclenché) ainsi que l'algorithme de sécurité choisi comme vu en partie 4.2.2.

Deux approches de l'autonomie sont possibles pour le concepteur :

1. La batterie est déjà choisie par rapport à des contraintes d'encombrement physique. Sa capacité est donc déjà fixée.
2. La batterie est choisie après l'application de cette méthodologie pour correspondre à la "durée de vie" associée au cassage l'algorithme de chiffrement utilisé. Cela est particulièrement utile pour les équipements nécessitant un retour usine pour changer la batterie (pour les équipements ATEX par exemple). Il peut être pertinent d'utiliser ce retour pour également changer la clé de chiffrement en programmant à nouveau l'équipement.

Dans le cas 1, le concepteur fixe l'autonomie souhaitée et choisit l'algorithme lui permettant d'atteindre cette autonomie, en estimant l'environnement où les mesures seront effectuées (λ).

Dans le cas 2, l'algorithme est déjà choisi et le concepteur pose des limites d'utilisations garantissant l'autonomie visée (λ doit être relativement peu élevé pour garder l'autonomie visée).

Résumé de la partie

Cette partie a mis en avant l'influence de la sécurité sur la qualité de service d'un WSN, mais également l'influence du nombre d'émetteurs, et du comportement des grandeurs mesurées à partir de chaque émetteur sur la QoS.
La recherche d'une configuration multi-critère peut alors être simplifiée grâce à ce modèle où le concepteur peut entrer ses critères et contraintes et faire évoluer ces paramètres jusqu'à la configuration idéale.

6 Conclusion

> *C'était vraiment très intéressant !*
>
> **— COGIP**
> Message à caractère informatif, réalisé par
> Nicolas Charlet et Bruno Lavaine

6.1 Synthèse

Dans un contexte où la sécurité absolue n'existe pas en pratique et où la quantité d'informations augmente continuellement, la partie 2 a montré que la sécurité au sein d'un système embarqué nécessite un compromis entre le choix des algorithmes (et leur robustesse) et la performance calculatoire du matériel utilisé.

Il a été vu dans la partie 3 que les problématiques de compromis entre sécurité, disponibilité et autonomie n'étaient pas abordées de manière complète dans le cadre de WSN. C'est notamment le cas en liant la sécurité d'une communication avec la capacité d'un récepteur à traiter l'ensemble des messages générés par des évènements aléatoires, tout en prenant en compte l'autonomie de l'émetteur.

Le protocole expérimental permettant de mesurer des performances temporelles des algorithmes sur un système embarqué ainsi que la définition de plusieurs paramètres et variables dans la partie 4 ont permis d'aborder le problème de l'autonomie dans un contexte déterministe.

C'est finalement la partie 5 qui met en évidence l'influence de la sécurité, de la fréquence des interruptions et du nombre d'émetteurs sur la qualité de service d'un WSN et l'autonomie de ses émetteurs. En s'appuyant sur un modèle de réseaux de Petri et des simulations à l'aide de l'outil Moca, il est possible de quantifier ces influences et donc d'estimer l'autonomie et la QoS d'un WSN.

Le modèle proposé et les simulations ont permis de valider certaines approches empiriques mais permettent surtout de quantifier cette qualité de service se-

lon le contexte du WSN (grandeurs mesurées, niveau de sécurité souhaité...) tout en estimant la durée de vie des émetteurs.

6.2 Travaux futurs

6.2.1 Outils d'optimisations

Les réseaux de Petri ne forment pas ici une finalité mais une étape à une démarche plus globale d'aide à la décision. L'étape suivante serait l'optimisation, qui reposerait sur des algorithmes d'heuristique afin de déterminer les jeux de paramètres offrant le meilleur compromis parmi les critères demandés par le concepteur de systèmes embarqués.

Par exemple, dans une telle démarche, le concepteur de systèmes embarqués définit d'abord les λ (nombre moyen d'évènements selon la grandeur mesurée) par rapport aux fonctionnalités de son système embarqué (réactivité, précision du capteur), un niveau de batterie (liée à la batterie qu'il aura choisi pour son système) et la sécurité à implémenter (chiffrement, authentification et contrôle d'intégrité, ou bien juste authentification et contrôle d'intégrité par exemple ; ou encore utilisation de suites asymétriques ou symétriques). Plus important, il définira l'autonomie et la QoS minimale souhaitée, la sécurité minimale souhaitée (qui dépendra des complexités de meilleures attaques), le nombre d'émetteurs minimum par récepteur, etc.

L'algorithme d'heuristique utilisé prendra ces contraintes en entrées, et testera plusieurs valeurs des différents paramètres évoqués précédemment pour y trouver la meilleure solution. Cela implique pour chaque jeu de paramètre généré de lancer une simulation pour avoir une estimation de la QoS et de l'autonomie.

Une conséquence directe est de réduire au maximum la durée des simulations, en diminuant le nombre de places et de transitions du modèle proposé ici.

Le concepteur pourra ensuite sélectionner, parmi les solutions optimales, celle qui correspond le plus à son besoin. Néanmoins, pour être plus complète, cette démarche (et le modèle en réseaux de Petri) pourrait intégrer davantage de paramètres, décrits ci-après.

6.2.2 Canal de communication perturbé

Les résultats présentés donnent une première approche pour modéliser un WSN. Cependant, ces simulations ne prennent pas en compte les perturbations du canal de communication (volontairement omises du modèle pour se concentrer sur l'influence de la sécurité).

Associé à ce problème est la distance entre l'émetteur et le récepteur et la puissance d'émission des émetteurs, autant de paramètres pouvant influencer la QoS du WSN, et également l'autonomie des émetteurs (pour la puissance d'émission).

6.2.3 Gestion des acquittements

La gestion d'un acquittement n'est pas pris en compte dans ce modèle. Une adaptation reste cependant possible en modifiant le réseau de Petri de l'émetteur avec des transitions et places supplémentaires : une fois le message envoyé et éventuellement répété, au lieu de repasser en veille, l'émetteur passerait en mode *écoute* pour recevoir un acquittement. S'il n'arrive pas après une durée (à déterminer), un renvoi du message (éventuellement répété) est nécessaire. Cela provoquera en revanche une réduction considérable de l'autonomie, en particulier si de nombreux messages sont échangés.

D'autres simulations seraient intéressantes pour évaluer l'influence de ces opérations sur la QoS et l'autonomie.

6.2.4 Plusieurs types de micro-contrôleurs

Cette thèse a démontré l'utilisation du modèle sur un micro-contrôleur en particulier : le PIC32MX795F512L de Microchip. Il serait intéressant de réaliser les mesures de la partie 4 sur d'autres supports matériels. Cela permettrait de comparer différents micro-contrôleurs pour la mise en place de WSN dans des contextes différents.

Afin de mutualiser les efforts pour réaliser ces mesures, une base de données commune pour référencer les performances des algorithmes sur différents micro-contrôleur pourrait être envisagée.

6.2.5 Des grandeurs différentes

Le modèle proposé considère la même grandeur. Or, dans la réalité, les capteurs mesurent des grandeurs différentes. Ces différences peuvent être par rapport à leur nature (une pression, une température) ou par rapport au contexte de mesure (température dans un four ou dans la nature).

Ces différences peuvent s'exprimer en modifiant le λ_i de l'émetteur i. Il convient d'estimer les lois stochastiques de ces grandeurs pour les intégrer au modèle. Un modèle auto-apprenant pourrait par exemple ajuster ces valeurs par rapport aux mesures effectuées concrètement.

6.3 Application industrielle

6.3.1 Utilisation du modèle

Un tel modèle peut être utile pour un concepteur de systèmes embarqués afin de correctement dimensionner son WSN. Certains paramètres comme le niveau de sécurité, le nombre de répétitions d'un message, l'acquittement, peuvent être paramétrables. Souvent, ils sont choisis à la compilation et codés *en dur* dans la mémoire du micro-contrôleur. Il peut être intéressant de rendre configurable ces paramètres par une Interface Homme-Machine ou par envoi d'un message de configuration. Cela permettra au concepteur d'appliquer le modèle à son WSN suite à une simulation pour le tester concrètement.

Après cette phase de test, il peut être intéressant de passer d'un cas à un autre de manière algorithmique. Par exemple, si le WSN envoie beaucoup de message, le récepteur pourrait envoyer un message aux émetteurs pour passer à un niveau de sécurité moindre, ou à diminuer le nombre de répétition, ou toute autre possibilité...

Attention toutefois à ne pas introduire de vulnérabilités avec ce mode de fonctionnement : un attaquant pourrait tenter de baisser le niveau de sécurité des communications en envoyant un message de configuration contrefait. De tels messages de configuration doivent donc être particulièrement sécurisés.

6.3.2 Outil logiciel

Pour une pratique plus opérationnelle du modèle, la création d'un logiciel peut être envisagée. Ce logiciel permettrait au concepteur de systèmes embarqués d'entrer

ses contraintes et paramètres dans le logiciel et d'en déduire la durée de vie des émetteurs et la QoS du WSN.

Le logiciel serait composé de quatre modules :

1. Modélisation
2. Paramétrage
3. Simulation
4. Optimisation

Un mode SaaS (Software as a Service) est envisagé pour permettre à l'utilisateur d'accéder au logiciel grâce à un client léger (navigateur web). Cette interface donnera la possibilité à l'utilisateur de modéliser le réseau de Petri comme le permet GRIF2011 et de le sauvegarder sur un espace en ligne pour y accéder depuis n'importe quel poste. Il s'agit du module de modélisation.

Une fois le modèle créé, l'utilisateur accédera au module de paramétrage pour associer les paramètres de son WSN à son modèle. L'ensemble des paramètres pourra posséder des valeurs limites (minimum et maximum).

Enfin, une fois le modèle paramétré, l'utilisateur pourra lancer un certain nombre de simulations. Cette phase pouvant nécessiter du temps, elle sera déléguée au serveur, plus puissant que le client léger. L'utilisateur peut alors quitter l'application : le serveur notifiera l'utilisateur une fois les résultats de la simulation disponibles.

Une fois la simulation effectuée, l'utilisateur peut jouer sur les paramètres (entre les minimums et maximums) pour essayer d'optimiser son WSN (sur la QoS ou l'autonomie des émetteurs).

Si nécessaire, le module d'optimisation peut être utilisé pour chercher la meilleure configuration possible selon les critères de l'utilisateur, automatisant le lancement d'autres simulations pour chaque jeu de paramètres générés.

Enfin, un développement supplémentaire pour un module applicatif à installer sur un poste de travail permettrait de communiquer avec le WSN pour y envoyer les paramètres voulus. Ce module applicatif est plus délicat à développer car il devra communiquer avec les WSN qui peuvent être composées de technologies très différentes.

Côté industriel, un tel logiciel pourrait permettre de mieux dimensionner les besoins (nombres de récepteurs, durée entre deux signes de vie, seuil de déclenchement d'une interruption...) afin de les communiquer à un bureau d'étude ou à une équipe en charge de la conception et du développement de la solution. Pour les plus

petites structures, ce logiciel peut directement être utilisé par l'équipe en charge du développement.

Bibliographie

[1]Nicola BARBER. *Who Broke the Wartime Codes ?* en. Google-Books-ID : jDRWAgAAQBAJ. Raintree, 2014 (cité à la page 19).

[2]Gustave BERTRAND. *Enigma ou la plus grande énigme de la guerre 1939-1945*. French. PLON, 1973 (cité à la page 13).

[3]Daniel BRISSAUD, Yannick FREIN et Valérie ROCCHI. „What Tracks for Sustainable Production Systems in Europe ?“ en. In : *Procedia CIRP*. Forty Sixth CIRP Conference on Manufacturing Systems 2013 7 (jan. 2013), p. 9-16 (cité à la page 53).

[4]R. CHANDRAMOULI, S. BAPATLA, K. P. SUBBALAKSHMI et R. N. UMA. „Battery Power-aware Encryption“. In : *ACM Trans. Inf. Syst. Secur.* 9.2 (mai 2006), p. 162-180 (cité aux pages 24, 68).

[5]G. CHIOLA, S. DONATELLI et G. FRANCESCHINIS. „GSPNs versus SPNs : what is the actual role of immediate transitions ?“ In : *Proceedings of the Fourth International Workshop on Petri Nets and Performance Models PNPM91*. Déc. 1991, p. 20-31 (cité à la page 98).

[6]Florian Laurentiu COMAN, Krzysztof Mateusz MALARSKI, Martin Nordal PETERSEN et Sarah RUEPP. „Security Issues in Internet of Things : Vulnerability Analysis of LoRaWAN, Sigfox and NB-IoT“. In : *2019 Global IoT Summit (GIoTS)*. Juin 2019, p. 1-6 (cité à la page 63).

[7]CONFÉRENCE TÉLÉGRAPHIQUE INTERNATIONALE DE LONDRES. *Règlement de service international télégraphique*. 1903 (cité à la page 11).

[8]Dina DEIF et Yasser GADALLAH. „Reliable wireless sensor networks topology control for critical internet of things applications“. en. In : *2018 IEEE Wireless Communications and Networking Conference (WCNC)*. Barcelona : IEEE, avr. 2018, p. 1-6 (cité à la page 67).

[9]Ethmane EL MOUSTAINE et Maryline LAURENT. „Systèmes et techniques RFID : risques et solutions de sécurité“. In : *Techniques de l'ingénieur. Sécurité des systèmes d'information* H5325 (nov. 2012) (cité à la page 57).

[10]ENOCEAN ALLIANCE. *Technical Specifications*. en-US. www.enocean-alliance.org. 2020 (cité à la page 61).

[11]G. FLORIN, C. FRAIZE et S. NATKIN. „Stochastic Petri nets : Properties, applications and tools“. en. In : *Microelectronics Reliability* 31.4 (jan. 1991), p. 669-697 (cité à la page 98).

[12]C. GHEZZI, D. MANDRIOLI, S. MORASCA et M. PEZZE. „A general way to put time in Petri nets“. In : *Proceedings of the 5th international workshop on Software specification and design*. IWSSD '89. Pittsburgh, Pennsylvania, USA : Association for Computing Machinery, avr. 1989, p. 60-67 (cité à la page 97).

[13]Vipul GUPTA, Sumit GUPTA, Sheueling CHANG et Douglas STEBILA. „Performance analysis of elliptic curve cryptography for SSL“. In : *Proceedings of the 1st ACM workshop on Wireless security*. WiSE '02. Atlanta, GA, USA : Association for Computing Machinery, sept. 2002, p. 87-94 (cité aux pages 42, 44).

[14]M. HAMDI et H. ABIE. „Game-based adaptive security in the Internet of Things for eHealth“. In : *2014 IEEE International Conference on Communications (ICC)*. Juin 2014, p. 920-925 (cité à la page 58).

[15]M. IKRAM, M. A. H. CHOWDHURY, H. REDWAN et al. „A Lightweight Mutual Authentication Scheme for Mobile Radio Frequency IDentification (mRFID) Systems“. In : *2008 IEEE International Performance, Computing and Communications Conference*. Déc. 2008, p. 289-296 (cité à la page 62).

[16]IPCC. *Global Warming of 1.5 °C* —. Library Catalog : www.ipcc.ch. 2018 (cité à la page 53).

[17]Wei JIANG, Zhenlin GUO, Yue MA et Nan SANG. „Measurement-based research on cryptographic algorithms for embedded real-time systems“. In : *Journal of Systems Architecture* 59.10, Part D (nov. 2013), p. 1394-1404 (cité aux pages 24, 68).

[18]Wei JIANG, Paul POP et Ke JIANG. „Design optimization for security- and safety-critical distributed real-time applications“. In : *Microprocessors and Microsystems* 52.Supplement C (juil. 2017), p. 401-415 (cité à la page 69).

[19]J. JONSSON et B. KALISKI. *Public-Key Cryptography Standards (PKCS) #1 : RSA Cryptography Specifications Version 2.1*. en. Fév. 2003 (cité à la page 39).

[20]Pascal JUNOD. „On the Complexity of Matsui's Attack“. en. In : *Selected Areas in Cryptography*. Sous la dir. de Serge VAUDENAY et Amr M. YOUSSEF. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2001, p. 199-211 (cité à la page 36).

[21]David KAHN. *The Code-Breakers : The Story of Secret Writing*. English. 1st edition. The Macmillan Company, 1967 (cité à la page 12).

[22]Phil KARN, William Allen SIMPSON et Perry METZGER. *The ESP Triple DES Transform*. en. Sept. 1995 (cité à la page 36).

[23]Stefan KATZENBEISSER, Ünal KOCABAŞ, Vladimir ROŽIĆ et al. „PUFs : Myth, Fact or Busted ? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon“. en. In : *Cryptographic Hardware and Embedded Systems – CHES 2012*. Springer, Berlin, Heidelberg, sept. 2012, p. 283-301 (cité à la page 65).

[24]KAY SOON LOW, W.N.N. WIN et MENG JOO ER. „Wireless Sensor Networks for Industrial Environments“. In : *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*. T. 2. Nov. 2005, p. 271-276 (cité aux pages 51, 53).

[25]Auguste KERCKHOFFS. *La cryptographie militaire*. Volapük. Jan. 1883 (cité à la page 8).

[26]Thorsten KLEINJUNG, Kazumaro AOKI, Jens FRANKE et al. „Factorization of a 768-Bit RSA Modulus“. en. In : *Advances in Cryptology – CRYPTO 2010*. Sous la dir. de Tal RABIN. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2010, p. 333-350 (cité à la page 40).

[27]Donald KNUTH. *Algorithmes*. Français. Stanford, Calif : University of Chicago Press, 2011 (cité à la page 22).

[28]Hugo KRAWCZYK, Ran CANETTI et Mihir BELLARE. *HMAC : Keyed-Hashing for Message Authentication*. en. Library Catalog : tools.ietf.org. Fév. 1997 (cité à la page 42).

[29]Yunbo LI, Anne-Cécile ORGERIE, Ivan RODERO et al. „End-to-end energy models for Edge Cloud-based IoT platforms : Application to data stream analysis in IoT“. In : *Future Generation Computer Systems* (2017) (cité à la page 67).

[30]Jie LIANG et Xue-Jia LAI. „Improved collision attack on hash function MD5“. In : *Journal of Computer Science and Technology* 22.1 (jan. 2007), p. 79-87 (cité à la page 30).

[31]Mitsuru MATSUI. „Linear Cryptanalysis Method for DES Cipher“. en. In : *Advances in Cryptology — EUROCRYPT '93*. Sous la dir. de Tor HELLESETH. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 1994, p. 386-397 (cité à la page 36).

[32]Nick MERRILL. „Better Not to Know ? The SHA1 Collision & the Limits of Polemic Computation“. In : *Proceedings of the 2017 Workshop on Computing Within Limits*. LIMITS '17. Santa Barbara, California, USA : Association for Computing Machinery, juin 2017, p. 37-42 (cité à la page 33).

[33]Hu MING et Wang YAN. „MD5-Based Error Detection“. In : *2009 Pacific-Asia Conference on Circuits, Communications and Systems*. ISSN : null. Mai 2009, p. 187-190 (cité à la page 31).

[34]Richard A. MOLLIN. *RSA and Public-Key Cryptography*. en. Google-Books-ID : owrOBQAAQBAJ. CRC Press, nov. 2002 (cité à la page 11).

[35]R. NARDONE, R. J. RODRÍGUEZ et S. MARRONE. „Formal security assessment of Modbus protocol“. In : *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*. Déc. 2016, p. 142-147 (cité à la page 54).

[36]Phong NGUYEN. „A Montgomery-like square root for the Number Field Sieve“. en. In : *Algorithmic Number Theory*. Sous la dir. de Joe P. BUHLER. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 1998, p. 151-168 (cité à la page 40).

[37]Karsten NOHL, David EVANS, Starbug STARBUG et Henryk PLÖTZ. „Reverse-engineering a Cryptographic RFID Tag“. In : *Proceedings of the 17th Conference on Security Symposium*. SS'08. Berkeley, CA, USA : USENIX Association, 2008, p. 185-193 (cité à la page 64).

[38]A. B. PAWAR et S. GHUMBRE. „A survey on IoT applications, security challenges and counter measures“. In : *2016 International Conference on Computing, Analytics and Security Trends (CAST)*. Déc. 2016, p. 294-299 (cité à la page 58).

[39]Florent PÉPIN et Maria Grazia VIGLIOTTI. „Risk Assessment of the 3Des in ERTMS“. en. In : *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification*. Sous la dir. de Thierry LECOMTE, Ralf PINGER et Alexander ROMANOVSKY. Lecture Notes in Computer Science. Cham : Springer International Publishing, 2016, p. 79-92 (cité à la page 37).

[40]Carl Adam PETRI. „Kommunikation mit Automaten“. ger. In : *http ://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/pdf/diss_petri.pdf* (1962) (cité à la page 68).

[41]J. PLOENNIGS, U. RYSSEL et K. KABITZSCH. „Performance analysis of the EnOcean wireless sensor network protocol“. In : *2010 IEEE 15th Conference on Emerging Technologies Factory Automation (ETFA 2010)*. Sept. 2010, p. 1-9 (cité à la page 56).

[42]Nachiketh R. POTLAPALLY, Srivaths RAVI, Anand RAGHUNATHAN et Niraj K. JHA. „Analyzing the energy consumption of security protocols“. In : *Proceedings of the 2003 international symposium on Low power electronics and design*. ISLPED '03. Seoul, Korea : Association for Computing Machinery, août 2003, p. 30-35 (cité à la page 68).

[43]Arbia RIAHI SFAR, Enrico NATALIZIO, Yacine CHALLAL et Zied CHTOUROU. „A roadmap for security challenges in the Internet of Things“. In : *Digital Communications and Networks* (avr. 2017) (cité à la page 58).

[44]R. RIVEST. „The MD5 Message-Digest Algorithm“. In : (1992) (cité à la page 29).

[45]Anna RYMASZEWSKA, Petri HELO et Angappa GUNASEKARAN. „IoT powered servitization of manufacturing – an exploratory case study“. In : *International Journal of Production Economics*. Service Implementation in Manufacturing Firms : Strategy, Economics and Practice 192.Supplement C (oct. 2017), p. 92-105 (cité à la page 54).

[46]Alexandre SAVÉRIEN et Jean-Charles FRANÇOIS. *Histoire Des Philosophes Modernes, Volume 5*. fr. Google-Books-ID : zacPAAAAQAAJ. chez Bleuet, 1773 (cité à la page 8).

[47]GT 18-4 CIAME SEE, Mireille BAYART, Blaise CONRARD, André CHOVIN et Michel ROBERT. „Capteurs et actionneurs intelligents“. fr. In : *Ref : TIP660WEB - "Automatique et ingénierie système"* (mar. 2005). Publisher : Editions T.I. | Techniques de l'Ingénieur (cité aux pages 48, 49).

[48]C. E. SHANNON. „Communication theory of secrecy systems“. In : *The Bell System Technical Journal* 28.4 (oct. 1949), p. 656-715 (cité à la page 9).

[49]SIMON SINGH et CATHERINE COQUERET. *Histoire des codes secrets*. Français. Paris : Le Livre de Poche, sept. 2001 (cité aux pages 8, 14).

[50]Marc STEVENS. „New Collision Attacks on SHA-1 Based on Optimal Joint Local-Collision Analysis“. en. In : *Advances in Cryptology – EUROCRYPT 2013*. Sous la dir. de Thomas JOHANSSON et Phong Q. NGUYEN. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2013, p. 245-261 (cité à la page 33).

[51]M. TAN et K. A. MASAGCA. „An Investigation of Bluetooth Security Threats“. In : *2011 International Conference on Information Science and Applications*. Avr. 2011, p. 1-7 (cité à la page 62).

[52]Biaoshuai TAO et Hongjun WU. „Improving the Biclique Cryptanalysis of AES“. en. In : *Information Security and Privacy*. Sous la dir. d'Ernest FOO et Douglas STEBILA. Lecture Notes in Computer Science. Cham : Springer International Publishing, 2015, p. 39-56 (cité à la page 38).

[53]T. TENKANEN et T. HÄMÄLÄINEN. „Security Assessment of a Distributed, Modbus-Based Building Automation System“. In : *2017 IEEE International Conference on Computer and Information Technology (CIT)*. Août 2017, p. 332-337 (cité à la page 54).

[54]Thierry VAL, Eric CAMPO et Adrien VAN DEN BOSSCHE. „Technologie ZigBee / 802.15.4 - Protocoles, topologies et domaines d'application". fr. In : *Ref : TIP382WEB - "Réseaux Télécommunications"* (mai 2008). Publisher : Editions T.I. | Techniques de l'Ingénieur (cité à la page 54).

[55]E. VATTAPPARAMBAN, İ GÜVENÇ, A. İ YUREKLI, K. AKKAYA et S. ULUAĞAÇ. „Drones for smart cities : Issues in cybersecurity, privacy, and public safety". In : *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*. Sept. 2016, p. 216-221 (cité à la page 55).

[56]N. VIDGREN, K. HAATAJA, J. L. PATIÑO-ANDRES, J. J. RAMÍREZ-SANCHIS et P. TOIVANEN. „Security Threats in ZigBee-Enabled Systems : Vulnerability Evaluation, Practical Experiments, Countermeasures, and Lessons Learned". In : *2013 46th Hawaii International Conference on System Sciences*. Jan. 2013, p. 5132-5138 (cité à la page 60).

[57]Xiaoyun WANG, Yiqun Lisa YIN et Hongbo YU. „Finding Collisions in the Full SHA-1". en. In : *Advances in Cryptology – CRYPTO 2005*. Sous la dir. de Victor SHOUP. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2005, p. 17-36 (cité à la page 33).

[58]M. WEYRICH, J. P. SCHMIDT et C. EBERT. „Machine-to-Machine Communication". In : *IEEE Software* 31.4 (juil. 2014), p. 19-23 (cité à la page 53).

[59]Tsz Hon YUEN, Cong ZHANG, Sherman S.M. CHOW et Siu Ming YIU. „Related Randomness Attacks for Public Key Cryptosystems". In : *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*. ASIA CCS '15. Singapore, Republic of Singapore : Association for Computing Machinery, avr. 2015, p. 215-223 (cité à la page 9).

[60]H. ZHANG et Y. AI. „Time Analysis of Scheduling Sequences Based on Petri Nets for Distributed Real-Time Embedded Systems". In : *2006 2nd IEEE/ASME International Conference on Mechatronics and Embedded Systems and Applications*. Août 2006, p. 1-5 (cité à la page 69).

[61]Hai-Tao ZHANG et Gui-Fang WU. „Modeling and analysis of scheduling for distributed real-time embedded systems". en. In : *International Journal of Automation and Computing* 7.4 (nov. 2010), p. 525-530 (cité à la page 69).

[62]Xiu ZHANG, Xiaohui LU et Xin ZHANG. „Mobile wireless sensor network lifetime maximization by using evolutionary computing methods". en. In : *Ad Hoc Networks* 101 (avr. 2020), p. 102094 (cité à la page 67).

[63]Xiaoling ZHENG et Jidong JIN. „Research for the application and safety of MD5 algorithm in password authentication". In : *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery*. ISSN : null. Mai 2012, p. 2216-2219 (cité à la page 25).

[64]E. ZOUGANELI et I. E. SVINNSET. „Connected objects and the Internet of things - A paradigm shift". In : *International Conference on Photonics in Switching*. Sept. 2009, p. 1-4 (cité à la page 51).

A

Détails des mesures des algorithmes de sécurité

A.1 Configuration des algorithmes utilisés sur MPLabX - Harmony 2.0

A.1.1 Configuration DES et 3DES

Cet annexe indique les clés utilisées pour les algorithmes présents dans MPLabX - Harmony 2.0. Ces valeurs correspondent aux valeurs par défaut de la librairie Harmony 2.0. Le lecteur est libre de réutiliser ces valeurs pour d'autres tests ou approfondir les mesures effectuées.

DES

Clé utilisée :

```
[0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef] // 8 octets
```

Vecteur initial utilisé :

```
[0x12,0x34,0x56,0x78,0x90,0xab,0xcd,0xef] // 8 octets
```

3DES

Clé utilisée :

```
[0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef,
 0xfe,0xde,0xba,0x98,0x76,0x54,0x32,0x10,
 0x89,0xab,0xcd,0xef,0x01,0x23,0x45,0x67] // 24 octets
```

Vecteur initial utilisé :

```
[0x12,0x34,0x56,0x78,0x90,0xab,0xcd,0xef,
 0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
 0x11,0x21,0x31,0x41,0x51,0x61,0x71,0x81] // 24 octets
```

A.1.2 Configuration AES

AES-128 (CBC ou CTR)

Clé utilisée :

```
[0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,
 0x38,0x39,0x61,0x62,0x63,0x64,0x65,0x66] // 16 octets
```

Vecteur initial utilisé :

```
[0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,
 0x38,0x39,0x61,0x62,0x63,0x64,0x65,0x66] // 16 octets
```

AES-192 (CBC ou CTR)

Clé utilisée :

```
[0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,
 0x38,0x39,0x61,0x62,0x63,0x64,0x65,0x66,
 0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37] // 24 octets
```

Vecteur initial utilisé :

```
[0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,
 0x38,0x39,0x61,0x62,0x63,0x64,0x65,0x66,
 0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37] // 24 octets
```

AES-256 (CBC ou CTR)

Clé utilisée :

```
[0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,
 0x38,0x39,0x61,0x62,0x63,0x64,0x65,0x66,
 0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,
 0x38,0x39,0x61,0x62,0x63,0x64,0x65,0x66] // 32 octets
```

Vecteur initial utilisé :

```
[0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,
 0x38,0x39,0x61,0x62,0x63,0x64,0x65,0x66,
 0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,
 0x38,0x39,0x61,0x62,0x63,0x64,0x65,0x66] // 32 octets
```

A.1.3 Configuration HMAC

HMAC-MD5

Clés utilisée :

```
[0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,
 0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b] // 16 octets
```

HMAC-SHA1, HMAC-SHA256, HMAC-SHA384, HMAC-SHA512

Clés utilisée :

```
[0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,
 0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,
 0x0b,0x0b,0x0b,0x0b] // 20 octets
```

A.1.4 Configuration RSA

RSA-1024

Tableau d'octet hexadécimal de la clé utilisée :

```
[0x30, 0x82, 0x02, 0x5C, 0x02, 0x01, 0x00, 0x02, 0x81, 0x81,
 0x00, 0xBC, 0x73, 0x0E, 0xA8, 0x49, 0xF3, 0x74, 0xA2, 0xA9,
```

```
0xEF, 0x18, 0xA5, 0xDA, 0x55, 0x99, 0x21, 0xF9, 0xC8, 0xEC,
0xB3, 0x6D, 0x48, 0xE5, 0x35, 0x35, 0x75, 0x77, 0x37, 0xEC,
0xD1, 0x61, 0x90, 0x5F, 0x3E, 0xD9, 0xE4, 0xD5, 0xDF, 0x94,
0xCA, 0xC1, 0xA9, 0xD7, 0x19, 0xDA, 0x86, 0xC9, 0xE8, 0x4D,
0xC4, 0x61, 0x36, 0x82, 0xFE, 0xAB, 0xAD, 0x7E, 0x77, 0x25,
0xBB, 0x8D, 0x11, 0xA5, 0xBC, 0x62, 0x3A, 0xA8, 0x38, 0xCC,
0x39, 0xA2, 0x04, 0x66, 0xB4, 0xF7, 0xF7, 0xF3, 0xAA, 0xDA,
0x4D, 0x02, 0x0E, 0xBB, 0x5E, 0x8D, 0x69, 0x48, 0xDC, 0x77,
0xC9, 0x28, 0x0E, 0x22, 0xE9, 0x6B, 0xA4, 0x26, 0xBA, 0x4C,
0xE8, 0xC1, 0xFD, 0x4A, 0x6F, 0x2B, 0x1F, 0xEF, 0x8A, 0xAE,
0xF6, 0x90, 0x62, 0xE5, 0x64, 0x1E, 0xEB, 0x2B, 0x3C, 0x67,
0xC8, 0xDC, 0x27, 0x00, 0xF6, 0x91, 0x68, 0x65, 0xA9, 0x02,
0x03, 0x01, 0x00, 0x01, 0x02, 0x81, 0x80, 0x13, 0x97, 0xEA,
0xE8, 0x38, 0x78, 0x25, 0xA2, 0x5C, 0x04, 0xCE, 0x0D, 0x40,
0x7C, 0x31, 0xE5, 0xC4, 0x70, 0xCD, 0x9B, 0x82, 0x3B, 0x58,
0x09, 0x86, 0x3B, 0x66, 0x5F, 0xDC, 0x31, 0x90, 0xF1, 0x4F,
0xD5, 0xDB, 0x15, 0xDD, 0xDE, 0xD7, 0x3B, 0x95, 0x93, 0x31,
0x18, 0x31, 0x0E, 0x5E, 0xA3, 0xD6, 0xA2, 0x1A, 0x71, 0x6E,
0x81, 0x48, 0x1C, 0x4B, 0xCF, 0xDB, 0x8E, 0x7A, 0x86, 0x61,
0x32, 0xDC, 0xFB, 0x55, 0xC1, 0x16, 0x6D, 0x27, 0x92, 0x24,
0x45, 0x8B, 0xF1, 0xB8, 0x48, 0xB1, 0x4B, 0x1D, 0xAC, 0xDE,
0xDA, 0xDD, 0x8E, 0x2F, 0xC2, 0x91, 0xFB, 0xA5, 0xA9, 0x6E,
0xF8, 0x3A, 0x6A, 0xF1, 0xFD, 0x50, 0x18, 0xEF, 0x9F, 0xE7,
0xC3, 0xCA, 0x78, 0xEA, 0x56, 0xD3, 0xD3, 0x72, 0x5B, 0x96,
0xDD, 0x4E, 0x06, 0x4E, 0x3A, 0xC3, 0xD9, 0xBE, 0x72, 0xB6,
0x65, 0x07, 0x07, 0x4C, 0x01, 0x02, 0x41, 0x00, 0xFA, 0x47,
0xD4, 0x7A, 0x7C, 0x92, 0x3C, 0x55, 0xEF, 0x81, 0xF0, 0x41,
0x30, 0x2D, 0xA3, 0xCF, 0x8F, 0x1C, 0xE6, 0x87, 0x27, 0x05,
0x70, 0x0D, 0xDF, 0x98, 0x35, 0xD6, 0xF1, 0x8B, 0x38, 0x2F,
0x24, 0xB5, 0xD0, 0x84, 0xB6, 0x79, 0x4F, 0x71, 0x29, 0x94,
0x5A, 0xF0, 0x64, 0x6A, 0xAC, 0xE7, 0x72, 0xC6, 0xED, 0x4D,
0x59, 0x98, 0x3E, 0x67, 0x3A, 0xF3, 0x74, 0x2C, 0xF9, 0x61,
0x17, 0x69, 0x02, 0x41, 0x00, 0xC0, 0xC1, 0x82, 0x0D, 0x0C,
0xEB, 0xC6, 0x2F, 0xDC, 0x92, 0xF9, 0x9D, 0x82, 0x1A, 0x31,
0xE9, 0xE9, 0xF7, 0x4B, 0xF2, 0x82, 0x87, 0x1C, 0xEE, 0x16,
0x6A, 0xD1, 0x1D, 0x18, 0x82, 0x70, 0xF3, 0xC0, 0xB6, 0x2F,
0xF6, 0xF3, 0xF7, 0x1D, 0xF1, 0x86, 0x23, 0xC8, 0x4E, 0xEB,
0x8F, 0x56, 0x8E, 0x8F, 0xF5, 0xBF, 0xF1, 0xF7, 0x2B, 0xB5,
0xCC, 0x3D, 0xC6, 0x57, 0x39, 0x0C, 0x1B, 0x54, 0x41, 0x02,
0x41, 0x00, 0x9D, 0x7E, 0x05, 0xDE, 0xED, 0xF4, 0xB7, 0xB2,
0xFB, 0xFC, 0x30, 0x4B, 0x55, 0x1D, 0xE3, 0x2F, 0x01, 0x47,
0x96, 0x69, 0x05, 0xCD, 0x0E, 0x2E, 0x2C, 0xBD, 0x83, 0x63,
```

```
0xB6, 0xAB, 0x7C, 0xB7, 0x6D, 0xCA, 0x5B, 0x64, 0xA7, 0xCE,
0xBE, 0x86, 0xDF, 0x3B, 0x53, 0xDE, 0x61, 0xD2, 0x1E, 0xEB,
0xA5, 0xF6, 0x37, 0xED, 0xAC, 0xAB, 0x78, 0xD9, 0x4C, 0xE7,
0x55, 0xFB, 0xD7, 0x11, 0x99, 0xC1, 0x02, 0x40, 0x18, 0x98,
0x18, 0x29, 0xE6, 0x1E, 0x27, 0x39, 0x70, 0x21, 0x68, 0xAC,
0x0A, 0x2F, 0xA1, 0x72, 0xC1, 0x21, 0x86, 0x95, 0x38, 0xC6,
0x58, 0x90, 0xA0, 0x57, 0x9C, 0xBA, 0xE3, 0xA7, 0xB1, 0x15,
0xC8, 0xDE, 0xF6, 0x1B, 0xC2, 0x61, 0x23, 0x76, 0xEF, 0xB0,
0x9D, 0x1C, 0x44, 0xBE, 0x13, 0x43, 0x39, 0x67, 0x17, 0xC8,
0x9D, 0xCA, 0xFB, 0xF5, 0x45, 0x64, 0x8B, 0x38, 0x82, 0x2C,
0xF2, 0x81, 0x02, 0x40, 0x39, 0x89, 0xE5, 0x9C, 0x19, 0x55,
0x30, 0xBA, 0xB7, 0x48, 0x8C, 0x48, 0x14, 0x0E, 0xF4, 0x9F,
0x7E, 0x77, 0x97, 0x43, 0xE1, 0xB4, 0x19, 0x35, 0x31, 0x23,
0x75, 0x9C, 0x3B, 0x44, 0xAD, 0x69, 0x12, 0x56, 0xEE, 0x00,
0x61, 0x64, 0x16, 0x66, 0xD3, 0x7C, 0x74, 0x2B, 0x15, 0xB4,
0xA2, 0xFE, 0xBF, 0x08, 0x6B, 0x1A, 0x5D, 0x3F, 0x90, 0x12,
0xB1, 0x05, 0x86, 0x31, 0x29, 0xDB, 0xD9, 0xE2]
```

RSA-2048

Tableau d'octet hexadécimal de la clé utilisée :

```
[0x30, 0x82, 0x04, 0xA4, 0x02, 0x01, 0x00, 0x02, 0x82, 0x01,
 0x01, 0x00, 0xC3, 0x03, 0xD1, 0x2B, 0xFE, 0x39, 0xA4, 0x32,
 0x45, 0x3B, 0x53, 0xC8, 0x84, 0x2B, 0x2A, 0x7C, 0x74, 0x9A,
 0xBD, 0xAA, 0x2A, 0x52, 0x07, 0x47, 0xD6, 0xA6, 0x36, 0xB2,
 0x07, 0x32, 0x8E, 0xD0, 0xBA, 0x69, 0x7B, 0xC6, 0xC3, 0x44,
 0x9E, 0xD4, 0x81, 0x48, 0xFD, 0x2D, 0x68, 0xA2, 0x8B, 0x67,
 0xBB, 0xA1, 0x75, 0xC8, 0x36, 0x2C, 0x4A, 0xD2, 0x1B, 0xF7,
 0x8B, 0xBA, 0xCF, 0x0D, 0xF9, 0xEF, 0xEC, 0xF1, 0x81, 0x1E,
 0x7B, 0x9B, 0x03, 0x47, 0x9A, 0xBF, 0x65, 0xCC, 0x7F, 0x65,
 0x24, 0x69, 0xA6, 0xE8, 0x14, 0x89, 0x5B, 0xE4, 0x34, 0xF7,
 0xC5, 0xB0, 0x14, 0x93, 0xF5, 0x67, 0x7B, 0x3A, 0x7A, 0x78,
 0xE1, 0x01, 0x56, 0x56, 0x91, 0xA6, 0x13, 0x42, 0x8D, 0xD2,
 0x3C, 0x40, 0x9C, 0x4C, 0xEF, 0xD1, 0x86, 0xDF, 0x37, 0x51,
 0x1B, 0x0C, 0xA1, 0x3B, 0xF5, 0xF1, 0xA3, 0x4A, 0x35, 0xE4,
 0xE1, 0xCE, 0x96, 0xDF, 0x1B, 0x7E, 0xBF, 0x4E, 0x97, 0xD0,
 0x10, 0xE8, 0xA8, 0x08, 0x30, 0x81, 0xAF, 0x20, 0x0B, 0x43,
 0x14, 0xC5, 0x74, 0x67, 0xB4, 0x32, 0x82, 0x6F, 0x8D, 0x86,
 0xC2, 0x88, 0x40, 0x99, 0x36, 0x83, 0xBA, 0x1E, 0x40, 0x72,
 0x22, 0x17, 0xD7, 0x52, 0x65, 0x24, 0x73, 0xB0, 0xCE, 0xEF,
```

```
0x19, 0xCD, 0xAE, 0xFF, 0x78, 0x6C, 0x7B, 0xC0, 0x12, 0x03,
0xD4, 0x4E, 0x72, 0x0D, 0x50, 0x6D, 0x3B, 0xA3, 0x3B, 0xA3,
0x99, 0x5E, 0x9D, 0xC8, 0xD9, 0x0C, 0x85, 0xB3, 0xD9, 0x8A,
0xD9, 0x54, 0x26, 0xDB, 0x6D, 0xFA, 0xAC, 0xBB, 0xFF, 0x25,
0x4C, 0xC4, 0xD1, 0x79, 0xF4, 0x71, 0xD3, 0x86, 0x40, 0x18,
0x13, 0xB0, 0x63, 0xB5, 0x72, 0x4E, 0x30, 0xC4, 0x97, 0x84,
0x86, 0x2D, 0x56, 0x2F, 0xD7, 0x15, 0xF7, 0x7F, 0xC0, 0xAE,
0xF5, 0xFC, 0x5B, 0xE5, 0xFB, 0xA1, 0xBA, 0xD3, 0x02, 0x03,
0x01, 0x00, 0x01, 0x02, 0x82, 0x01, 0x01, 0x00, 0xA2, 0xE6,
0xD8, 0x5F, 0x10, 0x71, 0x64, 0x08, 0x9E, 0x2E, 0x6D, 0xD1,
0x6D, 0x1E, 0x85, 0xD2, 0x0A, 0xB1, 0x8C, 0x47, 0xCE, 0x2C,
0x51, 0x6A, 0xA0, 0x12, 0x9E, 0x53, 0xDE, 0x91, 0x4C, 0x1D,
0x6D, 0xEA, 0x59, 0x7B, 0xF2, 0x77, 0xAA, 0xD9, 0xC6, 0xD9,
0x8A, 0xAB, 0xD8, 0xE1, 0x16, 0xE4, 0x63, 0x26, 0xFF, 0xB5,
0x6C, 0x13, 0x59, 0xB8, 0xE3, 0xA5, 0xC8, 0x72, 0x17, 0x2E,
0x0C, 0x9F, 0x6F, 0xE5, 0x59, 0x3F, 0x76, 0x6F, 0x49, 0xB1,
0x11, 0xC2, 0x5A, 0x2E, 0x16, 0x29, 0x0D, 0xDE, 0xB7, 0x8E,
0xDC, 0x40, 0xD5, 0xA2, 0xEE, 0xE0, 0x1E, 0xA1, 0xF4, 0xBE,
0x97, 0xDB, 0x86, 0x63, 0x96, 0x14, 0xCD, 0x98, 0x09, 0x60,
0x2D, 0x30, 0x76, 0x9C, 0x3C, 0xCD, 0xE6, 0x88, 0xEE, 0x47,
0x92, 0x79, 0x0B, 0x5A, 0x00, 0xE2, 0x5E, 0x5F, 0x11, 0x7C,
0x7D, 0xF9, 0x08, 0xB7, 0x20, 0x06, 0x89, 0x2A, 0x5D, 0xFD,
0x00, 0xAB, 0x22, 0xE1, 0xF0, 0xB3, 0xBC, 0x24, 0xA9, 0x5E,
0x26, 0x0E, 0x1F, 0x00, 0x2D, 0xFE, 0x21, 0x9A, 0x53, 0x5B,
0x6D, 0xD3, 0x2B, 0xAB, 0x94, 0x82, 0x68, 0x43, 0x36, 0xD8,
0xF6, 0x2F, 0xC6, 0x22, 0xFC, 0xB5, 0x41, 0x5D, 0x0D, 0x33,
0x60, 0xEA, 0xA4, 0x7D, 0x7E, 0xE8, 0x4B, 0x55, 0x91, 0x56,
0xD3, 0x5C, 0x57, 0x8F, 0x1F, 0x94, 0x17, 0x2F, 0xAA, 0xDE,
0xE9, 0x9E, 0xA8, 0xF4, 0xCF, 0x8A, 0x4C, 0x8E, 0xA0, 0xE4,
0x56, 0x73, 0xB2, 0xCF, 0x4F, 0x86, 0xC5, 0x69, 0x3C, 0xF3,
0x24, 0x20, 0x8B, 0x5C, 0x96, 0x0C, 0xFA, 0x6B, 0x12, 0x3B,
0x9A, 0x67, 0xC1, 0xDF, 0xC6, 0x96, 0xB2, 0xA5, 0xD5, 0x92,
0x0D, 0x9B, 0x09, 0x42, 0x68, 0x24, 0x10, 0x45, 0xD4, 0x50,
0xE4, 0x17, 0x39, 0x48, 0xD0, 0x35, 0x8B, 0x94, 0x6D, 0x11,
0xDE, 0x8F, 0xCA, 0x59, 0x02, 0x81, 0x81, 0x00, 0xEA, 0x24,
0xA7, 0xF9, 0x69, 0x33, 0xE9, 0x71, 0xDC, 0x52, 0x7D, 0x88,
0x21, 0x28, 0x2F, 0x49, 0xDE, 0xBA, 0x72, 0x16, 0xE9, 0xCC,
0x47, 0x7A, 0x88, 0x0D, 0x94, 0x57, 0x84, 0x58, 0x16, 0x3A,
0x81, 0xB0, 0x3F, 0xA2, 0xCF, 0xA6, 0x6C, 0x1E, 0xB0, 0x06,
0x29, 0x00, 0x8F, 0xE7, 0x77, 0x76, 0xAC, 0xDB, 0xCA, 0xC7,
0xD9, 0x5E, 0x9B, 0x3F, 0x26, 0x90, 0x52, 0xAE, 0xFC, 0x38,
0x90, 0x00, 0x14, 0xBB, 0xB4, 0x0F, 0x58, 0x94, 0xE7, 0x2F,
```

```
0x6A, 0x7E, 0x1C, 0x4F, 0x41, 0x21, 0xD4, 0x31, 0x59, 0x1F,
0x4E, 0x8A, 0x1A, 0x8D, 0xA7, 0x57, 0x6C, 0x22, 0xD8, 0xE5,
0xF4, 0x7E, 0x32, 0xA6, 0x10, 0xCB, 0x64, 0xA5, 0x55, 0x03,
0x87, 0xA6, 0x27, 0x05, 0x8C, 0xC3, 0xD7, 0xB6, 0x27, 0xB2,
0x4D, 0xBA, 0x30, 0xDA, 0x47, 0x8F, 0x54, 0xD3, 0x3D, 0x8B,
0x84, 0x8D, 0x94, 0x98, 0x58, 0xA5, 0x02, 0x81, 0x81, 0x00,
0xD5, 0x38, 0x1B, 0xC3, 0x8F, 0xC5, 0x93, 0x0C, 0x47, 0x0B,
0x6F, 0x35, 0x92, 0xC5, 0xB0, 0x8D, 0x46, 0xC8, 0x92, 0x18,
0x8F, 0xF5, 0x80, 0x0A, 0xF7, 0xEF, 0xA1, 0xFE, 0x80, 0xB9,
0xB5, 0x2A, 0xBA, 0xCA, 0x18, 0xB0, 0x5D, 0xA5, 0x07, 0xD0,
0x93, 0x8D, 0xD8, 0x9C, 0x04, 0x1C, 0xD4, 0x62, 0x8E, 0xA6,
0x26, 0x81, 0x01, 0xFF, 0xCE, 0x8A, 0x2A, 0x63, 0x34, 0x35,
0x40, 0xAA, 0x6D, 0x80, 0xDE, 0x89, 0x23, 0x6A, 0x57, 0x4D,
0x9E, 0x6E, 0xAD, 0x93, 0x4E, 0x56, 0x90, 0x0B, 0x6D, 0x9D,
0x73, 0x8B, 0x0C, 0xAE, 0x27, 0x3D, 0xDE, 0x4E, 0xF0, 0xAA,
0xC5, 0x6C, 0x78, 0x67, 0x6C, 0x94, 0x52, 0x9C, 0x37, 0x67,
0x6C, 0x2D, 0xEF, 0xBB, 0xAF, 0xDF, 0xA6, 0x90, 0x3C, 0xC4,
0x47, 0xCF, 0x8D, 0x96, 0x9E, 0x98, 0xA9, 0xB4, 0x9F, 0xC5,
0xA6, 0x50, 0xDC, 0xB3, 0xF0, 0xFB, 0x74, 0x17, 0x02, 0x81,
0x80, 0x5E, 0x83, 0x09, 0x62, 0xBD, 0xBA, 0x7C, 0xA2, 0xBF,
0x42, 0x74, 0xF5, 0x7C, 0x1C, 0xD2, 0x69, 0xC9, 0x04, 0x0D,
0x85, 0x7E, 0x3E, 0x3D, 0x24, 0x12, 0xC3, 0x18, 0x7B, 0xF3,
0x29, 0xF3, 0x5F, 0x0E, 0x76, 0x6C, 0x59, 0x75, 0xE4, 0x41,
0x84, 0x69, 0x9D, 0x32, 0xF3, 0xCD, 0x22, 0xAB, 0xB0, 0x35,
0xBA, 0x4A, 0xB2, 0x3C, 0xE5, 0xD9, 0x58, 0xB6, 0x62, 0x4F,
0x5D, 0xDE, 0xE5, 0x9E, 0x0A, 0xCA, 0x53, 0xB2, 0x2C, 0xF7,
0x9E, 0xB3, 0x6B, 0x0A, 0x5B, 0x79, 0x65, 0xEC, 0x6E, 0x91,
0x4E, 0x92, 0x20, 0xF6, 0xFC, 0xFC, 0x16, 0xED, 0xD3, 0x76,
0x0C, 0xE2, 0xEC, 0x7F, 0xB2, 0x69, 0x13, 0x6B, 0x78, 0x0E,
0x5A, 0x46, 0x64, 0xB4, 0x5E, 0xB7, 0x25, 0xA0, 0x5A, 0x75,
0x3A, 0x4B, 0xEF, 0xC7, 0x3C, 0x3E, 0xF7, 0xFD, 0x26, 0xB8,
0x20, 0xC4, 0x99, 0x0A, 0x9A, 0x73, 0xBE, 0xC3, 0x19, 0x02,
0x81, 0x81, 0x00, 0xBA, 0x44, 0x93, 0x14, 0xAC, 0x34, 0x19,
0x3B, 0x5F, 0x91, 0x60, 0xAC, 0xF7, 0xB4, 0xD6, 0x81, 0x05,
0x36, 0x51, 0x53, 0x3D, 0xE8, 0x65, 0xDC, 0xAF, 0x2E, 0xDC,
0x61, 0x3E, 0xC9, 0x7D, 0xB8, 0x7F, 0x87, 0xF0, 0x3B, 0x9B,
0x03, 0x82, 0x29, 0x37, 0xCE, 0x72, 0x4E, 0x11, 0xD5, 0xB1,
0xC1, 0x0C, 0x07, 0xA0, 0x99, 0x91, 0x4A, 0x8D, 0x7F, 0xEC,
0x79, 0xCF, 0xF1, 0x39, 0xB5, 0xE9, 0x85, 0xEC, 0x62, 0xF7,
0xDA, 0x7D, 0xBC, 0x64, 0x4D, 0x22, 0x3C, 0x0E, 0xF2, 0xD6,
0x51, 0xF5, 0x87, 0xD8, 0x99, 0xC0, 0x11, 0x20, 0x5D, 0x0F,
0x29, 0xFD, 0x5B, 0xE2, 0xAE, 0xD9, 0x1C, 0xD9, 0x21, 0x56,
```

```
    0x6D, 0xFC, 0x84, 0xD0, 0x5F, 0xED, 0x10, 0x15, 0x1C, 0x18,
    0x21, 0xE7, 0xC4, 0x3D, 0x4B, 0xD7, 0xD0, 0x9E, 0x6A, 0x95,
    0xCF, 0x22, 0xC9, 0x03, 0x7B, 0x9E, 0xE3, 0x60, 0x01, 0xFC,
    0x2F, 0x02, 0x81, 0x80, 0x11, 0xD0, 0x4B, 0xCF, 0x1B, 0x67,
    0xB9, 0x9F, 0x10, 0x75, 0x47, 0x86, 0x65, 0xAE, 0x31, 0xC2,
    0xC6, 0x30, 0xAC, 0x59, 0x06, 0x50, 0xD9, 0x0F, 0xB5, 0x70,
    0x06, 0xF7, 0xF0, 0xD3, 0xC8, 0x62, 0x7C, 0xA8, 0xDA, 0x6E,
    0xF6, 0x21, 0x3F, 0xD3, 0x7F, 0x5F, 0xEA, 0x8A, 0xAB, 0x3F,
    0xD9, 0x2A, 0x5E, 0xF3, 0x51, 0xD2, 0xC2, 0x30, 0x37, 0xE3,
    0x2D, 0xA3, 0x75, 0x0D, 0x1E, 0x4D, 0x21, 0x34, 0xD5, 0x57,
    0x70, 0x5C, 0x89, 0xBF, 0x72, 0xEC, 0x4A, 0x6E, 0x68, 0xD5,
    0xCD, 0x18, 0x74, 0x33, 0x4E, 0x8C, 0x3A, 0x45, 0x8F, 0xE6,
    0x96, 0x40, 0xEB, 0x63, 0xF9, 0x19, 0x86, 0x3A, 0x51, 0xDD,
    0x89, 0x4B, 0xB0, 0xF3, 0xF9, 0x9F, 0x5D, 0x28, 0x95, 0x38,
    0xBE, 0x35, 0xAB, 0xCA, 0x5C, 0xE7, 0x93, 0x53, 0x34, 0xA1,
    0x45, 0x5D, 0x13, 0x39, 0x65, 0x42, 0x46, 0xA1, 0x9F, 0xCD,
    0xF5, 0xBF]
```

A.1.5 Configuration ECC

Structure de la clé utilisée pour ECC. En particulier, la courbe utilisée est SECP256R1 pour une taille de 32 octets.

```
{
    /* size/bytes */
    32,
    /* ID */
    ECC_SECP256R1,
    /* curve name */
    "SECP256R1",
    /* prime */
    "FFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFF",
    /* A */
    "FFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFC",
    /* B */
    "5AC635D8AA3A93E7B3EBBD55769886BC651D06B0CC53B0F63BCE3C3E27D2604B",
    /* order */
    "FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551",
    /* Gx */
    "6B17D1F2E12C4247F8BCE6E563A440F277037D812DEB33A0F4A13945D898C296",
    /* Gy */
```

```
    "4FE342E2FE1A7F9B8EE7EB4A7C0F9E162BCE33576B315ECECBB6406837BF51F5",
    /* oid/oidSz */
    ecc_oid_secp256r1,
    sizeof(ecc_oid_secp256r1) / sizeof(ecc_oid_t),
    /* oid sum */
    ECC_SECP256R1_OID,
    /* cofactor */
    1,
}
```

A.2 Performances temporelles des algorithmes utilisés

Algorithmes	3 octets	32 octets	64 octets	128 octets
MD5	45.9 μs	46.5 μs	85.4 μs	126.4 μs
SHA1	120.8 μs	120.8 μs	229.2 μs	338.8 μs
SHA256	300 μs	300.4 μs	583 μs	866 μs
SHA384	1072 μs	1072 μs	1072 μs	2116 μs
SHA512	1072 μs	1072 μs	1072 μs	2116 μs

Tab. A.1.: Durées des algorithmes de hachage

Algorithmes	3 octets	32 octets	64 octets	128 octets
DES	53 μs	220 μs	424 μs	840 μs
3DES	160 μs	520 μs	1030 μs	2040 μs
AES-128 (CBC)	136 μs	228 μs	404 μs	768 μs
AES-192 (CBC)	110 μs	208 μs	408 μs	808 μs
AES-256 (CBC)	112 μs	220 μs	436 μs	864 μs
AES-128 (CTR)	136 μs	228 μs	404 μs	768 μs
AES-192 (CTR)	100 μs	208 μs	380 μs	744 μs
AES-256 (CTR)	100 μs	192 μs	380 μs	748 μs

Tab. A.2.: Durées des algorithmes de chiffrement symétrique

Algorithmes	3 octets	32 octets	64 octets	128 octets
HMAC MD5	177 μs	177 μs	216 μs	256 μs
HMAC SHA1	463 μs	463 μs	512 μs	682 μs
HMAC SHA256	1174 μs	1174 μs	1468 μs	1748 μs
HMAC SHA384	4200 μs	4200 μs	4200 μs	5220 μs
HMAC SHA512	4200 μs	4200 μs	4200 μs	5220 μs

Tab. A.3.: Durées des algorithmes de HMAC

Algorithmes	3 octets	32 octets	64 octets	128 octets
RSA 1024	156.4 ms	156.4 ms	156.4 ms	156.4 ms
RSA 2048	598 ms	598 ms	598 ms	598 ms
ECC	638 ms	638 ms	638 ms	638 ms

Tab. A.4.: Durées des algorithmes de chiffrement asymétrique

Algorithmes	3 octets	32 octets	64 octets	128 octets
RSA 1024	1312 ms	1312 ms	1312 ms	1312 ms
RSA 2048	9560 ms	9560 ms	9560 ms	9560 ms
ECC	1272 ms	1270 ms	1256 ms	1300 ms

Tab. A.5.: Durées des algorithmes de déchiffrement asymétrique

B Consommation énergétique de l'émetteur

Ces mesures de consommation ont été réalisées sur un système embarqué équipé d'un micro-contrôleur (PIC32MX795F512L), d'une antenne, d'un capteur de pression, d'une antenne d'émission et d'une batterie de 155 mAh.

B.1 Consommation énergétique d'un cycle - Paramétrage 1

Dans cet exemple, la sécurité est absente, le message est envoyé une seule et unique fois par évènement, et la veille dure 10s.

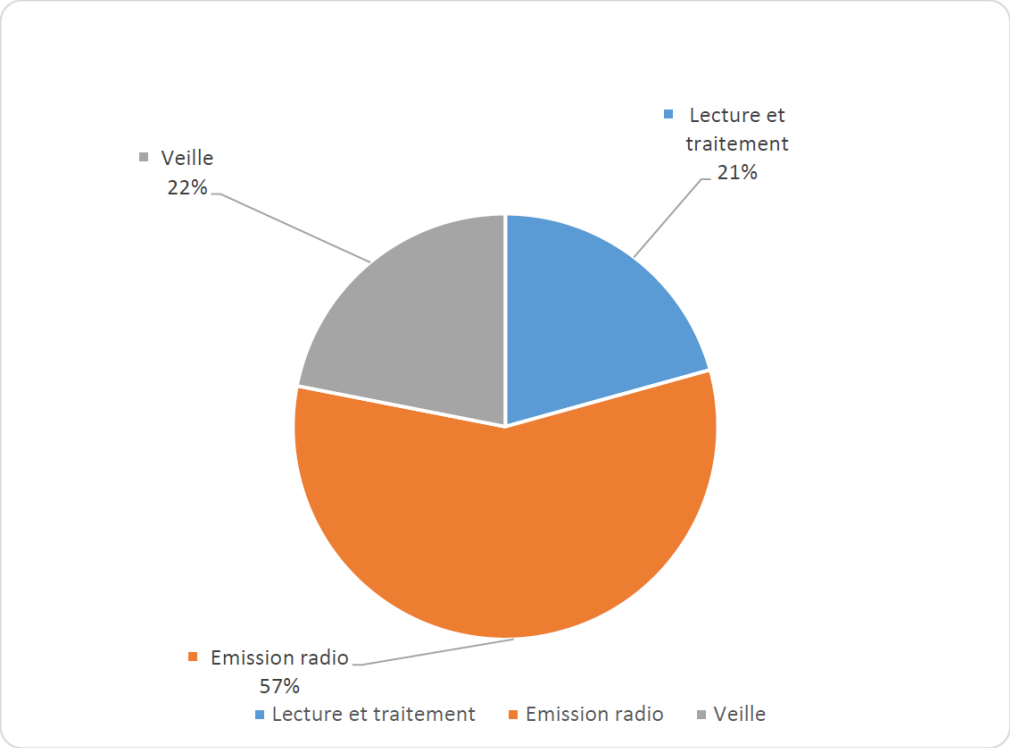

Fig. B.1.: Consommation énergétique d'un cycle - Paramétrage 1

Catégorie	Variable	Valeur	Unité
Données	Conso lecture capteur C_{mes}	33.30	mA
	Durée lecture capteur T_{mes}	0.134	ms
	Conso traitement C_{trait}	33.30	mA
	Durée traitement T_{trait}	1	ms
Sécurité	Conso hash C_{hash}	33.30	mA
	Durée hash T_{hash}	0	ms
	Conso chiffrement C_{chiff}	33.30	mA
	Durée chiffrement T_{chiff}	0	ms
	Conso authentification C_{auth}	33.30	mA
	Durée authentification T_{auth}	0	ms
Envoi	Conso émission C_{envoi}	36.64	mA
	Durée émission T_{envoi}	2.856	ms
	Nombre de répétition Nb_{rep}	0	
Veille	Durée moyenne entre chaque répétition T_{rep}	75	ms
	Durée avant prochain cycle T_{sigvie}	10 000	ms
	Conso veille C_{veille}	0.004	mA
Cycle	Durée maximale d'un cycle	10 004	ms
	Conso moyenne d'un cycle	0.01823	mA
Autonomie	Temps d'utilisation	8500.92	heures
	Temps d'utilisation	354.20	jours

Tab. B.1.: Consommation énergétique d'un cycle - Paramétrage 1

B.2 Consommation énergétique d'un cycle - Paramétrage 2

Dans cet exemple, la sécurité consiste en un HMAC-SHA-512 pour le hachage et l'authentification (regroupé dans la partie authentification du tableau) et un chiffrement AES-256. Le message est envoyé deux fois (une répétition) par évènement, et la veille dure 10s.

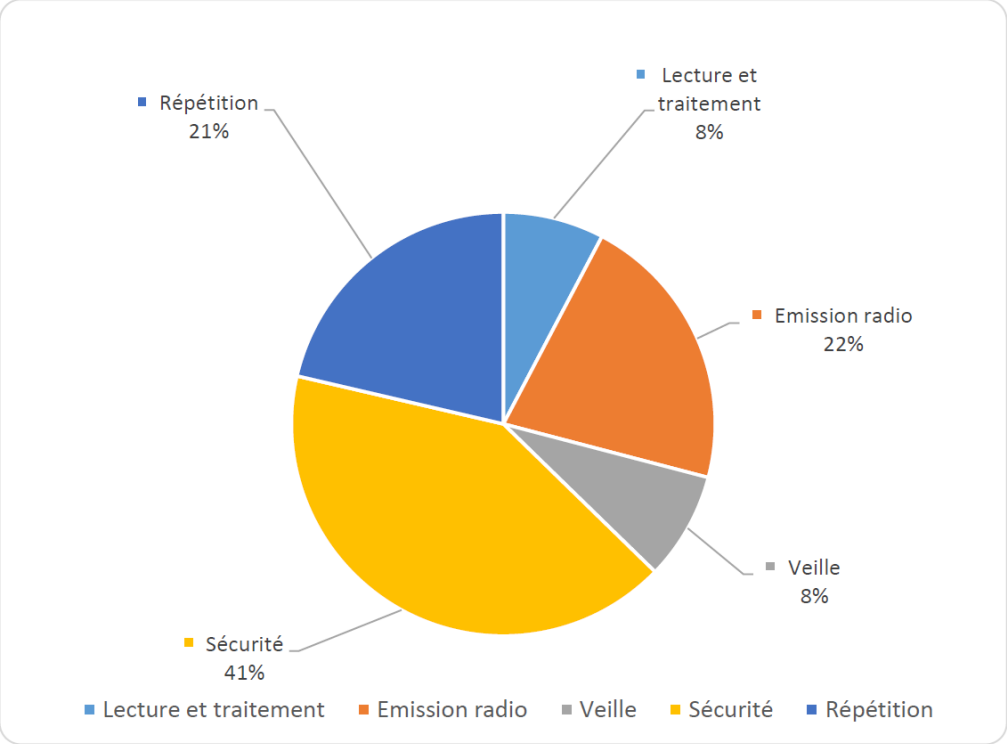

Fig. B.2.: Consommation énergétique d'un cycle - Paramétrage 2

Catégorie	Variable	Valeur	Unité
Données	Conso lecture capteur C_{mes}	33.30	mA
	Durée lecture capteur T_{mes}	0.134	ms
	Conso traitement C_{trait}	33.30	mA
	Durée traitement T_{trait}	1	ms
Sécurité	Conso hash C_{hash}	33.30	mA
	Durée hash T_{hash}	0	ms
	Conso chiffrement C_{chiff}	33.30	mA
	Durée chiffrement T_{chiff}	0.864	ms
	Conso authentification C_{auth}	33.30	mA
	Durée authentification T_{auth}	5.22	ms
Envoi	Conso émission C_{envoi}	36.64	mA
	Durée émission T_{envoi}	2.856	ms
	Nombre de répétition Nb_{rep}	1	
Veille	Durée moyenne entre chaque répétition T_{rep}	75	ms
	Durée avant prochain cycle T_{sigvie}	10 000	ms
	Conso veille C_{veille}	0.004	mA
Cycle	Durée maximale d'un cycle	10 087.9	ms
	Conso moyenne d'un cycle	0.04857	mA
Autonomie	Temps d'utilisation	3191.42	heures
	Temps d'utilisation	132.98	jours

Tab. B.2.: Consommation énergétique d'un cycle - Paramétrage 2

B.3 Consommation énergétique d'un cycle - Paramétrage 3

Dans cet exemple, la sécurité consiste en un hachage SHA-512, une authentification RSA-1024 (clé privée "courte" utilisée) et un chiffrement AES-256. Le message est envoyé trois fois (deux répétitions) par évènement, et la veille dure 1s.

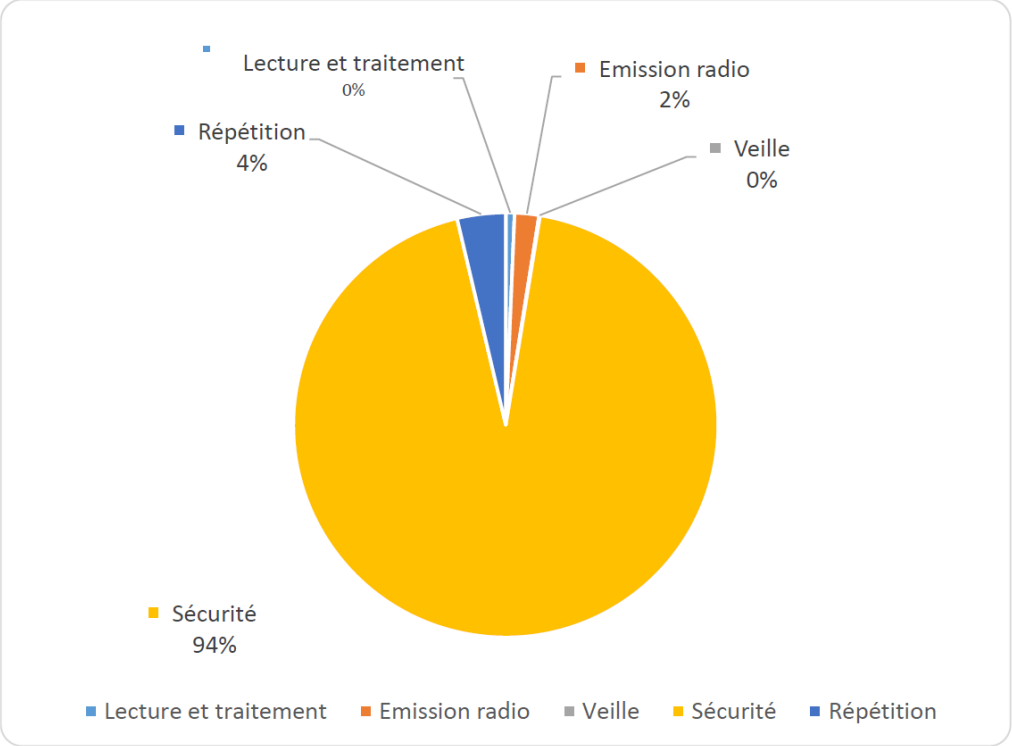

Fig. B.3.: Consommation énergétique d'un cycle - Paramétrage 3

Catégorie	Variable	Valeur	Unité
Données	Conso lecture capteur C_{mes}	33.30	mA
	Durée lecture capteur T_{mes}	0.134	ms
	Conso traitement C_{trait}	33.30	mA
	Durée traitement T_{trait}	1	ms
Sécurité	Conso hash C_{hash}	33.30	mA
	Durée hash T_{hash}	2.116	ms
	Conso chiffrement C_{chiff}	33.30	mA
	Durée chiffrement T_{chiff}	0.864	ms
	Conso authentification C_{auth}	33.30	mA
	Durée authentification T_{auth}	156.4	ms
Envoi	Conso émission C_{envoi}	36.64	mA
	Durée émission T_{envoi}	2.856	ms
	Nombre de répétition Nb_{rep}	2	
Veille	Durée moyenne entre chaque répétition T_{rep}	75	ms
	Durée avant prochain cycle T_{sigvie}	1 000	ms
	Conso veille C_{veille}	0.004	mA
Cycle	Durée maximale d'un cycle	1319.08	ms
	Conso moyenne d'un cycle	4.29363	mA
Autonomie	Temps d'utilisation	36.10	heures
	Temps d'utilisation	1.5	jours

Tab. B.3.: Consommation énergétique d'un cycle - Paramétrage 3

C

Durée des tâches d'un récepteur WSN

Les durées de chaque tâche du récepteur et la capacité du *buffer* utilisées pour l'étude sont détaillées dans le tableau C.1.

Catégorie	Variable	Valeur	Unité
Données	Durée traitement Tr_{trait}	1	ms
Sécurité	Durée hash T_{hash}	Selon scénario	ms
	Durée chiffrement T_{chiff}	Selon scénario	ms
	Durée authentification T_{auth}	Selon scénario	ms
Réception	Durée de réception T_r	2.856	ms
	Capacité du buffer $CapBuff$	1	

Tab. C.1.: Durée des tâches du récepteur

Table des figures

Liste des tableaux

Déclaration

Ce document a été rédigé avec LaTeX. Il utilise le template *Clean Thesis* développé par Ricardo Langner. Le design *Clean Thesis* a été inspiré par les guides d'utilisation d'Apple Inc.

Ce travail a été réalisé sous la supervision de Rémi Cogranne et Patrick Lallement avec le financement de VIGILOCK. Les images non sourcées de ce document ont été réalisées par l'auteur de ce document avec l'aide graphique de Lou Grimal. Les références citées ont grandement aidé à l'élaboration de cette thèse.

Troyes, Jeudi 3 Septembre 2020

Nicolas BURGER

Doctorat : Optimisation et Sûreté des Systèmes

Année 2020

Étude des compromis entre sécurité et disponibilité des systèmes embarqués communicants

La sécurité d'un moyen de communication dépend d'algorithmes de sécurité qui reposent sur des calculs souvent lourds, en particulier au sein d'un système embarqué par rapport à ses autres tâches. Dans un tel contexte, les algorithmes de sécurité consomment d'importantes ressources temporelles et énergétiques. La sécurité devient alors un curseur à faire varier grâce à des choix d'algorithmes et de paramètres adaptés aux besoins du système embarqué communicant. Trouver le bon compromis entre sécurité, disponibilité et autonomie d'un réseau de capteurs sans fil est délicat pour un concepteur d'objets connectés. L'objectif de cette thèse est de définir les paramètres sur lesquels il est possible de jouer pour trouver une solution acceptable. Un modèle sous forme de réseaux de Petri est proposé pour faire interagir ces paramètres et mettre en évidence leurs influences sur un réseau de capteur sans fil.

Mots clés : systèmes embarqués (informatique) – réseaux de Petri – internet des objets – microcontrôleurs – traitement d'événements (informatique) – cryptographie – systèmes informatiques, mesures de sûreté.

Study of the Trade-offs between Security and Availability of Communicating Embedded Systems

The security of communications depends on security algorithms which are often based on heavy calculations, in particular within an embedded system in view of its other tasks. In such a context, security algorithms consume significant time and energy resources. Security then becomes a slider, a slider that one can move by choosing the right algorithms and parameters for the needs of the embedded system. Finding the right compromise between security, availability and autonomy of a wireless sensor network is difficult for an embedded systems designer. The objective of this thesis is to define the parameters on which it is possible to act to find an acceptable solution. A model in the form of Petri nets is offered to make these parameters interact and highlight their mutual influences.

Keywords: embedded computer systems – Petri nets – internet of things – programmable controllers – event processing (computer science) – cryptography – electronic data processing departments, security measures.

Thèse réalisée en partenariat entre :

VigiLock

utt UNIVERSITÉ DE TECHNOLOGIE TROYES

Ecole Doctorale "Sciences pour l'Ingénieur"