WazaBee : attaque de réseaux Zigbee par détournement de puces Bluetooth Low Energy

Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette, Geraldine Marconato

WazaBee : attaque de réseaux Zigbee par détournement de puces Bluetooth Low Energy

Romain Cayre[1,3], Florent Galtier[1], Guillaume Auriol[1,2], Vincent Nicomette[1,2] et Géraldine Marconato[3]

prenom.nom@laas.fr

prenom.nom@airbus.com

[1] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[2] Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France
[3] APSYS.Lab, APSYS

Résumé. Le développement rapide et massif des objets connectés a eu de multiples conséquences sur la sécurité des systèmes d'information. L'une des plus importantes a été le déploiement de nouvelles technologies sans fil adaptées à ces systèmes d'un nouveau genre et à leurs contraintes, telles que l'économie d'énergie et des architectures matérielles aux performances limitées. Des technologies telles que le *Zigbee*, le *Bluetooth Low Energy* (*BLE*) ou le *Z-Wave* ont ainsi vu le jour, et sont aujourd'hui de plus en plus fréquemment rencontrées dans les environnements domestiques comme professionnels. De plus, leur développement concurrent a mené à un déploiement simultané de ces technologies, qui co-existent aujourd'hui. Dans cet article, nous présentons une nouvelle stratégie d'attaque pivot nommée *WazaBee*, permettant de détourner le fonctionnement de puces *BLE* couramment utilisées dans l'Internet des objets, ou *IoT*, afin d'attaquer des réseaux d'une autre technologie, dans notre cas, les réseaux *Zigbee* environnants. Autrement dit, nous montrons qu'il est possible de modifier le firmware d'une puce *BLE* afin de réaliser des communications *Zigbee* par l'intermédiaire de celle ci. Nous présentons les bases théoriques de l'attaque et nous décrivons les expériences qui nous ont permis de mettre en oeuvre avec succès cette attaque en pratique.

1 Introduction

Il est aujourd'hui évident que le déploiement rapide et massif des objets connectés pose des problèmes de sécurité majeurs, tant pour la sécurité de ces nouveaux systèmes que pour celle des infrastructures informatiques traditionnelles. Ces problèmes constituent de véritables défis pour la communauté scientifique, et il devient urgent d'identifier les risques nouveaux associés à ces systèmes ainsi que les contre-mesures adaptées.

De nombreux facteurs sont responsables de cette situation, en premier lieu des problématiques d'ordres économique et industriel : en effet,

le *time-to-market* impose aux fabricants de développer en continu de nouveaux systèmes intégrant toujours plus de fonctionnalités, tout en conservant des coûts de production bas. Ces contraintes peuvent ainsi amener les fabricants à privilégier l'ajout de nouvelles fonctionnalités au détriment de la sécurité, qui s'inscrit dans un processus de long terme potentiellement coûteux dont l'intérêt n'est pas immédiatement perceptible par l'utilisateur final. Le fait que les fabricants d'objets connectés ne soient généralement pas issus de l'industrie logicielle traditionnelle (petit électroménager, électronique, ...) est également problématique, car ils ne disposent pas de la culture de la sécurité et des effets d'apprentissage produits par les attaques successives et répétées qu'a pu subir l'industrie du logiciel.

D'autres facteurs de risques tiennent principalement à un certain nombre de caractéristiques propres à ces systèmes. On peut ainsi souligner l'usage fréquent de micro-contrôleurs peu puissants pour la conception de ces objets, principalement choisis pour des motifs d'économie d'énergie, compliquant l'implémentation de certaines fonctionnalités de sécurité couramment utilisées dans l'informatique traditionnelle, comme certaines fonctions cryptographiques notamment.

Enfin, l'une des caractéristiques récurrentes de ces objets est la connectivité sans fil. En effet, les objets connectés embarquent des composants radios permettant de communiquer avec d'autres systèmes par l'intermédiaire de nouveaux protocoles sans fil. Ces protocoles sont aujourd'hui nombreux et sont activement développés : on a ainsi vu apparaître de nombreuses technologies sans fil nouvelles ces dernières années, proposant des fonctionnalités telles que la faible consommation énergétique, des piles protocolaires simplifiées, clairement orientées pour conquérir ce nouveau marché. Certaines de ces technologies, telles que le *Zigbee* [23], le *Bluetooth Low Energy* (*BLE*) [21, 22] ou le *Z-Wave* [10] sont aujourd'hui bien connues techniquement, y compris du point de vue des risques en termes de sécurité, tandis que d'autres, plus obscures et parfois propriétaires, restent peu étudiées, rendant l'impact de leur déploiement plus difficile à évaluer.

Au delà des vulnérabilités inhérentes à un protocole sans fil donné ou à l'implémentation des piles protocolaires associées, le déploiement conjoint de ces technologies sans fil dans le même environnement pose également problème. En effet, il n'est pas rare de trouver dans un même environnement professionnel ou domestique de multiples objets, communiquant chacun avec des protocoles sans fil différents. De plus, certains de ces objets sont destinés à un usage mobile : une montre connectée, par

exemple, est amenée à être transportée par l'utilisateur aussi bien sur son lieu de travail qu'à son domicile, exposant de multiples environnements à une attaque potentielle.

Dans ces conditions, s'interroger sur l'impact de cette coexistence de protocoles sans fil en termes de surface d'attaque devient indispensable. L'un des risques majeurs de déployer de tels environnements est évidemment la possibilité pour un attaquant d'utiliser un système préalablement compromis pour «pivoter» sur d'autres systèmes d'information à proximité par l'intermédiaire de ces protocoles sans fil. Dans cet article, nous nous concentrerons sur une problématique particulière et peu étudiée sous l'angle de la sécurité : est-il possible de détourner le comportement d'un composant radio prévu pour communiquer avec un protocole donné pour le faire communiquer avec un protocole différent ?

L'impact d'une telle problématique nous semble particulièrement critique, car elle ouvre potentiellement la voie à des scénarios de compromission nouveaux et difficiles à anticiper du point de vue défensif. Nous focaliserons notre travail sur deux protocoles couramment utilisés au sein de l'Internet des objets, le *BLE* et le *Zigbee*. Notre contribution principale sera la présentation technique d'une nouvelle stratégie d'attaque pivot nommée *WazaBee*, exploitant certaines caractéristiques du *BLE* pour détourner des puces destinées à l'usage de cette technologie afin de les faire communiquer avec un autre protocole, le *Zigbee*.

2 Stratégies d'attaques pivot existantes

Dans cette section, nous présentons brièvement les différentes stratégies existantes permettant de mettre en place une attaque pivot vers un autre protocole. Dans un premier temps, nous décrivons les solutions liées à l'utilisation de composants radios multi-protocoles, puis nous dressons un panorama des travaux existants dans la littérature scientifique permettant de mettre en place une telle attaque sur un composant dédié à un protocole spécifique.

2.1 Les composants matériels multi-protocoles

L'objectif principal d'une attaque pivot étant de tirer profit de la coexistence de multiples protocoles dans le même environnement afin de compromettre de nouveaux systèmes, l'approche la plus naturelle consiste à compromettre un composant matériel supportant plusieurs protocoles de communication.

Différents types de composants matériels offrent cette possibilité. Les *Software Defined Radios*, par exemple, sont des composants radios dont l'architecture est conçue dans un objectif de généricité, autorisant de fait la communication avec de multiples protocoles sans limites liées à la modulation ou aux bandes de fréquences utilisées. Ces composants matériels ne sont cependant généralement pas déployés au sein des objets connectés et sont plus destinés à un usage de prototypage ou de recherche.

Il existe également des puces intégrant des composants radios multi-protocoles. C'est par exemple le cas du *B-L475E-IOT01A* [2], un système basé sur le microcontrôleur *STM32L4* destiné au développement *IoT* et intégrant de multiples protocoles radios (tels que le *Bluetooth*, le *WiFi* ou le *NFC*). De même, le *CC2652R* [3] de *Texas Instruments* supporte de multiples technologies dans la bande 2.4 à 2.5GHz. La compromission d'une telle puce facilite grandement la mise en place d'une attaque pivot visant les technologies nativement supportées par le composant. Cependant, leur coût élevé et le fait que l'utilisation de telles puces multi-protocoles ne soit généralement intéressante que pour un nombre très restreint d'équipements (de type *gateway*, notamment) limitent de facto leur déploiement dans l'*IoT*.

2.2 Les composants matériels mono-protocole

La plupart des objets connectés étant basés sur des composants matériels ne supportant qu'un seul protocole de communication, la mise en oeuvre pratique d'une stratégie d'attaque pivot est beaucoup plus complexe. A notre connaissance, il n'existe pas dans la littérature scientifique de travaux ayant exploré spécifiquement cette problématique sous l'angle de la sécurité. Cependant, certaines contributions ont exploré des thématiques connexes.

Les contributions les plus pertinentes sont sans doute celles concernant les *Cross-Technology Communication* (ou *CTC*) : ces travaux ont pour vocation de fournir des solutions de communications entre des composants matériels mono-protocoles supportant des technologies sans fil hétérogènes. Deux grandes approches peuvent être distinguées : les *Packet-Level CTC* et les *PHY-layer CTC*. Les *Packet-Level CTC* permettent d'établir un canal de communication par l'intermédiaire d'informations liées aux paquets. Par exemple, K. Chebrolu et A. Dhekne proposent d'utiliser la durée des paquets pour transporter de l'information [9], tandis que S. Min Kim et T. He défendent dans leur approche *FreeBee* [15] l'utilisation de l'intervalle entre des trames *beacons* afin d'atteindre cet objectif. D'un

point de vue offensif, ces approches peuvent être utilisées dans une optique d'exfiltration de données, mais ne permettent pas d'envisager des scénarios de type attaque pivot. De plus, des limitations importantes intrinsèques aux approches décrites, notamment en terme de débit, compliquent considérablement leur déploiement en pratique.

Les *PHY-layer CTC*, basées sur le détournement de la couche physique des protocoles supportés, sont beaucoup plus pertinentes vis à vis de notre problématique : notre stratégie offensive peut être assimilée à ce type de *CTC*. On peut ainsi citer les travaux de Z. Li et T. He [16], ayant permis de simuler une trame *Zigbee* par l'intermédiaire d'un émetteur *WiFi*. De même, W. Jiang et al ont successivement proposé l'approche *BlueBee* [14], permettant de générer une trame *Zigbee* par l'intermédiaire d'un émetteur *BLE*, et *XBee* [13], démontrant la possibilité sous certaines conditions de recevoir une trame *Zigbee* depuis un récepteur *BLE*. Cependant, ces travaux souffrent de limitations importantes, qui empêchent leur utilisation dans un cadre offensif, notamment dans une optique d'attaque pivot. A titre d'exemple, la sélection du canal *Zigbee* par *BlueBee* est basée sur un détournement du mécanisme de saut de fréquence du mode connecté du *BLE*, nécessitant donc qu'une connection soit préalablement établie. De même, la réception de trames *Zigbee* par *XBee* n'est possible que si les données à transmettre sont préfixées d'un identifiant précis, nécessitant donc une coopération de l'émetteur *Zigbee*. Ces contraintes peuvent être facilement résolues dans un cadre fonctionnel, où une communication *CTC* légitime pourrait être intentionnellement mise en place, mais rendent ces solutions trop limitées dans le cadre d'une stratégie offensive de type attaque pivot. Notre approche s'affranchit de ces limites en proposant une solution *CTC* bidirectionnelle, fiable et n'impliquant aucune collaboration des équipements environnants, la rendant utilisable dans un contexte offensif.

Une autre contribution intéressante est la présentation d'une stratégie d'attaque nommée *Packet-in-Packet* [12]. Proposée par T. Goodspeed et al., elle consiste à encapsuler une trame complète dans une *payload* de niveau applicatif : une mauvaise interprétation du début de la trame légitime par le récepteur (par exemple en raison d'interférences ayant provoqué l'occurence de *bitflips* dans la démodulation) aura alors pour conséquence l'interprétation de la trame injectée. Cette stratégie est particulièrement intéressante pour pouvoir contourner des vérifications logicielles effectuées par la couche protocolaire, et peut ainsi permettre d'accéder à un contrôle bas niveau partiel du composant radio. Cependant, l'auteur souligne une application possible particulièrement intéressante dans le cadre du dé-

veloppement d'une attaque pivot : cette technique offensive peut être employée pour injecter du trafic correspondant à une autre technologie, en encapsulant une trame correspondant à un protocole différent. Il est ainsi possible dans certaines conditions de développer une primitive d'émission visant un protocole différent de celui supporté initialement par le composant radio. Cependant, cette stratégie reste limitée à un nombre restreint de protocoles, ceux ci n'étant atteignables que sous réserve que les modulations employées présentent des caractéristiques similaires (bande de fréquences, débit de données, etc). Par exemple, M. Millian et V. Yadav consacrent un article à l'application de cette stratégie d'attaque sur des trames 802.15.4 [17], et évoquent en perspective la possibilité d'injecter du trafic 802.15.4 en l'encapsulant dans des trames 802.11. Ils soulignent cependant la difficulté d'une telle stratégie, liées à de multiples différences entre les technologies concernées.

Une autre contribution de T. Goodspeed peut également être mise en avant : il a en effet présenté une vulnérabilité touchant la puce *nRF24L01+* et facilitant l'écoute passive et l'injection de trames sur un ensemble de protocoles utilisant la modulation *Gaussian Frequency Shift Keying* (tels que le *Bluetooth Low Energy* ou l'*Enhanced ShockBurst*). Il est en effet possible de détourner l'usage d'un registre destiné à la sélection d'adresse afin de sélectionner un préambule arbitraire [11]. L'usage naturel de cette vulnérabilité est l'ajout d'un mode *promiscuous* pour le protocole *Enhanced ShockBurst*, celui ci étant supporté nativement par la puce. Cependant, il est également possible de détourner ce registre afin de détecter les préambules utilisés par des technologies différentes, à la condition que celles ci utilisent des modulations et des débits similaires. Cette vulnérabilité a ainsi permis à M. Newlin d'écrire, dans le cadre de ses recherches sur les vulnérabilités des périphériques d'entrée sans fil (MouseJack [18]), un *firmware* dotant le *nRF24* de fonctionnalités d'écoute passive avancées pour le protocole *Enhanced ShockBurst*, mais également pour certains protocoles similaires comme le protocole *Mosart*, par exemple.

Dans le prolongement de ces travaux, D. Cauquil a constaté la présence d'une vulnérabilité similaire sur d'autres modèles de puces de *Nordic Semiconductors* [6], amenant ainsi au développement d'un outil similaire pour le *nRF51*. Il a ainsi pu utiliser cette fonctionnalité pour implémenter des primitives de communication avec un protocole propriétaire non initialement supporté par la puce, permettant de contrôler un mini-drone [7]. Une implémentation de ces primitives est proposée dans le cadre du projet *radiobit* [5].

Ces travaux apportent une réponse partielle à notre problématique : il semble en effet possible de développer des primitives de communication à destination de protocoles non initialement supportés par les composants matériels concernés. Cependant, plusieurs limites restreignent fortement l'usage de ces techniques : celles-ci sont en effet restreintes à des technologies utilisant des modulations similaires ou impliquent une collaboration active des équipements environnants, et sont parfois dépendantes de l'utilisation de puces spécifiques (notamment les puces *Nordic SemiConductors nRF24* et *nRF51*). Notre contribution principale présente une stratégie d'attaque pivot s'affranchissant de certaines de ces contraintes, permettant l'implémentation de primitives de communication visant une technologie présentant une modulation différente de celle initialement supportée par les puces étudiées et ne reposant pas sur la collaboration d'équipements tiers, dont l'usage pourrait être généralisé à de multiples composants matériels de constructeurs différents.

3 Présentation des protocoles étudiés

Dans cette partie, nous introduisons des notions et des définitions utiles à la compréhension de la stratégie d'attaque. Nous décrivons ensuite succinctement le fonctionnement des couches inférieures des protocoles *BLE* et *Zigbee*.

3.1 Modulation numérique

Les protocoles de communication sans fil ont pour objectif la transmission d'informations numériques d'une source à un (ou plusieurs) destinataire(s), véhiculées par la propagation d'ondes radio-fréquences. La mise en oeuvre de cette transmission est réalisée par des techniques de **modulation numérique**.

La **modulation numérique** est définie comme le processus par lequel un signal numérique (le signal modulant) est transformé en un signal adapté au canal de transmission. Cette transformation s'effectue généralement en faisant varier les caractéristiques d'une onde sinusoïdale nommée porteuse en fonction des données à transmettre : le signal résultant, nommé signal modulé, peut ainsi être transmis par l'intermédiaire du canal de transmission. Un processus symétrique, la **démodulation**, permet alors au récepteur d'effectuer l'opération de transformation inverse, pour extraire du signal modulé l'information initiale.

La porteuse étant définie comme une onde sinusoïdale, l'équation qui la représente est la suivante :

$$A\cos(2\pi f_c t + \phi) \tag{1}$$

avec :

— A l'amplitude de la porteuse,
— f_c la fréquence de la porteuse,
— ϕ la phase de la porteuse.

Une modulation numérique peut ainsi faire varier l'amplitude, la fréquence et/ou la phase de la porteuse en fonction du signal modulant. Le signal modulé peut donc être exprimé comme une sinusoïde dont les paramètres sont variables en fonction du temps :

$$s(t) = A(t)\cos(2\pi f_c t + \phi(t)) \tag{2}$$

Il est également possible de représenter l'état du signal modulé à un instant donné en le représentant par un vecteur dans le plan complexe. La norme de ce vecteur représente alors l'amplitude du signal, tandis que son argument correspond à la phase (instantanée). En effet, d'après la relation trigonométrique suivante :

$$\cos(\alpha + \beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta) \tag{3}$$

On peut développer la formule (2) sous la forme :

$$\begin{aligned} s(t) &= A(t)\cos(2\pi f_c t + \phi(t)) && (4)\\ &= A(t)\cos(2\pi f_c t)\cos(\phi(t)) - A(t)\sin(2\pi f_c t)\sin(\phi(t)) && (5)\\ &= I(t)\cos(2\pi f_c t) - Q(t)\sin(2\pi f_c t) && (6) \end{aligned}$$

avec :

— $I(t) = A(t)\cos(\phi(t))$ la composante dite «en phase»,
— $Q(t) = A(t)\sin(\phi(t))$ la composante dite «en quadrature».

Il est ainsi possible de représenter sous forme vectorielle dans le plan complexe l'état du signal à un instant donné. Cette représentation, illustrée en figure 1a, est couramment utilisée en traitement du signal, notamment pour représenter une modulation graphiquement (on parle alors de constellation). On peut également noter que l'équation (6) démontre qu'il est possible de contrôler la phase instantanée, la fréquence instantanée et l'amplitude du signal modulé en manipulant l'amplitude des signaux I et Q. Cette propriété est la base d'un type de modulateur dit I/Q, illustré en figure 1b.

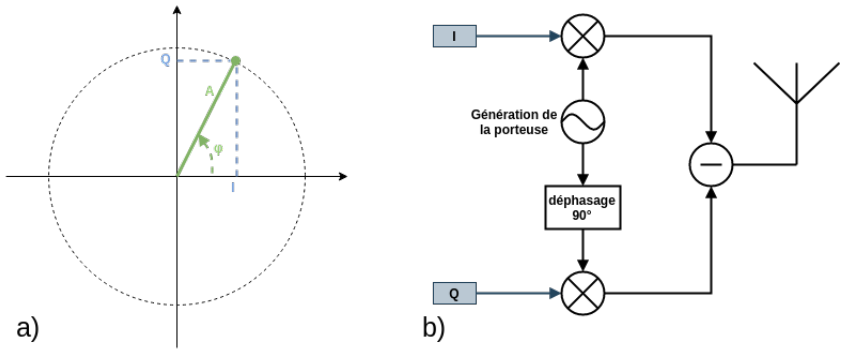

Fig. 1. a) Représentation I/Q de l'état d'un signal à un instant donné, b) Modulateur I/Q

3.2 Le Bluetooth Low Energy (BLE)

Le protocole *Bluetooth Low Energy*, ou *BLE*, est une variante simplifiée du protocole *Bluetooth* classique destinée aux objets connectés, introduite dans la version 4.0 de la spécification *Bluetooth* [4]. Il est notamment optimisé pour l'économie d'énergie, et son usage se généralise dans l'*IoT* en raison de sa faible complexité et de son large déploiement (il est supporté par défaut par la plupart des smartphones et des ordinateurs).

On s'intéresse ici principalement aux couches inférieures du protocole, notamment la couche physique. Une description plus complète de ce protocole et des problématiques de sécurité associées a été présentée à *SSTIC* [8].

La couche physique du protocole (couche *PHY*) prévoit un format de paquet unique, illustré en figure 2 et composé des champs suivants :

— **Préambule :** champ d'un octet correspondant à une série de bits alternés (0x55), utilisé pour synchroniser le récepteur sur le début de la trame,

— **Access Address :** adresse unique composée de 4 octets, permettant d'identifier de façon unique une connexion particulière (en mode connecté) ou un paquet d'annonce (ou *advertisement*),

— **Protocol Data Unit (PDU) :** champ de taille variable contenant une entête niveau liaison (dite *LL Header*) et les données à transmettre,

— **Cyclic Redundancy Check (CRC) :** champ de 3 octets destiné au contrôle d'intégrité, par l'intermédiaire d'un contrôle de redondance cyclique.

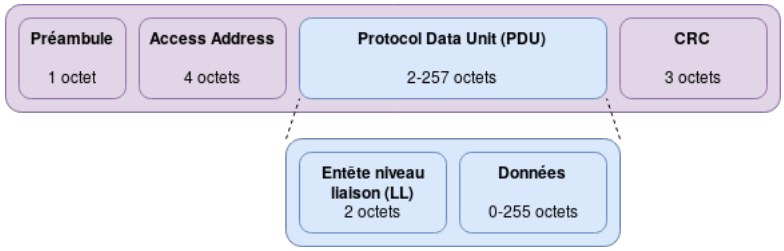

Fig. 2. Format d'une trame *BLE*

Lors de la transmission d'une trame, la donnée transmise par les couches supérieures est donc préfixée d'une entête par la couche liaison (dite *LL*), et encapsulée dans le champ PDU. Le CRC correspondant est calculé et la valeur obtenue est ajoutée à la suite du PDU. Une transformation nommée *whitening* est ensuite appliquée, permettant de construire une séquence pseudo-aléatoire afin d'éviter la présence de longues séquences répétées de 1 ou 0 qui pourraient être problématiques lors de la transmission du signal modulé. Finalement, le préambule et l'Access Address sont ajoutées avant le PDU, et la trame est ensuite fournie en entrée du modulateur.

La couche physique du *BLE* est basée sur une modulation de fréquence, nommée *Gaussian Frequency Shift Keying* (*GFSK*), opérant dans la bande 2.4 à 2.5 GHz. Il s'agit d'une variante de la modulation dite *2-Frequency Shift Keying* (*2-FSK*) : en effet, à la différence de cette dernière, le signal modulant est fourni en entrée d'un filtre gaussien. Cette transformation est destinée à éviter les changement brutaux de fréquence lors du changement de symbole.

Une modulation de type *2-FSK* va donc coder deux symboles (0 et 1 dans le cas de données binaires) par deux fréquences différentes. Ces fréquences sont calculées grâce aux relations suivantes :

$$F_0 = f_c - \Delta f = f_c - \frac{m}{2T_s} \tag{7}$$

$$F_1 = f_c + \Delta f = f_c + \frac{m}{2T_s} \tag{8}$$

Avec :
— f_c la fréquence de la porteuse, dite fréquence centrale,
— Δf la déviation de la modulation (correspondant au décalage entre la fréquence codant le symbole et la fréquence de la porteuse),
— m l'indice de modulation (correspondant à une valeur entre 0 et 1 caractérisant la modulation),

— T_s la durée d'un symbole (soit l'inverse du débit de données).

Une telle modulation produira donc un signal modulé dont l'amplitude est constante et la phase continue au cours du temps, comme l'illustre la figure 3.

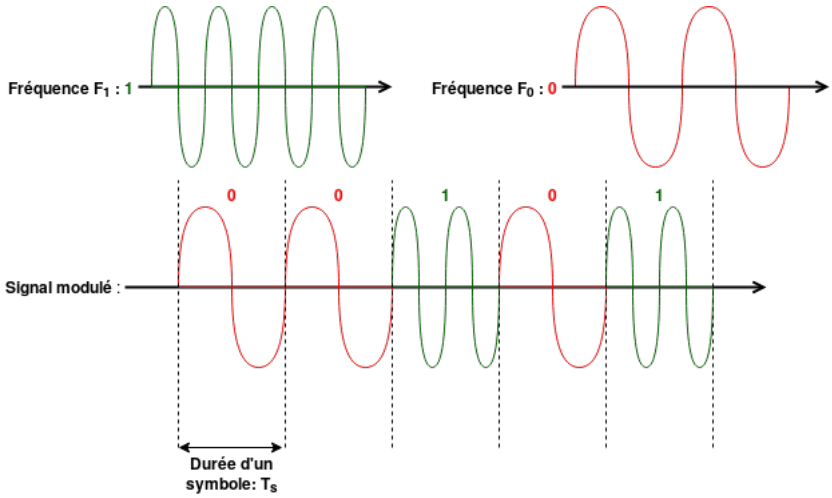

Fig. 3. Représentation temporelle du signal modulé d'une *GFSK*

Il n'est pas naturel de représenter une modulation en fréquence sous la forme d'une constellation (représentation I/Q dans le plan complexe), car la fréquence instantanée du signal modulant n'est pas directement observable sur celle ci (seules les informations de phase instantanée et d'amplitude apparaissent). Cependant, il est possible d'en donner une intuition en notant la relation suivante, reliant la phase instantanée à la fréquence instantanée :

$$f(t) = \frac{1}{2\pi}\frac{d\phi(t)}{dt} \tag{9}$$

Avec :

— $f(t)$ la fréquence instantanée,

— $\phi(t)$ la phase instantanée.

Ainsi, la variation de fréquence instantanée peut être observée en observant le sens de rotation de la phase instantanée : une augmentation de la fréquence (codant la valeur 1) provoquera une rotation de la phase dans le sens anti-horaire, tandis qu'une diminution de la fréquence (codant la valeur 0) provoquera une rotation dans le sens horaire. On peut ainsi

représenter une telle modulation dans le plan complexe en observant le sens de rotation de la phase, comme illustré par la figure 4.

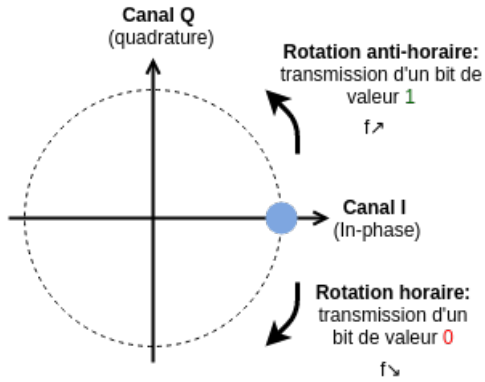

Fig. 4. Représentation I/Q d'une modulation *2-FSK*

Dans le cas du *BLE*, la spécification précise que l'indice de modulation doit être compris entre 0.45 et 0.55. Quant à la durée d'un symbole T_s, elle dépend du mode utilisé. En effet, les premières versions de la spécification imposaient un débit de données de 1 Mbits/s (soit $T_s = 10^{-6}s$). Cependant la version 5 de la spécification a introduit deux nouveaux modes de fonctionnement de la couche physique : un mode dit *LE Coded*, que nous ne développerons pas ici, et un mode dit *LE 2M* fonctionnant à 2 Mbit/s (soit $T_s = 5 \times 10^{-7}s$).

La fréquence centrale est, quant à elle, dépendante du canal de communication utilisé. En effet, la spécification prévoit 40 canaux de communication dans la bande de fréquence 2.4 à 2.5 GHz, occupant chacun une largeur de bande de 2 MHz. Trois de ces canaux (numérotés 37, 38 et 39) étaient initialement dédiés à la diffusion de messages d'annonce (canaux d'*advertising*) tandis que les 37 autres canaux étaient prévus pour l'échange de données en mode connecté (canaux de données). Cependant, l'ajout des nouveaux modes *LE Coded* et *LE 2M* introduit la possibilité d'utiliser les canaux de données comme canaux d'*advertising* secondaires. Chaque canal étant numéroté par un nombre entier compris entre 0 et 39 (inclus), la relation suivante relie la fréquence centrale du canal à son numéro :

$$f_c = \begin{cases} 2402, & \text{si } k = 37 \\ 2426, & \text{si } k = 38 \\ 2480, & \text{si } k = 39 \\ 2400 + 2(k+2), & \text{si } k < 11 \\ 2400 + 2(k+3), & \text{sinon.} \end{cases} \quad (10)$$

Avec :
— k le numéro de canal, compris entre 0 et 39,
— f_c la fréquence centrale du canal (en MHz).

3.3 Le Zigbee

Le protocole *Zigbee* est l'un des protocoles de communication sans fil les plus communément utilisés dans l'*IoT*. Sa faible consommation énergétique, le coût modique des composants radios ainsi que la possibilité de construire des topologies complexes le rendent particulièrement attractif pour le développement de ce type de systèmes, et en font un concurrent sérieux du *BLE*. Basé sur la norme IEEE 802.15.4 [1], qui définit notamment le fonctionnement des couches physique et liaison, sa spécification décrit principalement les couches supérieures de la pile protocolaire (notamment réseau et applicative). Nous nous concentrerons ici sur les couches inférieures du protocole, et particulièrement la couche physique telle qu'elle est définie par la norme 802.15.4.

La couche physique du protocole (nommée *PHY*) définit le format de paquet illustré en figure 5.

Celui-ci est composé des champs suivants :

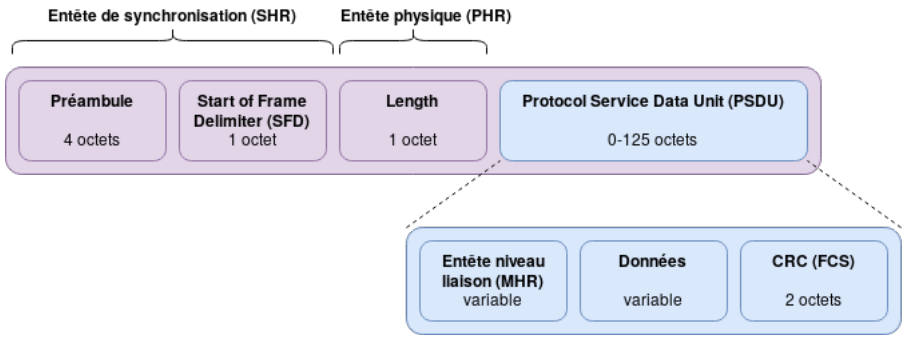

Fig. 5. Format d'une trame 802.15.4

— **Préambule :** champ de 4 octets correspondant à une série de 4 octets nuls (0x00 0x00 0x00 0x00), utilisé pour synchroniser le récepteur sur le début de la trame,
— **Start of Frame Delimiter (SFD) :** champ d'un octet correspondant à la valeur 0x7A, indiquant le début de la trame,
— **Length (PHR) :** champ d'un octet correspondant à la taille en octets du Protocol Service Data Unit,
— **Protocol Service Data Unit (PSDU) :** champ de taille variable, encapsulant la trame niveau liaison (ou *MAC*). Cette trame est composée d'une entête (le *MAC Header*, ou *MHR*), des données à encapsuler, transmises par les couches protocolaires supérieures, ainsi qu'un champ de deux octets, le Frame Check Sequence (ou *FCS*), permettant de vérifier l'intégrité de la trame reçue.

La trame ainsi générée n'est cependant pas fournie en entrée du modulateur sous cette forme : en effet, la norme 802.15.4 utilise une technique d'étalement de spectre dite *Direct Sequence Spread Spectrum* (ou *DSSS*). Chaque octet est découpé en deux blocs de 4 bits, les 4 bits de poids faible (*Least Significant Bits*, ou *LSB*) puis les 4 bits de poids forts (*Most Significant Bits*, ou *MSB*). A chacun de ces blocs est ensuite substitué une séquence pseudo-aléatoire de 32 bits nommée séquence *PN* (*Pseudorandom Noise*) selon la correspondance indiquée dans le tableau 1. Les bits de cette séquence sont aussi appelés *chips*.

Bloc ($b_0b_1b_2b_3$)	Séquence PN ($c_0c_1 \ldots c_{30}c_{31}$)
0000	11011001 11000011 01010010 00101110
1000	11101101 10011100 00110101 00100010
0100	00101110 11011001 11000011 01010010
1100	00100010 11101101 10011100 00110101
0010	01010010 00101110 11011001 11000011
1010	00110101 00100010 11101101 10011100
0110	11000011 01010010 00101110 11011001
1110	10011100 00110101 00100010 11101101
0001	10001100 10010110 00000111 01111011
1001	10111000 11001001 01100000 01110111
0101	01111011 10001100 10010110 00000111
1101	01110111 10111000 11001001 01100000
0011	00000111 01111011 10001100 10010110
1011	01100000 01110111 10111000 11001001
0111	10010110 00000111 01111011 10001100
1111	11001001 01100000 01110111 10111000

Tableau 1. Table de correspondance bloc / séquence PN

Les séquences PN sont ensuite fournies en entrée du modulateur. La couche physique de la norme 802.15.4 se base sur une modulation de phase nommée *Offset Quadrature Phase Shift Keying* (ou *O-QPSK*) dans la bande 2.4 à 2.5 GHz. Il s'agit d'une variante de la modulation de phase *Quadrature Phase Shift Keying*, dont le principe est de coder l'information binaire fournie en entrée en modulant la phase de la porteuse. Quatre valeurs de phase sont ainsi utilisées pour transmettre quatre symboles, chaque symbole étant composé de deux bits consécutifs, comme l'illustre la figure 6.[4]

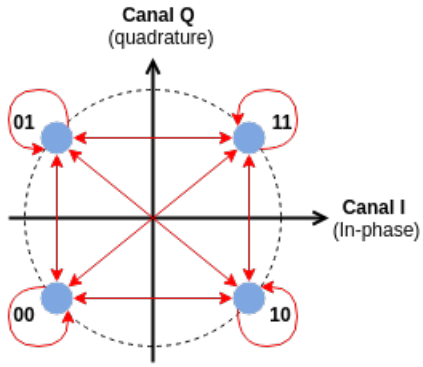

Fig. 6. Représentation I/Q d'une *QPSK*

Pour générer un tel signal, il est possible de contrôler indépendamment les composantes I (en phase) et Q (en quadrature) : la composante I sera utilisée pour moduler les bits pairs et la composante Q pour moduler les bits impairs. Pour cela, la première étape consiste à transformer le message binaire à moduler en une séquence d'impulsions rectangulaires (notée $m(t)$) de durée T_b (où T_b correspond à la moitié de la durée d'un symbole) : un bit à 1 sera codé par une impulsion rectangulaire positive tandis qu'un bit à 0 sera codé par une impulsion rectangulaire négative. On génère ensuite la séquence $I(t)$ à partir des bits pairs de $m(t)$ et $Q(t)$ à partir des bits impairs : chaque impulsion rectangulaire composant ces séquences aura ainsi une durée de $T_s = 2T_b$. Il est ensuite possible de générer le signal modulé $s(t)$ à partir des composantes en phase et en quadrature, tel que :

4. Dans le cas précis du *Zigbee* et donc de la modulation *O-QPSK*, chaque symbole est composé de 2 chips.

$$s(t) = I(t)\cos(2\pi f_c t) - Q(t)\sin(2\pi f_c t) \tag{11}$$

avec :

— $I(t)$ la séquence d'impulsions rectangulaires correspondant aux bits pairs de $m(t)$,

— $Q(t)$ la séquence d'impulsions rectangulaires correspondant aux bits impairs de $m(t)$,

— f_c la fréquence de la porteuse.

La figure 7 présente un exemple d'un signal modulé et des étapes nécessaires à sa construction.

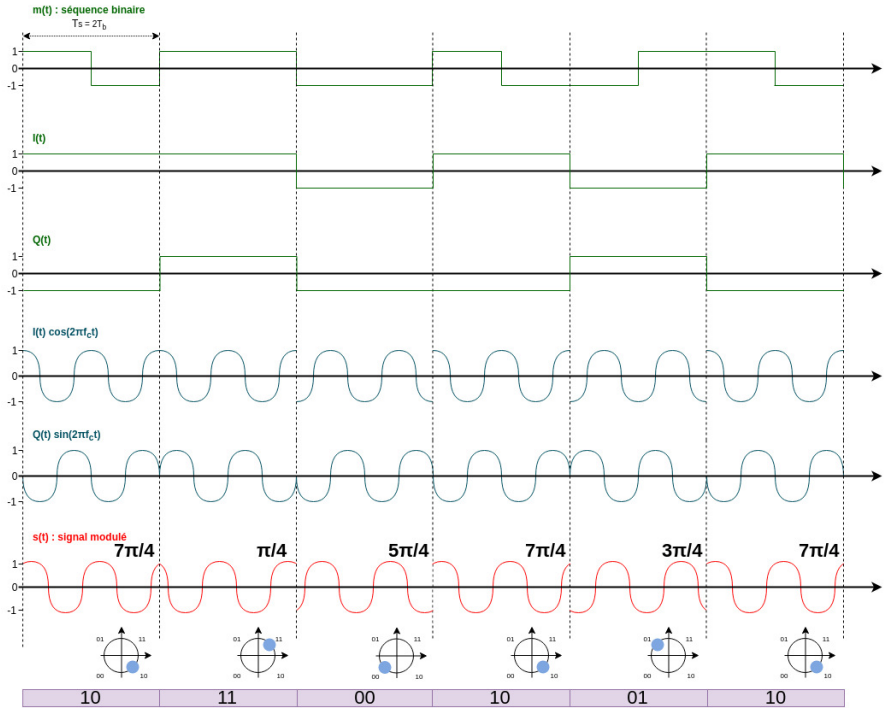

Fig. 7. Représentation temporelle d'un signal modulé en *QPSK* et des étapes intermédiaires nécessaires à sa construction

Cependant, cet exemple permet de mettre en évidence certains inconvénients liés à ce type de modulation. En effet, le diagramme de constellation présenté en figure 6 indique deux transitions diagonales, traversant les deux axes à l'origine. Ces transitions sont problématiques car elles peuvent être amenées à générer de fortes variations d'amplitude du signal lors des

sauts de phase de $\pm\pi$. De même, des sauts de phase brusques impliquent une bande passante plus importante.

L'*O-QPSK* résout en partie cette problématique : le principe est très similaire à la *QPSK*, mais la composante en quadrature est décalée temporellement de T_b par rapport à la composante en phase. Ainsi, cela a pour effet de supprimer les transitions de phase de $\pm\pi$ comme illustré dans le diagramme de constellation de la figure 8 : les composantes étant modifiées alternativement, les sauts de phase possibles sont limités à $\pm\frac{\pi}{2}$ (une transition de $\pm\pi$ de la *QPSK* correspondra à deux transitions successives de $\pm\frac{\pi}{2}$ avec l'*O-QPSK*).

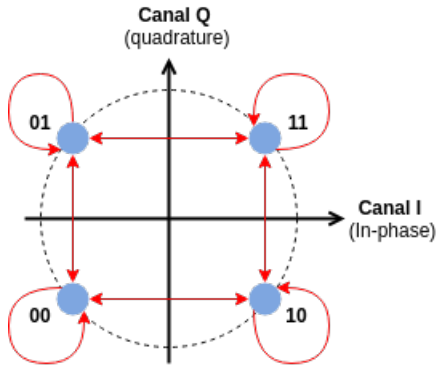

Fig. 8. Représentation I/Q d'une *O-QPSK*

La représentation temporelle des signaux générés par cette modulation (consultable en figure 9) illustre une dernière problématique : on observe un certain nombre de discontinuités de phase, qui ont pour effet d'entraîner l'utilisation d'une bande passante plus importante.

En conséquence, la norme 802.15.4 prévoit l'utilisation d'une impulsion de mise en forme semi-sinusoïdale pour la formation des séquences d'impulsions $I(t)$ et $Q(t)$ à partir du message à moduler : chaque chip à 1 est modulé par une semi-sinusoïde positive et chaque chip à 0 par une semi-sinusoïde négative, comme présenté en figure 10.

Cette modification a pour effet de remplacer les sauts de phases brusques des modulations précédentes par des sauts de phases continus, évoluant linéairement durant la période d'un chip T_b : la phase instantanée du signal modulé devient donc continue en fonction du temps et l'amplitude du signal reste quant à elle constante. Ainsi, à chaque instant d'échantillonage, il n'y a que deux transitions possibles vers l'état suivant :

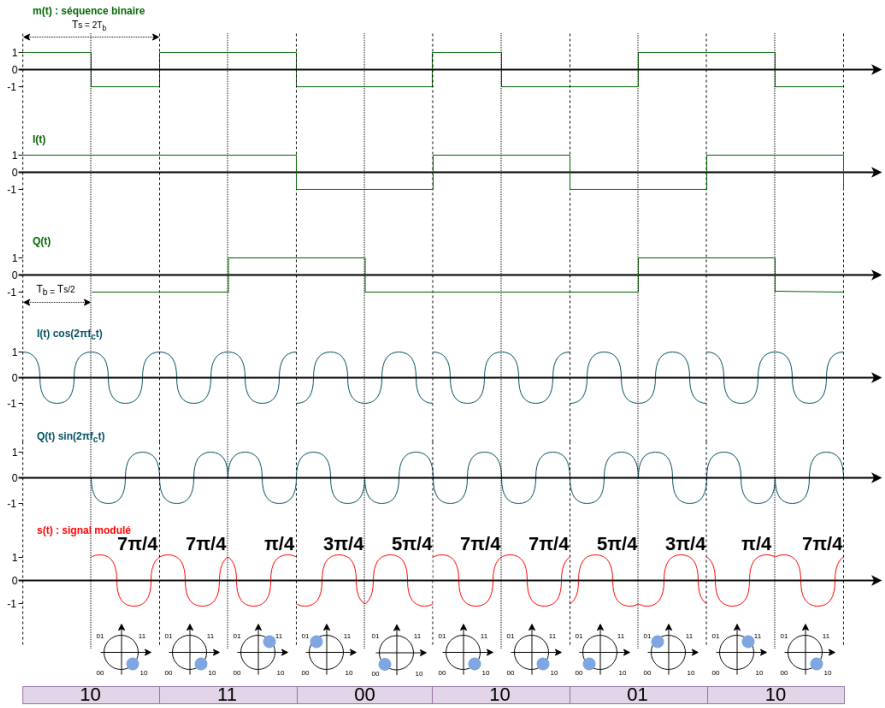

Fig. 9. Représentation temporelle d'un signal modulé en *O-QPSK* et des étapes intermédiaires nécessaires à sa construction

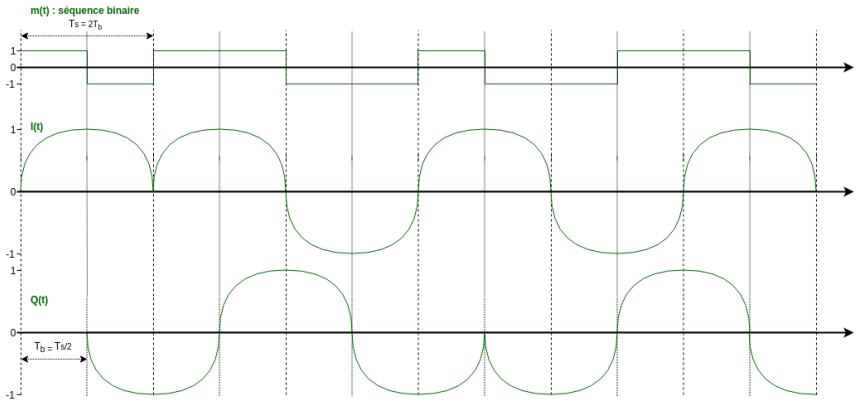

Fig. 10. Représentation temporelle des séquences d'impulsions semi-sinusoïdales

$+\frac{\pi}{2}$ et $-\frac{\pi}{2}$. La transition à emprunter dépend de la valeur du bit, si l'on est en train de moduler un bit pair ou impair et de l'état actuel : par exemple, si l'état de départ correspond au symbole 11 et qu'on désire moduler un

bit impair à 1, on empruntera la transition vers l'état 01, provoquant une augmentation linéaire de $+\frac{\pi}{2}$ de la phase instantanée durant la période T_b. Le diagramme de constellation ainsi qu'un exemple d'application de cette modulation sur le début d'une séquence PN sont présentés en figure 11.

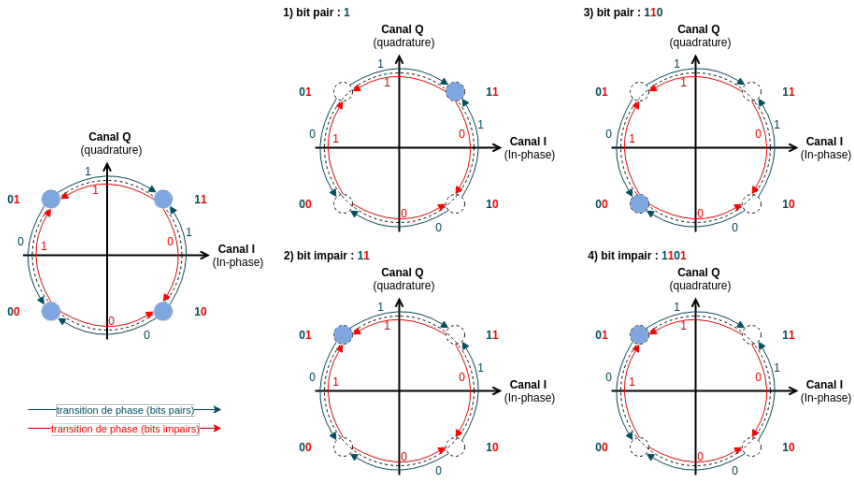

Fig. 11. Représentation I/Q de la modulation *O-QPSK* mise en forme par une impulsion semi-sinusoïdale

La spécification de la norme 802.15.4 indique un débit de données de 2 Mchips/s dans la bande 2.4 à 2.5 GHz, ce qui correspond donc à $T_b = 5 \times 10^{-7} s$. En conséquence, le débit correspondant aux bits formant la *PPDU* avant substitution par les séquences PN correspond à 250 kbits/s. La fréquence de la porteuse (aussi appelée fréquence centrale) est, quant à elle, dépendante du canal de communication utilisé. La norme 802.15.4 propose en effet l'utilisation de 16 canaux de communication, numérotés de 11 à 26 et occupant chacun une largeur de bande de 2 MHz. Deux canaux consécutifs sont espacés de 3 MHz. La relation suivante relie la fréquence centrale f_c au numéro du canal utilisé k :

$$f_c = 2405 + 5(k - 11) \tag{12}$$

Avec :
— k le numéro de canal, compris entre 11 et 26,
— f_c la fréquence centrale du canal (en MHz).

4 Présentation de l'attaque WazaBee

Dans cette section, nous allons présenter l'attaque *WazaBee*, dont l'objectif consiste à détourner le composant radio embarqué sur une puce *BLE* afin de l'utiliser pour émettre et recevoir des trames 802.15.4 (et notamment des trames *Zigbee*). Nous commencerons par décrire le principe de l'attaque et ses fondements théoriques, puis nous listerons les différentes contraintes liées au fonctionnement légitime de la puce et les solutions à mettre en oeuvre pour les contourner.

4.1 Situation initiale

Nous considérons que l'attaquant a pu préalablement compromettre une puce capable de communiquer en *BLE* et qu'il est en mesure d'exécuter sur celle-ci un code arbitraire. Cette compromission a pu être atteinte via diverses stratégies, telles que des mécanismes d'attaque réseaux (attaque d'un processus de mise à jour *Over The Air*), des vulnérabilités propres au système embarqué de l'objet et à son firmware (*Remote Code Execution* dûe à l'exploitation d'une vulnérabilité dans le code du firmware) ou des attaques physiques autorisant le flashage du composant. Cette compromission est considérée comme une condition préalable à l'attaque *WazaBee*, et ne sera pas développée dans cet article.

4.2 Principe de l'attaque

L'intuition à la base de cette stratégie d'attaque se base sur l'existence d'une relation étroite entre les modulations utilisées par ces deux protocoles, *GFSK* et *O-QPSK*. Les sous paragraphes suivants expliquent comment on peut établir le lien entre les différentes modulations impliquées.

De la GFSK à la MSK Comme vu précédemment, la modulation utilisée par le *BLE* est une *Gaussian Frequency Shift Keying* dont l'indice de modulation m est compris entre 0.45 et 0.55. Cette dernière caractéristique est assez intéressante, car elle nous permet d'assimiler la modulation utilisée par le *BLE* à un cas particulier de la *GFSK*, nommée *GMSK* (*Gaussian Minimum Shift Keying*) avec un indice de modulation $m = \frac{1}{2}$. Le signal modulé généré par une modulation de type *GMSK* a pour caractéristique de présenter une amplitude constante ainsi qu'une phase évoluant de façon continue au cours du temps. De plus, une modulation *GMSK* est une modulation *MSK* (*Minimum Shift Keying*) dont le signal

modulant est préalablement mis en forme par un filtre Gaussien. Si nous négligeons l'effet du filtre Gaussien, on peut assimiler la modulation du *BLE* à la modulation *MSK*, qui fait varier linéairement et continûment la phase de $-\frac{\pi}{2}$ lors de la modulation d'un bit à 0 et de $+\frac{\pi}{2}$ lors de la modulation d'un bit à 1. La figure 12 illustre ce comportement par un diagramme appelé treillis de phase, représentant toutes les variations de phase possibles en fonction du temps d'une *FSK* (à phase continue) ainsi que son application à la modulation *MSK*.

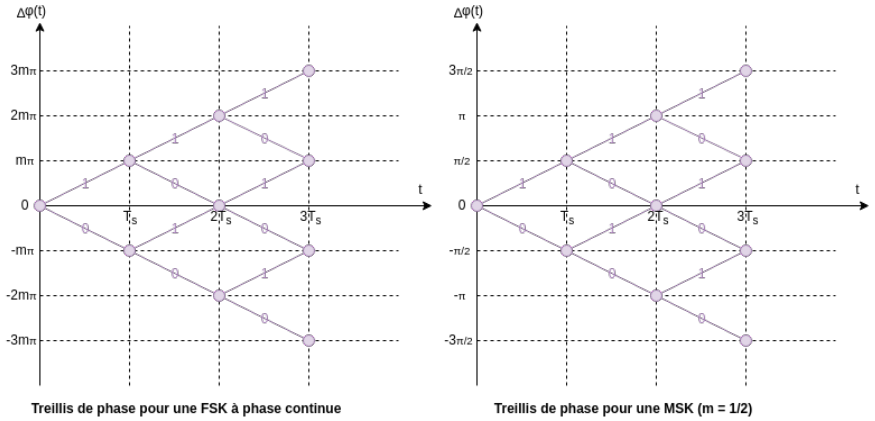

Fig. 12. Treillis de phase d'une *FSK* à phase continue et d'une *MSK*

De la MSK à l'O-QPSK Comme expliqué précédemment, une *O-QPSK* mise en forme par une impulsion semi-sinusoïdale partage avec la *MSK* la propriété de conserver une amplitude constante et une phase continue. De plus, chaque bit modulé a pour effet de générer une transition continue et linéaire de la phase de $\pm\frac{\pi}{2}$. Les deux modulations *MSK* et *O-QPSK* sont donc très proches. De façon plus formelle, les travaux de [19] montrent l'équivalence théorique entre *MSK* et *O-QPSK* mise en forme par une impulsion semi-sinusoïdale, si on choisit adéquatement un codage équivalent avec la même durée de symbole $T_{s(MSK)} = T_{b(OQPSK)}$.

Du BLE au Zigbee Si on néglige l'effet du filtre gaussien (qui aura pour conséquence de rendre les transitions de phase plus progressives), on peut donc faire l'hypothèse que la modulation du *BLE* peut être approximée par une *MSK* qui sera proche d'une modulation *O-QPSK* employé par les puces *Zigbee*.

En conclusion, on peut faire les hypothèses suivantes :
— Contrôler le message fourni en entrée d'un modulateur *GFSK* compatible avec la spécification du *BLE* devrait permettre de générer un signal modulé correspondant à une séquence binaire interprétable par un démodulateur *O-QPSK* (mis en forme par une impulsion semi-sinusoïdale) compatible avec la norme 802.15.4.
— Un message arbitraire modulé par un modulateur *O-QPSK* (mis en forme par une impulsion semi-sinusoïdale) compatible avec la norme 802.15.4 devrait générer un signal modulé correspondant à une séquence binaire interprétable par un démodulateur *GFSK* compatible avec la spécification du *BLE*.

4.3 Génération de la table de correspondance

La première problématique à résoudre consiste à établir une table de correspondance entre les séquences PN utilisées par la norme 802.15.4 (qui résultent donc d'une interprétation du signal par une modulation de phase, l'*O-QPSK* mise en forme par une impulsion semi-sinusoïdale) et leur interprétation par une modulation de fréquence de type *MSK*. A partir de cette table de correspondance, il sera alors possible de construire une séquence binaire à fournir en entrée d'un modulateur compatible avec la spécification *BLE* afin de générer un signal modulé proche de celui attendu par un démodulateur 802.15.4, mais également d'interpréter une trame 802.15.4 comme un signal modulé en fréquence et pouvant être démodulé par un démodulateur *BLE*.

Le principe de génération des séquences *MSK* équivalentes consiste à coder chaque transition de phase de l'*O-QPSK* par un 1 si elle correspond à une rotation du vecteur représentant le signal dans le plan complexe dans le sens anti-horaire (augmentation de la phase instantanée de $+\frac{\pi}{2}$) ou par un 0 si elle correspond à une rotation dans le sens horaire (diminution de la phase instantanée de $-\frac{\pi}{2}$). Si on applique ce principe de codage à l'exemple présenté en figure 11 correspondant à un début de séquence PN (1101), on obtient les étapes illustrées en figure 13.

Tout d'abord, le premier bit est un bit pair, qui correspond sur la constellation à l'état codant au symbole 11. Aucune transition de phase n'a encore été réalisée, la séquence équivalente en *MSK* est donc vide pour l'instant.

Le prochain bit de la séquence PN étant un bit impair, on peut suivre soit la transition de $+\frac{\pi}{2}$ menant à l'état 01 (correspondant à un 1), soit la transition de $-\frac{\pi}{2}$ menant à l'état 10. Dans notre cas, la valeur du bit étant 1, on va suivre la transition $+\frac{\pi}{2}$. Celle ci correspond à une rotation

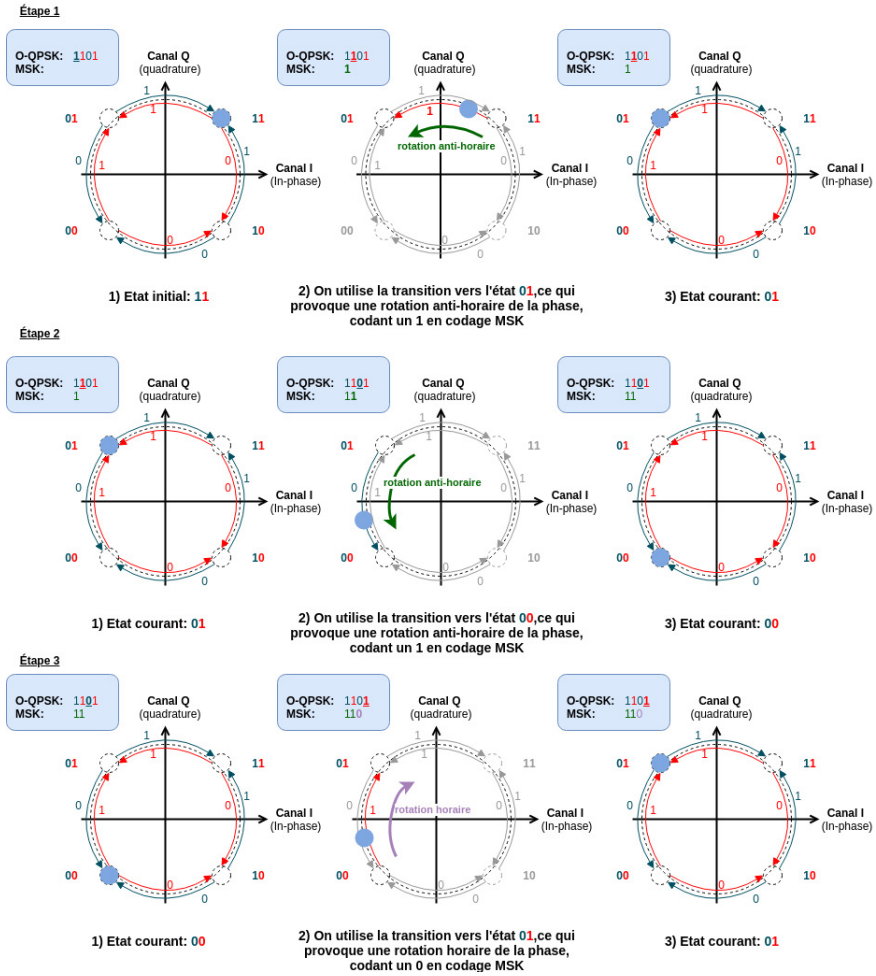

Fig. 13. Construction d'une séquence équivalente

de la phase dans le sens anti-horaire, qui correspond à un 1 dans le codage *MSK* équivalent. A ce stade, on a donc la séquence 11 en codage *O-QPSK* qui correspond à la séquence 1 en codage *MSK*.

Le prochain bit de la séquence PN à coder est un bit pair, correspondant à la valeur 0. On va donc suivre la transition de $+\frac{\pi}{2}$ menant à l'état 00 : cette transition correspondant à nouveau à une rotation de phase dans le sens anti-horaire, le codage *MSK* équivalent est un 1. A ce stade, la séquence 110 en codage *0-QPSK* a donc pour équivalence la séquence 11 en codage *MSK*.

Le prochain bit de la séquence PN étant un bit impair ayant pour valeur 1, la transition à suivre est la transition de $-\frac{\pi}{2}$ menant à l'état 01 : ceci a pour effet de provoquer une rotation de phase dans le sens horaire, correspondant donc à un 0 en codage *MSK*. On a donc la séquence *O-QPSK* 1101 et son équivalence en codage *MSK* valant 110.

Si on généralise ce principe de codage, on obtient l'algorithme 1. En appliquant cet algorithme aux 16 séquences PN existantes, on peut construire la table de correspondance 2. On peut également noter qu'une séquence codée en *O-QPSK* de longueur n aura pour équivalence en codage *MSK* une séquence de longueur $n-1$ étant donné que celle ci représente les transitions entre phases.

```
    Input : sequence_oqpsk[32]
    Output : sequence_msk[31]
 1: etats_pairs[4] ← {1, 0, 0, 1}
 2: etats_impairs[4] ← {1, 1, 0, 0}
 3: etat_courant ← 0
 4: i ← 1
 5: while i < 32 do
 6:     if i est impair then
 7:         if sequence_oqpsk[i] = etats_impairs[(etat_courant + 1)  mod 4] then
 8:             etat_courant ← (etat_courant + 1)  mod 4
 9:             sequence_msk[i − 1] ← 1
10:         else
11:             etat_courant ← (etat_courant − 1)  mod 4
12:             sequence_msk[i − 1] ← 0
13:     else
14:         if sequence_oqpsk[i] = etats_pairs[(etat_courant + 1)  mod 4] then
15:             etat_courant ← (etat_courant + 1)  mod 4
16:             sequence_msk[i − 1] ← 1
17:         else
18:             etat_courant ← (etat_courant − 1)  mod 4
19:             sequence_msk[i − 1] ← 0
20:     i ← i + 1
```

Algorithme 1. Algorithme de conversion d'une séquence PN

4.4 Contraintes

La mise en oeuvre pratique d'une telle attaque nécessite de résoudre un certain nombre de contraintes, notamment liées aux caractéristiques de la couche physique du *BLE* précédemment présentées. En pratique, notre objectif étant d'être en mesure d'implémenter des primitives de réception et d'émission de trame 802.15.4 sur une puce supportant la version 5.0 de la spécification du *BLE*, il est nécessaire d'être en mesure de contrôler les éléments suivants :

Bloc ($b_0b_1b_2b_3$)	Séquence PN - codage MSK ($m_0m_1 \ldots m_{29}m_{30}$)
0000	11000000111011110101110011011 00
1000	10011100000011101111010111001 10
0100	01011001110000001110111101011 10
1100	01001101100111000000111011110 10
0010	11011100110110011100000011101 11
1010	01110101110011011001110000001 11
0110	11101111010111001101100111000 00
1110	00001110111101011100110110011 10
0001	00111111000100001010001100100 11
1001	01100011111100010000101000110 01
0101	10100110001111110001000010100 01
1101	10110010011000111111000100001 01
0011	00100011001001100011111100010 00
1011	10001010001100100110001111110 00
0111	00010000101000110010011000111 11
1111	11110001000010100011001001100 01

Tableau 2. Table de correspondance des séquences PN

— **le débit de données :** l'attaque nécessite que la durée d'un symbole dans le codage *MSK* soit identique à la durée d'un bit dans le codage *O-QPSK*, soit $T_{s(MSK)} = T_{b(OQPSK)}$. On doit donc être en mesure de configurer le modulateur et le démodulateur utilisé par la puce pour utiliser un débit de 2 Mbits/s, identique au débit de 2Mchips/s indiqué par la norme 802.15.4.

— **la fréquence centrale :** l'attaque nécessite que la fréquence centrale du canal *BLE* utilisé corresponde à celle d'un canal *Zigbee*.

— **les données en entrée du modulateur :** afin d'implémenter une primitive d'émission, il est nécessaire de pouvoir contrôler (directement ou indirectement) les données transmises au modulateur de la puce, afin de pouvoir fournir les séquences PN codées en *MSK*.

— **les données en sortie du démodulateur :** afin d'implémenter une primitive de réception, il est nécessaire de pouvoir détecter la réception d'une trame 802.15.4 et de pouvoir récupérer (directement ou indirectement) les données en sortie du démodulateur de la puce.

Contrôler le débit de données est relativement trivial depuis l'introduction d'un nouveau mode *LE 2M* pour la couche physique du *BLE*. Cette modification a été introduite dans la version 5.0 de la spécification, et autorise notamment l'usage d'un débit de données de 2Mbits/s, ce qui correspond parfaitement à nos besoins. Il devrait donc être possible de

résoudre cette première contrainte sur toute puce implémentant la version 5.0 de la spécification *Bluetooth*.

La seconde contrainte concerne le contrôle de la fréquence centrale : il est en effet nécessaire de pouvoir sélectionner la fréquence en fonction du canal *Zigbee* visé par l'attaque. Plusieurs solutions peuvent être mises en oeuvre pour résoudre cette problématique selon la permissivité de la puce et de l'API fournie. En effet, la plupart des puces étudiées supportant le *BLE* dans sa version 5.0 permettent de choisir arbitrairement une fréquence dans la bande 2.4 à 2.5GHz, dans ce cas il est donc possible de fournir directement la fréquence centrale du canal *Zigbee* visé. Si la puce n'autorise pas une telle fonctionnalité, il est alors possible de sélectionner un canal *BLE* dont la fréquence centrale correspond à un canal *Zigbee* : on n'aura alors accès qu'à un sous ensemble des canaux *Zigbee* existants, ceux correspondant à un canal défini par la spécification Bluetooth. Les canaux concernés sont indiqués dans le tableau 3. Ce détournement des canaux *BLE* est rendu possible car les canaux *Zigbee* comme *BLE* ont les même caractéristiques (2MHz de largeur de bande) et car le mode *LE 2M* autorise l'usage des canaux de données comme canaux d'*advertising* secondaires, autorisant ainsi une émission ou réception «directe» sur le canal grâce au mode *advertising* (le mode connecté implémente en effet un algorithme de *channel hopping* qui complique énormément cette stratégie d'attaque).

Canaux *Zigbee*	Canaux *BLE*	fréquence centrale (f_c)
12	3	2410 MHz
14	8	2420 MHz
16	12	2430 MHz
18	17	2440 MHz
20	22	2450 MHz
22	27	2460 MHz
24	32	2470 MHz
26	39	2480 MHz

Tableau 3. Table de correspondance entre les canaux *Zigbee* et les canaux *BLE*

La troisième contrainte est de pouvoir contrôler les données qui vont être fournies en entrée du modulateur de la puce : il est en effet nécessaire de pouvoir fournir une succession arbitraire de séquences PN (codées en *MSK*) pour pouvoir construire une primitive d'émission. La principale difficulté ici est liée au *whitening*, qui va appliquer un algorithme de transformation sur les données à transmettre, modifiant ainsi la trame

avant sa modulation. Certaines puces permettent de désactiver cette fonctionnalité, autorisant ainsi un contrôle direct sur les bits transmis au modulateur. Cependant, même en l'absence d'une telle possibilité, l'algorithme de *whitening* est réversible car basé sur un simple registre à décalage à rétroaction linéaire : il est donc possible de construire une séquence de bits qui, une fois la transformation appliquée, corresponde aux séquences PN, en appliquant au préalable l'algorithme de *dewhitening* sur les séquences à transmettre.

Dans ces deux cas, la suite de séquences PN à transmettre pour générer la trame 802.15.4 attendue pourra être encapsulée dans la charge utile d'un paquet *BLE*, par exemple dans les données d'*advertising* (le mode *LE 2M* autorisant la transmission de paquets d'*advertising* de grande taille, dont la charge utile peut faire jusqu'à 255 octets).

La quatrième contrainte est déterminante pour construire une primitive de réception : il faut être en mesure de détecter une trame 802.15.4, mais également de pouvoir la récupérer et la décoder pour retrouver les symboles correspondant aux séquences PN. Il est possible de détecter une telle trame en configurant l'*Access Address* de la puce *BLE* : en effet, celle-ci sert généralement de motif pour détecter une trame *BLE* légitime. Dès lors, il est possible de fournir comme *Access Address* la séquence PN (codée en *MSK*) correspondant au symbole 0000, afin de détecter le préambule d'une trame 802.15.4 (pour rappel, celui ci est composé de 4 octets nuls, soit 8 symboles 0000). Il sera alors nécessaire de désactiver la vérification d'intégrité, car les trames 802.15.4 ne seront pas des trames *BLE* valides (la puce doit impérativement autoriser cette désactivation pour qu'une primitive de réception puisse être implémentée) et de configurer la taille de trame à la taille maximale disponible. A ce stade, la problématique du *dewhitening* se pose exactement de la même manière qu'à la contrainte précédente : il doit être dans l'idéal désactivé, et si ce n'est pas possible il faudra appliquer un algorithme de *whitening* correctement configuré sur la trame pour récupérer les vrais bits produits en sortie du démodulateur. La conversion vers les symboles *Zigbee* d'origine peut être réalisée très simplement en utilisant *la distance de Hamming*. Ainsi, pour tout paquet reçu, on peut séparer ce paquet en blocs de 31 bits et pour chaque bloc, on utilise la distance de Hamming afin de retrouver à quelle séquence PN codée en *MSK* (elles sont indiquées dans la table 2) correspond le mieux le bloc reçu. L'utilisation de la distance de Hamming permet ici de contourner deux difficultés : des erreurs de bits provoquées par l'approximation que nous avons présenté précédemment, mais également des interférences dûes au canal, pouvant générer des *bitflips* durant la transmission.

On peut noter que l'équivalence de la modulation *O-QPSK* mise en forme par une impulsion semi-sinusoïdale et de la modulation *MSK* devrait en théorie permettre une attaque pivot «symétrique» de détournement de puces *Zigbee* pour attaquer le protocole *BLE*. Cependant, cette stratégie semble particulièrement difficile à mettre en oeuvre, car la pile protocolaire *Zigbee* nous empêche de contrôler finement les données en entrée du modulateur 802.15.4 ou en sortie du démodulateur, principalement en raison de la fonctionnalité de *Direct Sequence Spread Spectrum*, qui effectue l'opération de transformation des symboles aux sequences de chips. Il serait nécessaire de pouvoir contrôler directement l'entrée du modulateur et récupérer la sortie du démodulateur, ce qui ne nous semble pas facilement réalisable en l'état actuel des équipements déployés.

La figure 14 présente une vue d'ensemble du fonctionnement des primitives de réception et d'émission, ainsi que les éléments de configuration nécessaires à l'implémentation de la stratégie d'attaque.

5 Expérimentations

L'attaque *WazaBee* a été implémentée en tant que preuve de concept sur deux puces de constructeurs différents, le *nRF52832* de *Nordic Semi-Conductors* et le *CC1352-R1* de *Texas Instruments*. Dans cette section, nous nous proposons de décrire succinctement les implémentations réalisées, puis de présenter les expériences menées pour évaluer notre stratégie d'attaque.

5.1 Implémentation sur le nRF52832

La première implémentation de l'attaque a été réalisée sur le chip *nRF52832*. Il nous a semblé intéressant de tester l'implémentation sur cette puce en particulier car elle offre une grande souplesse dans la configuration du composant radio associé (principalement en raison de la vulnérabilité précédemment évoquée liée au registre de sélection d'adresses) et le support du *BLE* 5.0 (et notamment la couche *LE 2M PHY*). Son *API* radio est assez proche de celle du *nRF51*, bien connu de la communauté sécurité pour avoir été massivement détourné aux cours des dernières années en vue de développer des outils offensifs dédiés au BLE et au Enhanced ShockBurst (*BTLEJack*, *radiobit*, ...). Le prototype a été réalisé sur une carte de développement proposée par *AdaFruit* embarquant cette puce, l'*Adafruit Feather nRF52 Bluefruit LE*.

Nous allons ici présenter les éléments principaux permettant la mise en place de l'attaque. La première étape pour implémenter *WazaBee* sur cette

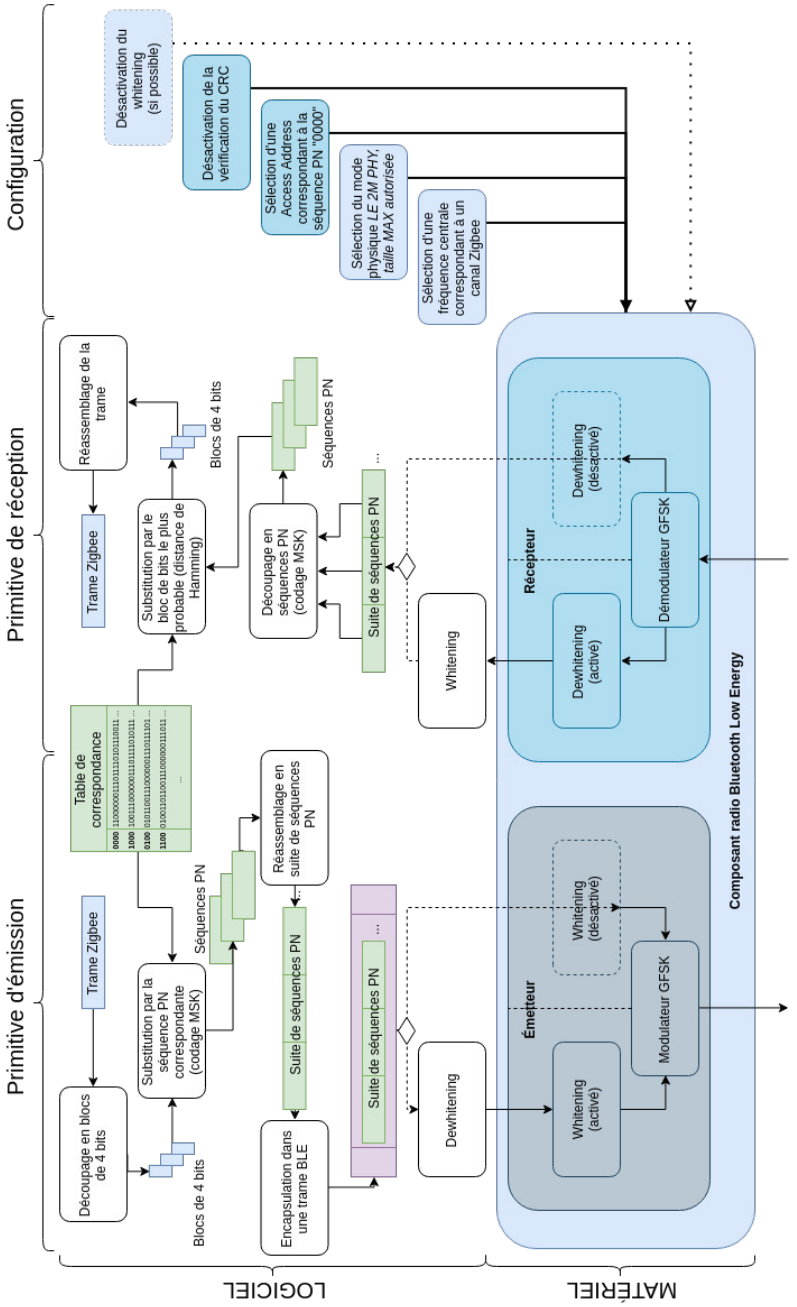

Fig. 14. Vue d'ensemble de la stratégie d'attaque *WazaBee*

puce consiste à écrire une fonction d'initialisation, chargée de configurer le composant radio. Le premier registre pertinent à examiner est le registre *FREQUENCY*. Celui-ci attend une valeur entière en MHz à laquelle 2400 MHz sera additionné pour sélectionner la bonne fréquence. Dans cette optique, nous avons développé la fonction suivante, qui ré-implémente le comportement défini par la relation (12) :

```
int channel_to_frequency(int channel) {
    return 5+5*(channel-11);
}
```

La sélection de la fréquence est ainsi réalisée par l'intermédiaire d'un appel à cette fonction, avec en paramètre le numéro de canal visé :

```
NRF_RADIO->FREQUENCY = channel_to_frequency(this->channel);
```

Le contrôle du débit de donnée est réalisé par l'intermédiaire du registre *MODE*, qui permet ainsi de spécifier l'utilisation du mode *LE 2M PHY* :

```
NRF_RADIO->MODE = (RADIO_MODE_MODE_Ble_2Mbit <<RADIO_MODE_MODE_Pos);
```

La sélection du motif de détection est une étape particulièrement importante : afin de maximiser le nombre de trames détectées, il est pertinent d'utiliser la séquence PN codée en *MSK* correspondant au symbole 0000, répétée 8 fois durant le préambule d'une trame 802.15.4. La puce pouvant être paramétrée en Big Endian, la table suivante a été générée à partir des séquences PN codées en *MSK* (chacune ayant été préfixée d'un bit à 0 pour pouvoir facilement être représentée sur 4 octets) :

```
static uint8_t SYMBOL_TO_CHIP_MAPPING[16][4] = {
    {0x60,0x77,0xae,0x6c},   // 11000000111011111010111001101100 (0)
    {0x4e,0x07,0x7a,0xe6},   // 10011100000011101111010111001110 (1)
    {0x2c,0xe0,0x77,0xae},   // 01011001110000001110111101011110 (2)
    {0x26,0xce,0x07,0x7a},   // 01001101100111000000111011111010 (3)
    {0x6e,0x6c,0xe0,0x77},   // 11011100110110011100000011110111 (4)
    {0x3a,0xe6,0xce,0x07},   // 01110101110011011001110000000111 (5)
    {0x77,0xae,0x6c,0xe0},   // 11101111010111001101110011100000 (6)
    {0x07,0x7a,0xe6,0xce},   // 00001110111101011100110110011110 (7)
    {0x1f,0x88,0x51,0x93},   // 00111111000100001010001100100011 (8)
    {0x31,0xf8,0x85,0x19},   // 01100011111100010000101000110001 (9)
    {0x53,0x1f,0x88,0x51},   // 10100110001111110001000010100001 (10)
    {0x59,0x31,0xf8,0x85},   // 10110010011000111111000100000101 (11)
    {0x11,0x93,0x1f,0x88},   // 00100011001001100011111110001000 (12)
    {0x45,0x19,0x31,0xf8},   // 10001010001100100110001111111000 (13)
    {0x08,0x51,0x93,0x1f},   // 00010000101000110010011000111111 (14)
    {0x78,0x85,0x19,0x31},   // 11110001000010100011001001100001 (15)
};
```

La sélection du motif de détection est réalisée par l'intermédiaire des registres *BASE0* et *PREFIX0* :

```
NRF_RADIO->BASE0=bytewise_bit_swap(
    ((uint32_t)SYMBOL_TO_CHIP_MAPPING[0][2])<<24 |
    ((uint32_t)0x00) |
    ((uint32_t)SYMBOL_TO_CHIP_MAPPING[0][1])<<16  |
    ((uint32_t)SYMBOL_TO_CHIP_MAPPING[0][0])<<8
);
NRF_RADIO->PREFIX0=bytewise_bit_swap(SYMBOL_TO_CHIP_MAPPING[0][3]);
```

Les registres *PCNF0* et *PCNF1* vont permettre une configuration globale du composant radio, permettant notamment de désactiver le *whitening*, configurer un motif de 4 octets (3 octets d'adresse + 1 octet de préambule) et configurer la récupération de blocs bruts en sortie du démodulateur de taille maximale (soit 255 octets) :

```
// Raw output (without length field)
NRF_RADIO->PCNF0 = 0x00000000;
NRF_RADIO->PCNF1 = (
    // Whitening off
    (RADIO_PCNF1_WHITEEN_Disabled << RADIO_PCNF1_WHITEEN_Pos) |
    // Big endian
    (RADIO_PCNF1_ENDIAN_Big << RADIO_PCNF1_ENDIAN_Pos)  |
    // Detection pattern size
    (3 << RADIO_PCNF1_BALEN_Pos) |
    // Data length
    (255 << RADIO_PCNF1_STATLEN_Pos) |
    // Max data length
    (255 << RADIO_PCNF1_MAXLEN_Pos)
);
```

La vérification d'intégrité, quant à elle, est désactivée par l'intermédiaire du registre *CRCCNF* :

```
NRF_RADIO->CRCCNF = 0x0;
```

La réception et le décodage des trames 802.15.4 sont implémentées dans l'interruption *RADIO_IRQHandler* correspondant au composant radio, et appliquent un algorithme basé sur la distance de Hamming utilisant la table précédemment décrite afin de retrouver les symboles initialement transmis. De même, la primitive d'émission (correspondant à la méthode *send*) construit la séquence à transmettre en se basant sur la table de correspondance, puis transmet les données au modulateur.

5.2 Implémentation sur le CC1352-R1

La seconde implémentation a été réalisée sur la puce *CC1352-R1* de *Texas Instruments* : le principal intérêt était de pouvoir tester l'approche

sur une puce moins réputée pour sa permissivité que le *nRF52*. La puce supporte nativement plusieurs protocoles, dont le *BLE* et le 802.15.4, cependant seule l'API du Bluetooth a été utilisée pour l'implémentation. Celle ci étant commune à plusieurs puces de *Texas Instruments*, l'implémentation de l'attaque devrait être similaire sur d'autres systèmes produits par *TI*.

La configuration du composant radio peut être réalisée par l'intermédiaire de multiples commandes. Ainsi, la commande nommée *CMD_BLE5_RADIO_SETUP* permet d'initialiser la communication avec le composant radio. Celle-ci est laissée à sa configuration par défaut, excepté la propriété *pRegOverrideCommon*, qui va permettre d'écraser la configuration par défaut de certains registres du composant radio : elle est ici utilisée pour autoriser la réception de trame jusqu'à 255 octets, limitée à 37 par défaut. Cette modification a été, entre autres, utilisée par Sultan Qasim Khan pour le développement d'un *sniffer Bluetooth* nommé *Sniffle* [20].

```
uint32_t pOverridesCommon[] =
{
    HW_REG_OVERRIDE(0x5328,0x0000),
    // Increases max RX packet length from 37 to 255
    // Sets one byte firmware parameter at offset 0xA5 to 0xFF
    (uint32_t)0x00FF8A53,
    (uint32_t)0xFFFFFFFF
};
// CMD_BLE5_RADIO_SETUP
rfc_CMD_BLE5_RADIO_SETUP_t RF_cmdBle5RadioSetup =
{
    .commandNo = 0x1820,
    .status = 0x0000,
    // ...
    .pRegOverrideCommon = pOverridesCommon,
    .pRegOverride1Mbps = 0,
    .pRegOverride2Mbps = 0,
    .pRegOverrideCoded = 0,
};
```

La configuration de la réception est réalisée par l'intermédiaire de la commande *CMD_BLE5_GENERIC_RX*. La sélection de la fréquence est réalisée par l'intermédiaire de la propriété *channel*, qui autorise la sélection d'une fréquence arbitraire (par exemple, la valeur 0x69 correspond à 2405 MHz, soit le canal *Zigbee* 11). La couche *LE 2M PHY* est activée en fixant à 0x1 la valeur de la propriété *phyMode.mainMode*. Le *whitening* peut être désactivé en fournissant une valeur d'initialisation à 0 dans la propriété *whitening.init*. On autorise la remontée des paquets au CRC invalide en indiquant 0 dans la propriété *rxConfig.bAutoFlushCrcErr*. Enfin, la

puce étant configurée en *Little Endian*, on fournit le motif de détection 0x9b3af703 (correspondant à la séquence PN codée en *MSK* du symbole 0000, réécrit en *Little Endian*) dans la propriété *pParams→accessAddress*, correspondant à l'Access Address. La table de correspondance symbole / séquence PN utilisée est similaire à celle implémentée pour le *nRF52*, mais les séquences ont été réécrites au format *Little Endian*. L'algorithme de décodage est similaire à celui implémenté sur le *nRF52*.

```
RF_cmdBle5GenericRx.channel = 0x69;
RF_cmdBle5GenericRx.whitening.init = 0;
RF_cmdBle5GenericRx.phyMode.mainMode = 1;
RF_cmdBle5GenericRx.pParams->accessAddress = 0x9b3af703;
// ...
RF_cmdBle5GenericRx.pParams->rxConfig.bAutoFlushCrcErr = 0;
// ...
RF_runCmd(rfBleHandle, (RF_Op*)&RF_cmdBle5GenericRx,
    RF_PriorityNormal,
    &rxCallback, RF_EventRxEntryDone);
```

L'émission, quant à elle, est réalisée par l'intermédiaire de la commande *CMD_BLE5_ADV_AUX*, et construit un paquet d'*advertisement* étendu à partir de la succession de séquences PN construite avant de le transmettre au modulateur. Sa configuration est comparable à celle de la commande précédente *CMD_BLE5_GENERIC_RX* :

```
RF_cmdBle5AdvAux.channel = 0x69;
RF_cmdBle5AdvAux.whitening.init = 0;
RF_cmdBle5AdvAux.phyMode.mainMode = 1;
```

5.3 Évaluations

Deux expériences ont été réalisées afin d'évaluer les primitives de réception et d'émission précédemment décrites. La première expérience, consacrée à la réception, consistait à transmettre 100 trames 802.15.4 dont la charge utile intégrait un compteur (incrémenté à chaque trame) à l'aide d'un émetteur *Zigbee* (l'AVR RZUSBStick d'Atmel). La carte de développement implémentant l'attaque *WazaBee*, espacée de l'émetteur par une distance de 3 mètres, recevait et décodait les trames correspondantes, puis calculait la FCS correspondant à la trame reçue pour évaluer son intégrité. Pour chaque canal *Zigbee*, les trames ont été classées en trois catégories : non reçue, reçue avec corruption d'intégrité, reçue sans corruption d'intégrité. Les résultats correspondants sont représentés sous forme de graphiques sur la figure 15.

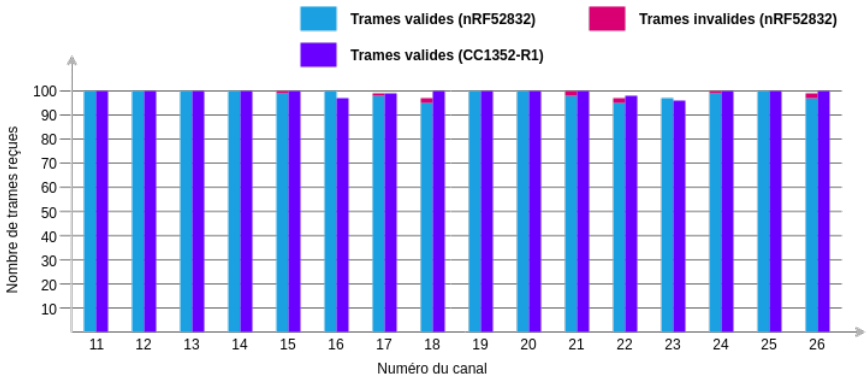

Fig. 15. Résultats de l'évaluation de la primitive de réception

On constate que la primitive de réception de *WazaBee* présente un taux de réception très satisfaisant pour les deux implémentations sur l'ensemble des canaux, avec une moyenne de 98.625% des trames reçues sans corruption d'intégrité pour le *nRF52832* et 99.375% pour le *CC1352-R1*. On constate dans les deux cas une légère diminution du taux de réception pour les canaux 17, 18, 21, 22 et 23, phénomène explicable par les interférences liées aux canaux 6 et 11 du WiFi, utilisés dans nos conditions expérimentales. On observe également que le *CC1352-R1* semble présenter une réception plus stable que son homologue, avec aucune corruption d'intégrité sur les trames reçues contre 0.6875 % pour le *nRF52832*.

La primitive d'émission a été évaluée dans des conditions similaires : la carte de développement implémentant *WazaBee* a été configurée pour émettre 100 trames intégrant un compteur, et un récepteur 802.15.4 (le RZUSBStick) a été placé à une distance de 3 mètres en mode réception. Chaque trame émise a ainsi pu être classée en trois catégories : non reçue, reçue avec corruption d'intégrité et reçue sans corruption d'intégrité. L'expérience a été réalisée sur l'ensemble des canaux *Zigbee*, et les résultats correspondants sont représentés sous forme de graphiques sur la figure 16.

On constate ainsi que dans les deux cas et pour l'ensemble des canaux, le taux de trames émises qui ont été reçues sans corruption d'intégrité par le RZUSBStick est très satisfaisant, avec une moyenne de trames reçues valides de 97.5 % pour les trames émises par le *nRF52832* et de 99.438 % pour le *CC1352-R1*. On observe un phénomène similaire à celui précédemment observé pour l'évaluation de la primitive de réception pour les canaux 17 et 18, probablement lié à l'utilisation du canal WiFi numéro

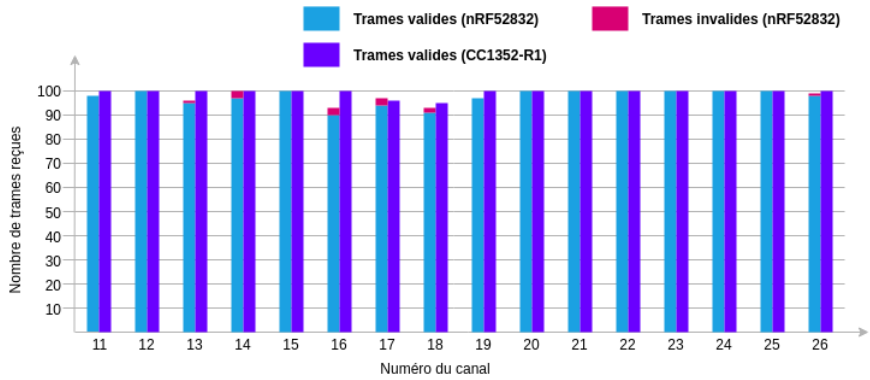

Fig. 16. Résultats de l'évaluation de la primitive d'émission

6. Le taux de trames reçues avec corruption d'intégrité est également légèrement plus élevé pour le *nRF82832* (avec une moyenne de 0.8125 % contre 0 % pour le *CC1352-R1*).

6 Conclusion et perspectives

Dans cet article, nous avons mis en évidence une nouvelle stratégie d'attaque pivot, nommée *WazaBee*, permettant de détourner le fonctionnement légitime d'une puce destinée à communiquer par l'intermédiaire du *BLE* afin d'émettre et recevoir des trames 802.15.4. Nous avons montré la généricité de cette attaque en l'implémentant sur deux puces présentant des architectures différentes, ainsi que sa stabilité et sa fiabilité, lors des deux évaluations réalisées. La conséquence directe de cette attaque est une augmentation considérable de la surface d'attaque, chaque système communiquant par l'intermédiaire d'un protocole basé sur la norme 802.15.4 (*Zigbee*, *6LoWPan*, ...) étant ainsi potentiellement atteignable depuis un composant supportant le *BLE*, technologie particulièrement répandue.

L'impact de cette attaque sur la sécurité des systèmes d'information nous semble particulièrement critique étant donné l'expansion rapide des objets connectés et le développement de multiples protocoles de communication sans fil. La coexistence de ces protocoles dans les mêmes environnements, ainsi que certaines caractéristiques des objets connectés (comme la mobilité) constituent des facteurs aggravants et amènent à s'interroger sur l'utilisation de ce type de stratégies offensives dans des scénarios d'attaque ciblés. On peut ainsi envisager l'utilisation de ce type d'attaques dans des stratégies offensives où l'attaquant tente de pivoter d'un système compromis à un autre plus difficilement accessible, mais

également dans des attaques de type canal caché, où l'attaquant pourrait exploiter ces primitives d'émission et de réception pour exfiltrer des données sensibles par l'intermédiaire d'un protocole sans fil non surveillé.

L'étude de ce type de stratégies offensives et le développement de contre-mesures adaptées nous semble être une perspective très importante pour la sécurité des environnements *IoT*. Ainsi, nous envisageons d'orienter nos futurs travaux sur ce type de problématique, en généralisant et formalisant ce type de stratégies d'attaque à d'autres protocoles de communication sans fil. Dans une perspective plus défensive, l'existence de ce type de stratégies offensives nous paraît plaider en faveur de solutions défensives capables de surveiller en temps réel de multiples protocoles, y compris s'ils ne sont pas déployés dans l'environnement légitime. En conséquence, nous envisageons d'étudier la conception et l'implémentation d'un système de détection et de prévention d'intrusion multi-protocolaire et multi-couches.

Références

1. Ieee standard for low-rate wireless networks. *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pages 1–709, April 2016.
2. B-L475E-IOT01A Data Brief. `https://www.st.com/resource/en/data_brief/b-l475e-iot01a.pdf`, 2018.
3. CC2652R Data Sheet. `http://www.ti.com/lit/ds/symlink/cc2652r.pdf`, 2019.
4. Bluetooth SIG. *Bluetooth Core Specification*, 12 2019.
5. Damien Cauquil. Radiobit, a BBC Micro :Bit RF firmware. `https://github.com/virtualabs/radiobit`, 2017.
6. Damien Cauquil. Sniffing btle with the micro :bit. *PoC or GTFO*, 17 :13–20, 2017.
7. Damien Cauquil. Weaponizing the BBC Micro :Bit. `https://media.defcon.org/DEFCON25/DEFCON25presentations/DEFCON25-Damien-Cauquil-Weaponizing-the-BBC-MicroBit-UPDATED.pdf`, 2017.
8. Romain Cayre, Jonathan Roux, Eric Alata, Vincent Nicomette, and Guillaume Auriol. Mirage : un framework offensif pour l'audit du Bluetooth Low Energy. In *Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 2019)*, pages 229–258, Rennes, France, June 2019.
9. Kameswari Chebrolu and Ashutosh Dhekne. Esense : Communication through energy sensing. In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking*, MobiCom '09, page 85–96, New York, NY, USA, 2009. Association for Computing Machinery.
10. Behrang Fouladi and Sahand Ghanoun. Security evaluation of the z-wave wireless protocol. 2013.
11. Travis Goodspeed. Promiscuity is the nrf24l01+'s duty. `http://travisgoodspeed.blogspot.com/2011/02/promiscuity-is-nrf24l01s-duty.html`, 2011.
12. Travis Goodspeed, Sergey Bratus, Ricky Melgares, Rebecca Shapiro, and Ryan Speers. Packets in packets : Orson welles' in-band signaling attacks for modern radios. pages 7–7, 08 2011.

13. Wenchao Jiang, Song Min Kim, Zhijun Li, and Tian He. Achieving receiver-side cross-technology communication with cross-decoding. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, page 639–652, New York, NY, USA, 2018. Association for Computing Machinery.
14. Wenchao Jiang, Zhimeng Yin, Ruofeng Liu, Zhijun Li, Song Min Kim, and Tian He. Bluebee : A 10,000x faster cross-technology communication via phy emulation. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, SenSys '17, New York, NY, USA, 2017. Association for Computing Machinery.
15. Song Min Kim and Tian He. Freebee : Cross-technology communication via free side-channel. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, MobiCom '15, page 317–330, New York, NY, USA, 2015. Association for Computing Machinery.
16. Zhijun Li and Tian He. Webee : Physical-layer cross-technology communication via emulation. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, MobiCom '17, page 2–14, New York, NY, USA, 2017. Association for Computing Machinery.
17. Michael C. Millian and Vibhu Yadav. Packet-in-packet exploits on 802.15.4. 2015.
18. Marc Newlin. MouseJack : White Paper. `https://github.com/BastilleResearch/mousejack/blob/master/doc/pdf/DEFCON-24-Marc-Newlin-MouseJack-Injecting-Keystrokes-Into-Wireless-Mice.whitepaper.pdf`, 2016.
19. S. Pasupathy. Minimum shift keying : A spectrally efficient modulation. *IEEE Communications Magazine*, 17(4) :14–22, July 1979.
20. Sultan Qasim Khan. Sniffle : A sniffer for Bluetooth 5 (LE). `https://hardwear.io/netherlands-2019/presentation/sniffle-talk-hardwear-io-nl-2019.pdf`, 2019.
21. Mike Ryan. Bluetooth : With Low Energy comes Low Security. 2013.
22. Mike Ryan. How Smart is Bluetooth Smart ? 2013.
23. Tobias Zillner and Sebastian Strobl. Zigbee Exploited : The Good, the Bad and the Ugly. `https://www.blackhat.com/docs/us-15/materials/us-15-Zillner-ZigBee-Exploited-The-Good-The-Bad-And-The-Ugly.pdf`, 2015.