Physical Tracking : menaces, performances et applications.

Taher Issoufaly

THÈSE

de

Taher Issoufaly

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE LA RÉUNION

Délivré par *l'Université de la Réunion*

Discipline: *Informatique et Télécommunications*

Physical Tracking : performances, menaces et applications

Jury

M. Emmanuel Lochin,	Rapporteur
M. Promethée Spathis,	Rapporteur
M. Rémy Courdier,	Président du Jury
M. Jérome Lacan,	Examinateur
Mme. Kim Loan Thai,	Invitée
M. Mathieu Cunche,	Invité
M. Pascal Anelli,	Directeur de thèse
M. Pierre-Ugo Tournoux,	Co-encadrant de thèse

Cette thèse a reçu le soutien financier de la Région Réunion et de l'Union Européenne - Fonds européen de développement régional (FEDER)

Remerciements

Mes premiers remerciements vont à l'encontre de Monsieur Pascal Anelli, Maître de conférences HDR à l'Université de la Réunion, qui a accepté d'encadrer cette thèse.

Je remercie tout particulièrement Monsieur Prométhée Spathis HDR à l'Université Pierre et Marie Curie et Monsieur Emmanuel Lochin Professeur à l'Institut Supérieur de l'Aéronautique et de l'Espace d'avoir accepté de juger mes travaux et d'en être rapporteur.

Je tiens à remercier Monsieur Rémy Courdier, Professeur à l'Université de la Réunion, d'avoir accepté de présider le jury de thèse ainsi qu'à Madame Kim Loan Thai et Jérôme Lacan d'avoir accepter d'être membre de mon jury.

Des remerciements spéciales vont à l'encontre de Monsieur Mathieu Cunche, Maître de Conférences à l'Institut National des Sciences Appliquées de Lyon qui m'a apporté de très bon conseils sur son expertise dans le domaine du *WiFi based Physical Tracking* lors de nos échanges durant les comités de suivi de thèse. Il me fait aussi l'honneur d'être membre de mon jury.

Enfin, mes sincères remerciements vont à l'endroit de Monsieur Pierre-Ugo Tournoux qui a proposé ce sujet de thèse et sans qui je n'aurais pu réaliser ces travaux. J'ai eu la chance de travailler avec lui depuis mon Master 2 sur le domaine du *Phyisical Tracking* et je lui suis très reconnaissant pour sa générosité, son sérieux, ses conseils et le soutien qu'il m'a apporté durant ces dernières années, même dans les moments les plus difficiles de mon parcours de doctorant.

Je remercie également le Laboratoire d'Informatique et de Mathématiques, dirigé par Monsieur Jean Diatta, Professeur à l'Université de la Réunion, ainsi que l'ensemble des membres, qui ont mis tout en oeuvre pour que le déroulement de ma thèse se fasse dans les meilleurs conditions.

Je remercie également la Région Réunion qui a soutenu financièrement ces travaux de thèse.

Bien évidemment, je remercie mes collègues doctorants pour leurs encouragements et leurs contributions.

Enfin, je tiens à remercier tout particulièrement ma femme et mon fils pour leur soutien infaillible et qui ont contribué à la réussite de cette thèse.

Résumé

La récente émergence des *smartphones* et des objets connectés a révolutionné le mode de vie des utilisateurs. Ces dispositifs ubiquitaires et équipés de plusieurs interfaces sans fil de communication, sont rapidement devenus indispensables dans la vie quotidienne des utilisateurs avec une utilisation intensive.

Les interfaces sans fil de ces objets connectés émettent périodiquement des informations, certaines sont spécifiques aux utilisateurs et permettent par effet de bord d'identifier et de suivre leur déplacements. Le suivi des utilisateurs via les informations fortuitement émises par leurs périphériques sans fil se nomme le *Wireless Physical Tracking*.

Les possibilités offertes par le *Wireless Physical Tracking* ont suscité un fort intérêt. Plusieurs applications se sont développés et ont permis d'apporter de l'innovation dans plusieurs domaines. Des sociétés de marketing l'utilisent afin de proposer à leurs clients de la publicité ciblée en fonction de leurs parcours dans leur zone d'activité. Á une échelle plus grande, les villes intelligentes, ou *smart-cities* analysent le mouvement des utilisateurs afin d'apporter des services pour le confort des habitants. Enfin, dans le domaine de la recherche, les réseaux *ad-hoc* mobiles et autres *DTN* nécessitent de s'intéresser à cette pratique car l'étude de la mobilité des utilisateurs représentent un élément clé pour améliorer les performances de ce type de réseau. Cependant, la collecte de ces informations sans le consentement des utilisateurs ou sans qu'elles soient correctement protégées représentent un risque réel pour leur vie privée.

C'est autour de ce contexte que s'articule cette thèse divisée en deux parties. La première présente les technologies *PAN* et *WAN*, l'état de l'art des méthodes de *Wireless Physical Tracking* et les contre mesures adoptés. La deuxième partie présentent les contributions de la thèse qui visent à proposer de nouvelles méthodes de suivi, analyser les performances de celles-ci face aux méthodes existantes et dans le cas particulier de l'application de *crowd-localisation*, à proposer des méthodes de suivi respectueuse de la vie privée.

Abstract

The recent rise of smart-phones and connected objects has a deep impact its users lifestyle. In 2017, more than a billion and a half smart-phones were sold around the world. These ubiquitous devices, equipped with several wireless communication interfaces, have quickly become essential in the daily life of users with an intensive use.

The wireless interfaces of these connected objects periodically transmit information on the network, some of which are user-specific and allow to identify and track their mobility. Tracking users by collecting the information generated by their wireless devices is called *Wireless Physical Tracking*.

The opportunities offered by the *Wireless Physical Tracking* raised a lot of interest. Several applications have been developed and have brought innovation in several areas. Marketing companies use it to offer to their customers targeted advertising based on their movements in their area of activity. On a larger scale, Smart Cities or *smart-cities* analyse the movement of users in order to provide services for their inhabitants. Finally, in the field of research in mobile Ad-Hoc networks and DTNs, users mobility is a key element which need to be collected and analysed.

However, the collection of this information without the consent of the users or without being properly protected induce a real risk to their privacy.

It is around this context that this thesis is focused on. It's divided into two parts. The first presents the PAN and WAN technologies, the state of the art of *Wireless Physical Tracking* methods and the adopted counter measures. The second part presents the contributions of the thesis which aims at developing new methods for Physical Tracking and analysing their performances compared to the existing methods. We first present an evaluate BPM, a bluetooth passive monitoring that allows to track the users of Classic Bluetooth device with a detection delay significantly lower than the methods previously used. We then focus on Bluetooth Low Energy and propose the use of a BLEB, a botnet of users tracking BLE objects with their smart-phones. Finally, we also focus on preserving users privacy through the proposal of PPCL, a privacy preserving crowd-localisation method which allow to track users assets without being trackable.

Table des matières

Table des figures

Liste des tableaux

Acronymes

BLE Bluetooth Low Energy. 2

CNIL Commission Nationale de l'Informatique et des Libertés. 7

IEEE Institute of Electrical and Electronics Engineers. 31

LAP Lower Address Part. 31

MAC Medium Access Control. 2

NAP Non-Significant Address Part. 31

NIC Network Interface Controller. 31

OSI Open Systems Interconnection. 1

OUI Organizationally Unique Identifier. 31

PT Physical Tracking. 3

UAP Upper Address Part. 31

WLAN Wireless Local Area Network. 4

WPAN Wireless Personal Area Network. 4

Introduction

0.1 Contexte

Les *smartphones* sont devenus un outil indispensable à la vie quotidienne. En effet, ces derniers ont connu un développement significatif ces dernières années, aussi bien en nombre qu'en capacité. Ils sont désormais de véritables micro-ordinateurs, ubiquitaires et ultra-connectés de par le fait qu'ils embarquent depuis peu de nombreuses interfaces de communication (*e.g. Bluetooth, WiFi, NFC, ANT, 3G, GSM ...*). Un effet de bord de cette nouvelle capacité de communication est que le *smartphone* échange fréquemment des données, émet périodiquement des informations de contrôles et peut à tout moment répondre à des requêtes initiées par d'autres périphériques. Le comportement du périphérique observable depuis le canal sans fil peut constituer une empreinte qu'un attaquant ou une personne indiscrète pourrait utiliser en vue d'inférer la présence de l'utilisateur.

0.1.1 Empreinte sans fil

L'empreinte d'un périphérique peut être obtenue de manière passive, l'attaquant qui est alors indétectable, n'a qu'à écouter les informations qui transitent sur le canal. L'empreinte est obtenue de manière active lorsque l'attaquant doit émettre des données sur le canal afin de solliciter une réponse du périphérique ciblée. La présence des attaquants peut alors être détectée mais pas nécessairement discernable du comportement légitime d'un utilisateur. Les données qui constituent l'empreinte peuvent être visible en clair ou nécessiter une transformation pour faire apparaître l'empreinte identifiant le périphérique. L'empreinte n'est pas nécessairement une série de bits envoyée sur le canal et peux se composer de tout éléments quantifiables tels que la dérive des horloges, le temps de réponse à une requête ou un ensemble de fonctionnalités disponibles. Une empreinte peut également être composite *c.a.d.* la combinaison d'empreintes pour constituer une empreinte unique. L'extraction d'une empreinte nécessite une connaissance approfondie des composants matériel ainsi que des protocoles de l'ensemble de la pile Open Systems

Interconnection (OSI) (en particulier la couche physique, Medium Access Control (MAC) et application).

0.1.2 Empreinte sans fil basée sur l'adresse MAC

Dans le cadre de cette thèse, nous nous focaliserons essentiellement sur les empreintes dérivées de l'adresse MAC. Celle-ci est censée être unique, permet facilement d'identifier facilement l'utilisateur et apparaît dans les données émises par la plupart des périphériques sans fil.

Prenons l'exemple du WiFi, à savoir la solution la plus répandue pour les communications à moyenne portée. Grâce à des mécanismes de chiffrement et d'authentification robustes (WPA/WPA2 [1]), cette technologie permet de sécuriser les échanges effectués sur le medium sans fil, *i.e.* a priori, personne ne peut déchiffrer ni modifier le contenu de la conversation. Bien que le chiffrement soit désormais largement adopté, les en-têtes contiennent l'adresse MAC et sont envoyées en clair. De plus, même lorsque l'utilisateur n'échange pas de données, le mécanisme de découverte de réseaux WiFi émet périodiquement des trames de contrôles. De ce fait, toute station à portée d'un utilisateur peut capter ces trames et, grâce à l'adresse MAC, inférer sa présence.

Créée en 1998, la technologie du Bluetooth appelé *Classic Bluetooth* a suscité un fort intérêt de la part des utilisateurs notamment avec l'arrivée des oreillettes, des casques sans fil ou des kit audio main libres pour voitures. Concernant son fonctionnement, lorsque des périphériques Bluetooth cherchent à détecter d'autres appareils à portée, l'adresse MAC Bluetooth est envoyée en clair dans les paquets de contrôles. De plus, lorsque deux périphériques sont en communication, il est possible d'extraire certaines parties de l'adresse MAC de l'utilisateur en décodant les paquets de données.

Le Bluetooth Low Energy (Bluetooth Low Energy (BLE)), aussi appelé bluetooth 4.0, apparaît en 2010. Bien que similaire à sa version précédente, le BLE présente des spécifications techniques différentes lors de la phase de découverte de périphériques à portée et de communication. Cette version propose de nouvelles fonctionnalités notamment pour éviter le suivi des utilisateurs en introduisant l'utilisation d'adresses MAC résolvables mais en apparence aléatoires dans les paquets. Cependant, comme nous le verrons dans ce document, ces nouvelles fonctionnalités sont assez peu utilisées et les périphériques BLE envoient des trames bien plus fréquemment que le BT et le WiFi. Cela facilite d'autant plus la détection d'un utilisateur par le biais de ses équipements connectés. Sous certaines conditions, il est également possible de récupérer des clés échangées lors de la phase d'appairage afin d'identifier les utilisateurs. A l'aide de ces clés, il est possible

de résoudre les adresses aléatoires et ainsi d'identifier les périphériques même lorsqu'ils utilisent les fonctionnalités de préservation de la vie privée du BLE.

0.1.3 Détection et suivi des utilisateurs

La détection de la présence des utilisateurs grâce à l'empreinte laissée par leurs périphériques a de nombreuses applications. Si la présence d'un utilisateur à proximité d'un attaquant ou d'une sonde contrôlée par l'attaquant est déjà une information exploitable, il est possible d'obtenir à la fois des informations référencées dans le temps et l'espace si l'attaquant utilise plusieurs sondes réparties dans la zone de déplacement. Chaque référence spatio-temporelle (identifiant de sonde et instant de détection) peut être associée à une information qualitative permettant d'enrichir la donnée de mobilité.

Comme indiqué dans l'exemple de la figure 1, s'il y a assez de stations de mesures réparties dans l'environnement il devient alors possible de capter la mobilité d'un *smartphone* équipé d'une interface WiFi/Bluetooth activée [2, 3]. Cette pratique s'appelle le *Physical Tracking* (Physical Tracking (PT)). On peut ensuite, via une phase d'analyse des données collectées, approfondir les connaissances sur un utilisateur *e.g.* comprendre son comportement par rapport à son déplacement, à sa vitesse, aux lieux visités, ou encore au temps passé devant un produit.

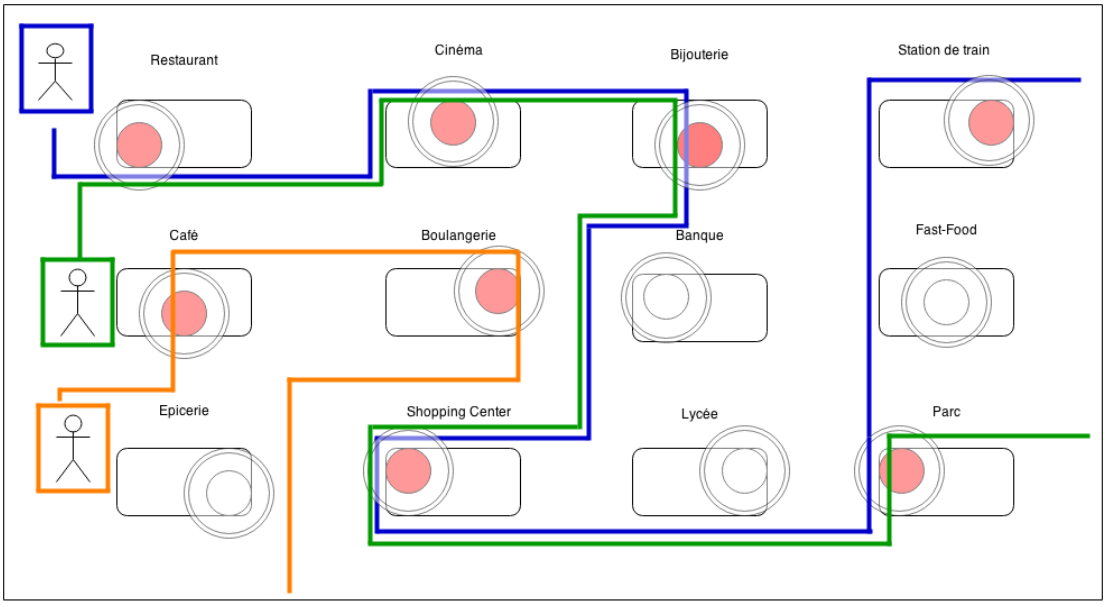


FIGURE 1: Schéma illustrant la possibilité de suivre un utilisateur à l'échelle d'une ville grâce à la collecte de l'empreinte digitale des interfaces sans fil de son *smartphone*.

0.2 Applications et problématiques

L'arrivée de ces *smartphones*, couplée avec le *physical tracking* permet la création de nombreuses applications. Cette section liste certains des domaines dans lesquels le *physical tracking*, et plus particulièrement celui basé sur le Bluetooth, le BLE ou le WiFi peut être appliqué.

0.2.1 Réseaux MANET et DTN

Les réseaux ad-hoc mobiles (MANET) et plus particulièrement les réseaux tolérants au délai (DTN) ont reçu une forte attention de part la communauté de recherche. La mobilité des utilisateurs est un paramètre clé de la performance de ces réseaux. L'évaluation des performances de ce type de réseaux est la plupart du temps, basée sur de la simulation, car il est difficile de développer des modèles analytiques adéquats ou de faire des déploiement réels. Les simulations utilisent des modèles de mobilité qui sont souvent peu réalistes et qui peuvent conduire à des résultats imprécis [4, 5]. Les traces de mobilité d'utilisateurs réels peuvent être collectées et ré-injectées durant une simulation pour pallier ce problème [6]. Une approche alternative est de collecter de telles traces pour un réseau et construire des modèles qui correspondent au schéma observé dans les traces [7–10].

L'inconvénient est que les traces de mobilité sont difficiles à collecter. Des collections typiques de traces requièrent d'acheter plusieurs périphériques et de recruter des volontaires qui porteront ces appareils sur eux [7]. Les périphériques portés par ces volontaires devraient détecter la présence de d'autres appareils ou périodiquement estimer leur position et stocker ces informations [11].

La technologie GSM a été considérée comme une source prometteuse des données de mobilité car elle peut détecter et enregistrer la présence d'un *smartphone* cible parmi les différents réseaux cellulaires [12]. Malheureusement, la position des utilisateurs n'est pas assez précise pour obtenir des opportunités de contact. De plus, les opérateurs de réseaux mobiles sont réticents à fournir les traces de mobilité de leurs clients. Par conséquent, la communauté scientifique se base sur les interfaces Wireless Local Area Network (WLAN) ou Wireless Personal Area Network (WPAN) des *smartphones*, par ex. le Bluetooth et le WiFi. Dans [11, 13], les auteurs ont utilisé la co-localisation des utilisateurs, soit via les points d'accès WiFi, soit via les stations de base GSM. Ces méthodes permettent de collecter des données concernant un grand nombre d'utilisateurs sur une longue période, mais il manque une grande partie des opportunités de contact et peut même inférer des contacts qui n'ont pas eu lieu.

Technologie	**Application mobile requise**	**Distance**	**Précision**	**Taux**
WiFi	Non	<100m	1 à 2m	80%
Caméra 2D/3D	Non	<30m	10cm	99%
Bluetooth	Non	<70m	1 à 2m	60%
BLE (beacon)	Oui	<70m	1 à 2m	de 0 à 60%*
Ultrason	Oui	<40m	10cm	de 0 à 40%*
NFC	Non	<10cm	1cm	60%
GPS	Oui	<200m	10cm	de 0 à 99%*
Laser	Non	<1m	10cm	99%

TABLE 1: Tableau comparatif des technologies utilisées pour le suivi de la mobilité des utilisateurs à l'intérieur de boutiques. (* capte uniquement les *smartphones* ayant installé l'application dédiée.)

Pour pallier ce problème, les méthodes de collecte de données étudiées dans cette thèse apportent un nouvel élan à l'évaluation des réseaux tolérants au délai. Des méthodes de mesure impliquant une infrastructure avec des sondes qui détectent la présence des utilisateurs portant leur appareil peuvent être utilisées dans ce contexte. Les opportunités de contact sont ensuite reconstruites hors ligne si la position de deux utilisateurs révèle qu'elles étaient assez proches à un moment donné. Il en va de même si deux utilisateurs sont détectés par une même sonde en même temps. Autre cas de figure plus intéressant, les *smartphones* des utilisateurs vont sonder les canaux des réseaux sans fil périodiquement afin de détecter la présence de d'autres individus à proximité sans avoir recours à une infrastructure. Toutes ces informations sont ensuite remontées vers une entité centrale où une analyse de la mobilité des utilisateurs est effectuée.

0.2.2 Publicité ciblée

Dans un contexte marketing, la connaissance du goût et du comportement des utilisateurs est essentiel pour proposer des produits auxquels ces utilisateurs présentent un intérêt. Le *Wireless Physical Tracking* a très vite été utilisé pour enrichir les services de publicité ciblée. Ces méthodes se résument à déployer des sondes captant l'empreinte digitale sans fil des utilisateurs dans les différents points d'intérêts qu'ils visitent afin de suivre leurs déplacements et d'en analyser leurs comportements. Plusieurs sociétés ont déjà déployé cette solution par le biais de l'interface WAN avec le WiFi [14–16] ou des interfaces PAN avec le Bluetooth et/ou le BLE [17, 18]. Certaines ont même opté pour l'utilisation d'ultrasons [19]. Le tableau 1 récapitule les différentes technologies utilisées pour le suivi des utilisateurs à l'intérieur de boutiques, étude réalisée par *smart-traffik* [20].

0.2.3 *Smart-cities*

Le concept de *smart-cities* ou *villes intelligentes* désigne une ville utilisant les technologies de l'information et de la communication afin d'apporter aux citoyens une amélioration des services urbains proposés. C'est dans ce contexte que l'engouement autour de l'utilisation de méthodes de PT s'est accentuée. En effet, les sondes BLE ou WiFi déployées à différents endroits permettent la détection de la présence des utilisateurs et leur suivi afin d'en dériver des informations sur le flux d'utilisateurs à l'échelle d'un bâtiment [21, 22], sur le réseau routier [23] ou lors d'évènements [24].

0.2.4 *Crowd Localisation*

Le *crowd-localisation* se définit comme la détection et le suivi d'un objet grâce aux informations remontées par une communauté vers une entité centrale. Dans ce contexte, les méthodes de PT analysant la mobilité des utilisateurs jouent un rôle important pour ce type d'applications. Des sociétés [25–27] se sont donc spécialisées dans le développement de telles solutions permettant à leurs utilisateurs de retrouver leurs objets perdus soit via l'application qui va sonder les canaux BLE à la recherche de l'identifiant de l'objet à retrouver, soit par les informations remontées par d'autres membres de la communauté qui auraient détecté la présence de cet objet. Par ailleurs, une analyse plus approfondie du *crowd-localisation* sera apportée en 6.

Cependant, il faut remarquer que les informations relatives à la mobilité de l'utilisateur posent un certain nombre de problèmes d'un point de vue du respect de la vie privée. Bien que les utilisateurs soient plutôt informés que les applications installées sur leurs *smartphones* traquent leur déplacement, le suivi par des périphériques tiers est bien moins connus et représente un danger réel pour la vie privée des utilisateurs [28, 29]. A l'heure actuelle, ces considérations ne sont que très peu prises en compte dans les solutions commerciales, notamment de par le fait que la *crowd-localisation* soit une application très récente. Les solutions déployées prétendent protéger l'utilisateur via des fonctions de *hash* des adresses MAC et une réduction de l'espace des identifiants. Il a récemment été montré que ces opérations sont réversibles en quelques minutes [30], ce qui implique que les méthodes actuellement utilisées sont inappropriées. De plus, la mise en place de méthodes exploitant le PT pour la collecte de données de mobilité des utilisateurs est simple d'utilisation et dans la plupart des cas ne requiert que peu d'efforts de la part d'un attaquant. Ceci est principalement dû à l'utilisation d'anciennes normes de technologies sans fil, à des périphériques mal-configurés ou à un manque de vigilance des utilisateurs, mal informés qui n'ont pas conscience des éventuelles risques pour leur vie privée. C'est autour des performances des méthodes de PT et des risques liés à la

vie privée des utilisateurs que cette thèse s'articule. Les applications utilisant le *PT* ne devraient pas négliger cet aspect car la Commission Nationale de l'Informatique et des Libertés (CNIL) [31] se penchent d'avantage sur ce sujet et notamment sur la protection des informations liées à la vie privée des utilisateurs. Plus récemment, l'introduction de la RGPD (Règlement Européen sur la Protection des Données) a été annoncée. Les entreprises doivent répondre aux exigences de ces règlements qui stipulent que la récolte des données privées doit faire l'objet d'un consentement de l'utilisateur. De plus, les données doivent être anonymisées de façon irréversible. Des actions ont déjà été prises à l'encontre des start-ups *Fidzup* et *Teemo* qui mettaient en cause l'absence de consentement des utilisateurs pour les données collectées à des fins de ciblage publicitaire [32].

0.3 Contributions et plan

Nous avons remarqué que malgré le récent engouement pour les solutions basées sur le *Physical Tracking*, la communauté scientifique ne dispose que de peu d'informations concernant ses performances réelles, les menaces et les différents cas d'utilisations. Parmi la pléthore de technologies existantes en lien avec le *PT*, nous avons décidé d'étudier dans cette thèse les technologies WAN et PAN car ces dernières sont principalement utilisées de façon quotidienne par les utilisateurs notamment suite à l'engouement pour l'utilisation des *smartphones* et plus récemment des objets connectés.

Ce manuscrit est composé de trois parties.

La première s'intéresse aux technologies WLAN et PAN dans un contexte de *physical tracking* et se focalise sur le WiFi 1, le *classic Bluetooth* 2 et le *Bluetooth Low Energy* 3. Pour chacune de ces technologies, nous présentons les différents mécanismes et failles permettant d'établir des méthodes de suivi de mobilité des utilisateurs. Par suite, un état de l'art est présenté pour mieux comprendre l'évolution du *PT* au travers de l'évolution des trois technologies sans fil étudiées. Enfin, une dernière partie, placée en annexe est consacrée aux outils techniques qui peuvent permettre la mise en place du *PT*.

La seconde partie présente les contributions de cette thèse. Elle est constitué de trois chapitres.

La première contribution (Chap. 4) traite de la technologie du *classic Bluetooth* où une nouvelle méthode passive de collecte de données pendant une communication entre des périphériques *Bluetooth* est présentée. Elle s'avère d'un intérêt particulier car la détection des utilisateurs basée sur les attaques actives du *Bluetooth* ne sont désormais plus satisfaisantes. En effet, les périphériques récents ne restent que peu de temps en mode

découvrable et réduise fortement les chances de réponses lors de scans de découvertes actives.

La seconde (Chap. 5) s'intéresse au *Bluetooth Low Energy* où nous présentons une application de type *botnet* permettant de faire du *PT* à grande échelle (sec. 5). En effet, la mise à jour de la norme *Bluetooth* a apporté plusieurs fonctionnalités, avec notamment l'introduction d'adresses mac aléatoires pour éviter le traçage des individus. Cependant, nous allons montrer qu'en pratique, il existe un potentiel risque de suivi à grande échelle des utilisateurs en déployant une application mobile officielle.

La dernière contribution (Chap. 6) se focalise sur les solutions utilisant le *crowd-localisation* *i.e.* une méthode qui permet de localiser des items selon les informations remontées par la communauté. Nous mettons en avant les principales menaces de cette méthode et présentons de possibles solutions dont le but est de garantir la sécurité concernant les informations de la vie privée des utilisateurs.

Enfin, nous terminons cette thèse avec une conclusion générale, ainsi que les perspectives pour la poursuite des travaux de cette thèse (sec. 7).

Première partie

Technologies sans fil et Physical Tracking

Chapitre 1

WiFi

1.1 Aperçu du *WiFi*

Cette section présente les points clés du *WiFi* afin de faciliter la compréhension du *Physical Tracking* avec cette technologie. La sous-section 1.1.1 introduit le contexte en présentant les différents standards, le mode de communication et les principales couches à analyser. Par la suite, les sous-sections 1.1.2 et 1.1.3 présentent les différents mécanismes de fonctionnement de la couche physique et MAC qui permettent de récolter des informations privées afin de déployer des solutions de suivi de mobilité des utilisateurs. Parmi les éléments indispensables pour faire du *WiFi-based Physical Tracking*, nous allons retrouver des attaques passives via l'écoute passive de communication ou durant le processus de découverte de point d'accès à portée mais aussi par l'utilisation d'attaque active avec la création de faux point d'accès. Pour bien appréhender toutes ces attaques, nous allons présenter dans cette section les éléments indispensables à leur compréhension.

1.1.1 Contexte

Les protocoles de communication du *WiFi* sont régies par l'*IEEE* (*Institute of Electrical and Electronics Engineers*) qui ratifiait en 1997, son premier standard appelé *IEEE* 802.11. Cette norme permettait d'atteindre des débits théoriques de 1 à 2 Mb/s avec les ondes radios de fréquence à 2.4 GHz. En effet, la bande de fréquence *IEEE* à 2.4 GHz a été choisie car elle avait l'avantage d'être utilisable sans licence dans de très nombreux pays. Plusieurs versions du standard 802.11 ont été déployées. Elles se différencient selon le débit proposé, la bande de fréquence, le nombre de canaux et la modulation radio utilisée. La figure 1.1 présente les différentes normes du *WiFi* selon la banque de fréquence

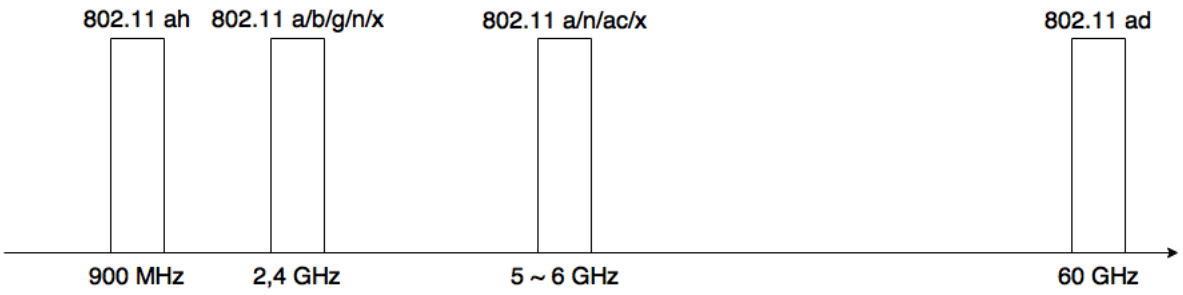


FIGURE 1.1: Les différentes normes du *WiFi* selon la bande de fréquence utilisée.

utilisée. Par exemple, le 802.11b et le 802.11g, compatibles entre eux, fonctionnent tous deux avec les ondes radio d'une fréquence de 2,4 GHz.

Deux types de topologies de réseaux existent en *WiFi*, le mode *infrastructure* et le mode *ad-hoc*. Dans les réseaux de type *infrastructure*, chaque périphérique du réseau est lié à un point d'accès *WiFi* 1.2. On associe le périphérique à un client et le point d'accès à un maître. Ce point d'accès couvrant une zone est identifié par un *BSSID* de 48bits qui correspond à l'adresse MAC du point d'accès. Le nom du réseau est identifié par la valeur du *SSID*, par exemple "WiFiMcDoSainteMarie". Dans les réseaux en mode *ad-hoc*, les nœuds du réseau peuvent communiquer directement entre eux s'il sont à portée de transmission, sans passer par un intermédiaire. Néanmoins, ce type de réseau nécessite une configuration préalable au niveau de chaque nœud pour pouvoir échanger des données. Dans le cadre du *PT*, le mode *infrastructure* est celui qui sera considéré.

(A) Mode *ad-hoc*

(B) Mode infrastructure

FIGURE 1.2: Communication en mode *ad-hoc* et infrastructure

Les éléments normalisés par le *WiFi* concernent la couche physique et liaison du modèle *OSI*.

1.1.2 Couche physique

Cette sous-section décrit les informations liées à la couche physique au travers des différents canaux utilisés par le *WiFi* selon la bande de fréquence (1.1.2.1), de la composition d'une trame physique et des différentes méthodes de modulation (1.1.2.2).

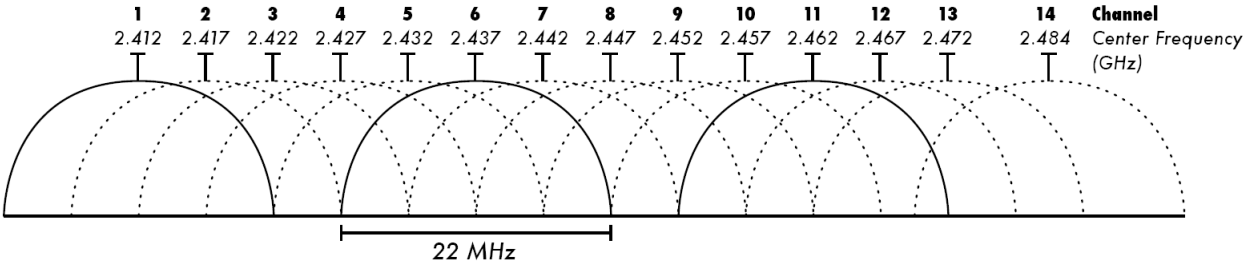


FIGURE 1.3: Liste des canaux *WiFi*. Les canaux 1,6 et 11 peuvent être utilisés simultanément (source : Michel Gautier `https://en.wikipedia.org/wiki/List_of\_WLAN_channels#/media/File:2.4_GHz_Wi-Fi_channels_(802.11b,g_WLAN).svg`)

1.1.2.1 Les canaux

Les différentes versions du standard 802.11 découpent la bande de fréquence sur laquelle elles opèrent, en canaux. Chaque canal possède une fréquence centrale. Le signal modulé occupe une bande à 22 MHz (802.11b), 20 MHz (802.11 g/n), 40, 80 et 160 MHz (802.11n/ac/ad). Si l'on prend le cas de la bande de fréquence à 2,4 Ghz, le spectre radio commence à 2400 MHz jusqu'à 2 483,5 MHz, découpée en 14 canaux dont le premier canal possède une fréquence central située à 2412 Mhz et le dernier à 2,484 Mhz.

Lors du déploiement d'un réseau *WiFi*, les canaux qui ne se recoupent pas, peuvent être utilisés aux mêmes endroits. Par exemple, pour la bande de fréquence à 2,4 Ghz, les canaux 1, 6 et 11 ont été les plus utilisés en Europe ces dernières années.

Lorsqu'une interface *WiFi* est associée à un point d'accès elle communiquera sur ce canal jusqu'à ce qu'elle se dé-associe. Un attaquant à portée de l'interface *WiFi* ou du point d'accès n'aura donc qu'à écouter sur le canal utilisé pour écouter les échanges et tenter d'inférer des informations identifiant le smartphone. Lorsqu'une interface *WiFi* est activée mais n'est pas associée à un point d'accès, elle effectuera périodiquement (entre 15s et 90s) une procédure de découverte des réseaux disponible (*a.k.a. scans WiFi* détaillé dans la section 1.1.3.2). L'interface *WiFi* envoie alors des trames sur l'ensemble des canaux *WiFi* et standards supportés. Un attaquant souhaitant écouter les trames émises n'aura donc pas à se soucier du canal sur lequel il écoute. Quel que soit le canal sur lequel il écoute, il recevra une des trames émises par l'interface.

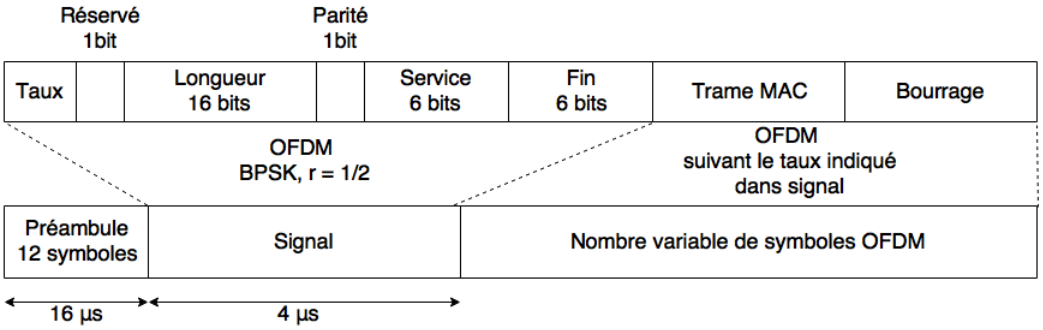


FIGURE 1.4: Décomposition d'une trame 802.11.

1.1.2.2 Trames 802.11 et modulations

La couche physique est chargée d'émettre la trame sur un des canaux WiFi. Prenons à titre d'exemple la norme 802.11b. La série de bits que constitue la trame est d'abord soumise à une méthode d'étalement de spectre (*e.g. DSSS, CCK*) ou un code correcteur d'erreur (*e.g. BCC, LDPC*) puis à une modulation telle que *BPSK, DPSK, QAM.*

Une trame 802.11 est composée d'un préambule, d'une en-tête physique suivie d'une entête MAC et des données. La figure 1.4 décrit la composition d'une trame 802.11a. Le préambule permet d'opérer la synchronisation. L'entête physique (noté champ signal dans la figure), qui est très courte comparée au reste de la trame, est toujours envoyée avec la même modulation et peut donc être décodée par n'importe quelle interface *WiFi* conforme au standard. Cette entête physique contient notamment les informations sur la modulation utilisée pour coder le reste de la trames et le code correcteur d'erreur. Par conséquent, une interface écoutant une trame 802.11 aura la possibilité de démoduler toutes les trames *WiFi* qu'il entendrait, qu'il en soit le destinataire ou non.

1.1.3 Couche MAC

Cette sous-section décrit les points clés de la couche MAC, à savoir comment accéder au média et les différentes étapes du processus d'association entre périphériques *WiFi*(1.1.3.1. Cette dernière met en avant les services de découverte de réseaux à portée utilisés 1.1.3.2, la composition des trames de balises (*probes request* et *probes response*) et le processus d'authentification, étape préalable avant la communication.

1.1.3.1 Le processus d'association

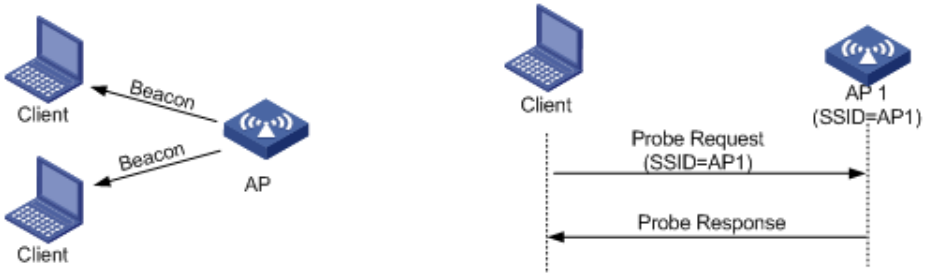
Pour pouvoir communiquer, les périphériques doivent d'abord procéder à un processus d'association. En mode *infrastructure*, chaque point d'accès va émettre périodiquement

des trames « balises » (*beacon frame)* dont le but est de garantir la synchronisation de toutes les stations clients qui lui sont associées. Ces trames contiennent plusieurs informations sur le point d'accès comme le *BSSID* (son adresse MAC) ou encore le *SSID* (le nom du réseau).

Ces informations sont envoyées en clair sur le canal. Le standard 802.11 a donc introduit deux modes de services permettant la découverte de réseaux à portée des stations clientes : le mode passif et le mode actif.

1.1.3.2 Service de découverte de périphériques à portée

Pour pouvoir s'associer, il faut d'abord détecter la présence des points d'accès à portée. Pour mieux comprendre le fonctionnement, la figure 1.5 illustre les deux modes du service de découverte de périphériques à portée utilisés par les stations clientes.



(A) Mode passif

(B) Mode actif

FIGURE 1.5: Service de découverte de point d'accès du standard IEEE 802.11.

Mode passif : Le point d'accès informe les stations clientes à portée en utilisant des trames balises. Les stations écoutent passivement ces trames à travers l'ensemble des canaux disponibles. Vu que les informations sont envoyées en clair, il est possible de lister tous les points d'accès à portée en plus de leur identifiant unique : leur adresse MAC. Pour rappel, l'adresse MAC est un identifiant numéroté sur 48 bits, soit 6 octets. Les 24 premiers bits représentent l'*OUI (Organizationally Unique Identifier)*, qui désigne le fabricant du matériel. Les 24 bits suivants appelé *NIC* pour *Network Interface Controller* sont assignés de manière à ce que l'adresse soit unique sur le réseau (voir 1.6).

Mode actif : La station cliente (*smartphone*) recherchent activement les points d'accès connus en "sondant" chaque canal. Elle diffuse périodiquement des requêtes de sondage appelées *probes requests* (voir figure 1.7) sur chacun des canaux et attend un message de réponse appelés *probe response message*. Les messages *probes requests*

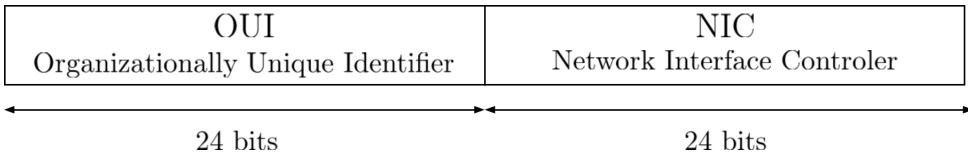


FIGURE 1.6: Structure d'une adresse MAC *WiFi*.

sont des trames de type *management* qui intègrent notamment le *SSID (Service Set Identifier)* des points d'accès ciblés. Ce processus est répété sur chaque canal jusqu'à qu'une station s'associe avec succès à un point d'accès. Dans ce cas présent, deux identifiants non-protégés sont émis périodiquement. Les messages émis par la station sont envoyés en clair et contiennent son adresse MAC en plus du nom du réseau dont la station cherche à détecter. De plus, la réponse envoyée par les points d'accès contient en clair leur identifiant.

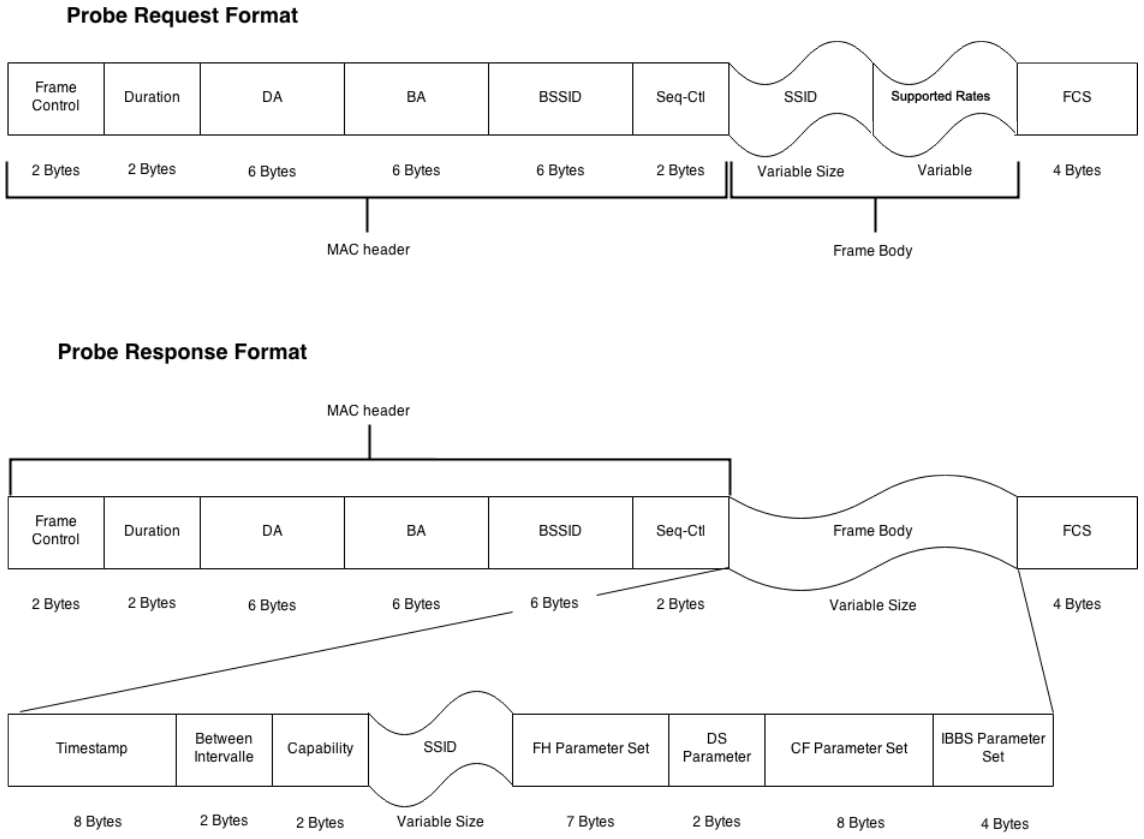


FIGURE 1.7: Structure des « probes requests » et des « probe response ».

Comparé au mode passif, le mode actif présente l'avantage de réduire le délai de découverte du réseau. De plus, le mode de service actif est requis dans le cas où des réseaux sans fil posséderaient un *SSID* caché. Les points d'accès cachés n'émettent pas de messages balises et par conséquent, le service de découverte passif ne peut pas être utilisé. Le mode

actif est donc l'unique alternative et est utilisé par les *smartphones*. De plus, le processus de découverte actif est présent dans la plupart des systèmes d'exploitations (*e.g* iOS, Google Android, Windows). Un *smartphone* utilisant un de ces systèmes d'exploitations, va continuellement diffuser des *probes requests*.

Dans un contexte de *PT*, l'envoi d'identifiant en clair sur le réseau présente un risque considérable à la vie privée des utilisateurs qui font office de stations clientes par le biais de leur *smartphone*. L'adresse MAC qui est unique, permet d'identifier l'utilisateur et de suivre sa mobilité à plus grande échelle (campus, université, ville) si l'on place par exemple, des capteurs à différentes zones qui collecteraient périodiquement les *probes requests* émises par les *smartphones*. La procédure de collecte de ces données est totalement passive et représente une réelle menace pour la sécurité des utilisateurs.

1.1.3.3 Authentification

Après avoir détecté la présence d'un point d'accès auquel la station cliente veut se connecter, un processus d'authentification est enclenché. Pour s'identifier, la station va envoyer une requête d'authentification. Si le réseau est sécurisé par l'utilisation d'une clé secrète *(WEP/WPA/WPA2)*, le point d'accès va renvoyer un trame de réponse. Le client va réémettre une trame chiffrée au point d'accès, qui va pouvoir vérifier si celle-ci est légitime. Cependant, le point d'accès peut prouver la légitimité des clients qui s'authentifient à lui mais l'inverse n'est pas vrai. Un client ne peut pas savoir si la station avec laquelle il communique est légitime.

De ce fait, dans notre contexte de *PT*, de potentielles attaques peuvent être élaborées pour inciter les utilisateurs à se connecter à de faux points d'accès crées afin de récolter des informations de la station cliente. Par exemple, si les *smartphones* utilisent des adresses MAC aléatoires, l'authentification va révéler son adresse réelle.

1.2 État de l'art

L'état de l'art regroupe les travaux effectués avec le *WiFi* dans un contexte de *PT*. De prime abord, nous analyserons les différentes méthodes de création d'une empreinte *WiFi* unique permettant d'identifier un utilisateur. Par suite, nous mettrons en avant les travaux montrant la possibilité de falsifier la position géographique d'un utilisateur qui dans certains cas peut permettre d'associer un identifiant à une personne physique réelle. Ensuite, nous nous intéresserons aux travaux qui présentent différents systèmes de suivi de la mobilité des utilisateurs avec des expérimentations donnant un aperçu des

performances en terme de temps de détection et du nombre d'utilisateurs détectés selon la méthode utilisée. Enfin, la dernière partie se consacre aux contre-mesures présentées dans la littérature pour permettre d'apporter une sécurité sur les informations liées à la vie privée des utilisateurs.

1.2.1 Empreinte *WiFi*

Une empreinte désigne un ensemble d'informations permettant d'identifier un objet de manière unique. Pour cela, une collecte de données est effectuée pour rassembler le plus d'informations possible. Le but est de construire un identifiant stable et unique d'un noeud selon la quantité d'information obtenue. Un noeud peut être de plusieurs natures : navigateur web [33], des véhicules [34, 35], des personnes au travers de leurs *smartphones* [36], des périphériques de *l'Internet Des Objets*[37] ou encore de nœuds d'un réseau représenté par des capteurs sans fil[38].

Au cours de cette section, nous discuterons et analyserons différentes méthodes d'empreinte *WiFi*.

1.2.1.1 Empreinte de la couche Physique

L'identification avec une empreinte de la couche physique consiste à identifier un dispositif basé sur des imperfections de transmission présentées par son émetteur-récepteur radio. L'utilisation de cette méthode est très ancienne et date de la 2ème guerre mondiale. Elle a été notamment utilisé durant la guerre du Vietnam où les signaux de code Morse collectés via la couche physique étaient désignés comme une empreinte. Les opérateurs radios analysaient ces empreintes en fonction de la fréquence du signal et de l'amplitude pour déterminer si le message venait d'une source alliée ou ennemie. De plus, une analyse plus poussé était effectuée afin de détecter des similarités et de pouvoir identifier la source du message. Cependant, cette méthode n'était pas entièrement fiable et était d'avantage utilisée en combinaison avec d'autres outils d'identification.

Plus récemment, des travaux ont été proposés pour estimer la faisabilité de cette méthode. Dans [39], les auteurs établissent comme empreinte le signal analogue des cartes Ethernet. Dans [40], ils décrivent la faisabilité d'effectuer des attaques d'usurpation d'identité basée sur la modulation des signaux de la couche physique.

De manière générale, ces méthodes sont peu utilisées notamment dû au fort coût de matériel à utiliser pour le bon fonctionnement des ces techniques comparé au coût d'une carte réseau nécessaire à établir une empreinte *WiFi* sur la couche MAC.

1.2.1.2 Empreinte du brouilleur *OFDM*

La norme 802.11 utilise plusieurs méthodes de modulation pour la transmission de données. L'une d'entre elle se nomme *OFDM (Orthogonal Frequency Division Multiplexing), voir figure 1.9.* Elle permet une utilisation optimale de la bande de fréquence, tout en apportant une très bonne résistance aux interférences. Dans le but de réduire les erreurs de transmissions, les données dans les trames sont « brouillées » comme le montre le figure 1.8. Cette phase est réalisée grâce à un registre de décalage à rétroaction linéaire (*LFSR (Linear Feedback Shift Register)*) qui va généré une séquence de bits pseudo-aléatoire. Cette dernière va être combinée avec les données à l'aide de l'opérateur « OU exclusif ».

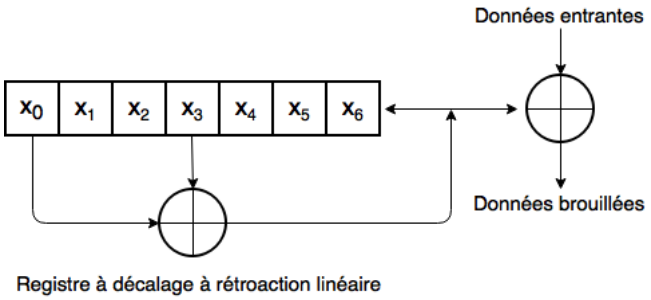


FIGURE 1.8: Brouilleur de données utilisés pour les trames 802.11

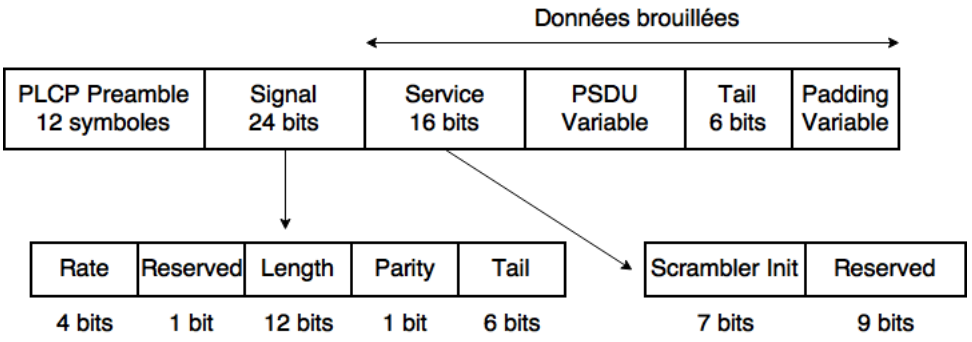


FIGURE 1.9: Format des trames *OFDM*

La graine utilisée lors du processus de "brouillage" des données peut être reconstruite à l'aide d'un champ de la trame ODFM intitulé *Scramble Init.* Ce champ change donc de manière prévisible. De ce fait, il est possible en collectant plusieurs trames OFDM de déterminer si celles-ci appartiennent à un même périphérique, même si ces trames ne contiennent aucune informations sur l'identifiant du périphérique.

Les premiers chercheurs à avoir été étudié cette technique l'ont appliqué dans le domaine des véhicules. En effet, dans [41], les auteurs présentent une attaque permettant d'élaborer une empreinte associé à un véhicule utilisant des cartes réseaux supportant la norme 802.11p, le standard utilisé pour les véhicules (*WAVE (Wireless Access in Vehicular Environments)*).

Dans [42], les auteurs présentent *Wobly*, un algorithme permettant d'associer une signature basée sur des attributs qui mesurent l'allure des utilisateurs. Cette méthode collecte passivement les trames *beacons OFDM* encodées selon le mouvement des utilisateurs pour en extraire une signature unique permettant le traçage des individus. En appliquant une classification naïve bayésienne sur les données générées, les expérimentations montrent que dans le meilleur des cas, un taux de 87% d'identification correcte des individus a été calculé.

De manière similaire, les auteurs dans [43] présentent un système nommé *WiFi-ID* qui analyse les informations sur l'état des canaux pour extraire des fonctionnalités uniques qui sont représentatives du style de marche/allure d'un individu permettant ainsi d'identifier cette personne. Leurs expérimentations ont démontré que leur système peut identifier de manière unique des personnes ayant une précision moyenne de 93% à 77% d'un groupe de 2 à 6 personnes respectivement.

1.2.1.3 Empreinte basée sur la dérive de l'horloge

La plupart des périphériques possèdent une horloge interne utilisée pour calculer le temps écoulé. Ces horloges dérivent de plus de plus du le temps global universel dû notamment à des imperfections matériels. De ce fait, il est possible de prédire la valeur des horloges où la variation atteint une grandeur de plusieurs parties par million (ppm), *i.e.* 10^{-6} secondes.

En 2005, Kohno et al. [44] ont introduit le concept d'empreinte avec la dérive des horloges inspirée par les travaux effectués par Moon et al. [45]. Ils ont mis en avant l'idée d'identifier un périphérique physique au lieu d'un système d'exploitation ou un pilote. Pour cela, ils ont décidé d'utiliser la dérive de l'horloge d'un périphérique comme identifiant. Ils ont montré que l'horloge d'un périphérique dérivait lentement selon le temps (0 à 4 ppm durant une période de 24 heures), tandis que les variations selon les périphériques étaient assez grandes pour permettre d'identifier un périphérique. Leur méthode consiste à interagir avec les périphériques cibles en utilisant des requêtes pour récupérer la valeur du *timestamp*. Cette méthode fonctionne même si les cibles sont distant de l'attaquant.

Pour mener à bien une attaque identifiant un périphérique en utilisant la dérive de son horloge, il faut avoir accès à différent types d'estampilles temporelles (*timestamp*) du périphérique cible. Plusieurs *timestamps* (*ICMP timestamp* [46], *TCP timestamp* [47], *NTP timestamp* [48], *TSF timestamp* [49]) existent et ont été étudiés au sein de la communauté scientifique.

Les *timestamps ICMP, TCP et NTP* sont issus des protocoles des couches réseaux et inférieurs. Le *timestamp TSF* est utilisé uniquement par les points d'accès, dans les

trames de *beacons* et les *probes responses*. De ce fait, les *smartphones* non-connectés à un point d'accès ne génèrent aucun *timestamp*. Par conséquent, la méthode d'empreinte par dérive de l'horloge est inutilisable avec les *smartphones* dans le cas du *WiFi*.

Ils existent néanmoins des travaux qui montrent que cette méthode fonctionne dans le cas où les *smartphones* sont connectés à un point d'accès. En effet, Cristea et Groza en 2013, ont montré l'utilisation d'une attaque active qui envoie des requêtes *ICMP timestamp* aux *smartphones* qui répondent avec une *ICMP timestamp response*. En calculant la dérive de leur horloge, ils démontrent la possibilité de pouvoir identifier les *smartphones*.[50].

De manière similaire, dans [51] Lu et al. ont amélioré la méthode précédemment décrite en utilisant un processeur qui supprime les valeurs aberrantes et un algorithme d'identification modifiée. Par conséquent, ils n'ont besoin que de quelques secondes pour pouvoir identifier un périphérique avec cette technique.

En 2010, Arackaprambil et al. ont étudié l'impact du processus d'empreintes par dérive de l'horloge. Dans [52], ils ont démontré que les techniques précédemment décrites présentaient des limites, notamment sur le fait que ces méthodes n'étaient pas suffisantes pour identifier un point d'accès dont l'identité aurait été usurpée par un faux point d'accès qui aurait modifié la valeur de son horloge par la valeur de l'horloge de la cible. De ce fait, une nouvelle méthode a été proposée en utilisant une horloge plus précise et de nouveaux paramètres afin de détecter de telles tentatives d'usurpation.

Dans [53], Huang et al. ont étudié une technique d'identification d'un nœud d'un réseau de capteur sans fil à l'aide du protocole *FTSP (Flooding Time Synchronization Protocol)*. Les expérimentations montrent que presque toutes les dérives de l'horloge d'un capteur diffèrent à l'intérieur d'un espace restreint. Pour deux nœuds dont les dérives de l'horloge sont très proches l'une de l'autre, une fonction de classification est proposée pour vérifier la continuité linéaire des dérives de l'horloge mesurée.

L'empreinte avec la dérive de l'horloge générée par un périphérique de mesure est calculé selon sa propre horloge interne. De ce fait, les observations sont relatives au périphérique qui collecte les données et ne peuvent être utilisées comme une empreinte universelle car la construction d'un tel système est très compliqué à mettre en place, notamment sur le fait que chaque nœud du système devrait constamment être au courant de la valeur de la dérive de son horloge par rapport à tous les autres nœuds.

Dans ce contexte, Lanz et al. dans [54] ont proposé un algorithme passive d'identification par empreinte en utilisant les *timestamps* régulièrement envoyés par les points d'accès dans les trames de *beacons*. L'avantage de leur méthode est qu'ils ont été capable d'éliminer l'influence de l'horloge des périphériques mesurant les données collectées. De ce fait, les données remontées par plusieurs périphériques étaient comparables. Cependant,

les expérimentations à grande échelle ont montré que parmi les points d'accès et les périphériques de mesures, il y avait un trop grand nombre de dérive d'horloge similaire. De ce fait, ils ont concluent que la dérive de l'horloge ne pouvait être suffisante pour calculer une empreinte unique d'un point d'accès.

1.2.1.4 Périphérie d'un périphérique associé à un point d'accès

L'*Internet des Objets* ou *IoT* est en marché en plein essor depuis ces dernières années. Sa définition peut se traduire par la faculté d'un objet à récolter des informations dans son environnement et à partager ces données sur le réseau. Ce phénomène en pleine expansion doit son succès à l'arrivée sur le marché des *smartphones* et des nombreux objets connectés qui ont changé le quotidien et le mode de vie des utilisateurs. Ces périphériques sont de vrai micro-ordinateurs ubiquitaires, ultra-connectés et sont dotés de plusieurs interfaces de communications.

On s'intéresse dans cette partie aux méthodes de détection d'empreintes digitales sans fil sur des périphériques associés à un point d'accès. Pang. et al ont analysé le trafic des périphériques associés à un point d'accès (*hotspot*) publique. Sachant que l'adresse MAC peut être envoyée en clair dans les *probes requests*, l'individu cible peut être traçable. Pour pallier ce problème, de nombreux travaux ont proposé d'utiliser des pseudonymes comme identifiants. Certains auteurs ont démontré que l'utilisation de pseudonymes était insuffisant pour éviter le traçage des périphériques. En effet, d'autres informations peuvent être utilisées comme identifiant tels que la destination des trames, la taille des trames, les noms des réseaux auxquels le périphérique s'est déjà connecté qui sont contenus dans les *probes requests*, les options de l'en-tête MAC ou encore les caractéristiques du trafic généré par un périphérique. Les expérimentations ont montré qu'un adversaire peut identifier 64% des utilisateurs avec une précision de 90% lorsque ces derniers passaient une journée connectés à un point d'accès publique [55].

Gopinath et al. ont étudié l'implémentation du *random backoff algorithm* qui est un algorithme de type *CSMA/CA* utilisé pour limiter la charge du réseau quand une collision se produit entre deux messages émis simultanément par deux stations. Il s'est s'avéré que plusieurs normes du standard 802.11 n'étaient pas implémentées. Plus précisément, dans certains cas, les périphériques peuvent ne pas être conformes à la norme 802.11 alors que dans d'autres cas, ils peuvent différer dans des détails significatifs qui ne font pas partie des spécifications obligatoires de la norme. De ce fait, ces variations permettent d'établir une empreinte pouvant identifier la carte réseau utilisée d'un périphérique.[56]

Neumann et al. ont étudiées des méthodes passives de génération d'empreintes en analysant les paramètres d'une simple carte réseau. Ils ont analysé différents paramètres : le

taux de transmission de trames, la taille d'une trame, ou encore le modèle de transmission des *probes requests*. Cependant, les résultats ont montré que les paramètres les plus efficaces pour l'identification de périphériques étaient le temps inter-arrivée des trames et le temps de transmission des trames.[57]

Franklin et al. ont étudié le temps inter-arrivée des *probe requests*. Cette valeur peut être exploitée pour construire une empreinte pouvant identifier la carte réseau d'un périphérique. Elle est construite en comptant le nombre et la valeur moyenne des temps inter-arrivé des *probe requests* dans chaque bloc. Un bloc est une valeur interne utilisée à la place des valeurs réelles collectées en convertissant un intervalle de point de données continues en valeurs discrètes [58].

O'Hanlon et al. ont montré et analysé les protections insuffisantes pour protéger les données pouvant identifier un périphérique. Ils ont trouvé une faille de sécurité dans le protocole d'authentification du *WiFi* qui permet à un adversaire à l'aide d'attaques passives, d'obtenir et de tracer la valeur de l'*IMSI (International Mobile Subscriber Identity)* d'un périphérique [59].

1.2.1.5 Probes requests

Dans un contexte de *PT* avec le *WiFi*, les *probes requests* émises périodiquement par les *smartphones* peuvent être utilisées pour inférer plus d'informations sur les utilisateurs.

Dans [60], Robyns et al. ont utilisé une base de donnée en collectant des *probes requests* pour inférer des statistiques sur les modèles de *smartphones* utilisés afin d'obtenir une estimation du nombre de périphériques affectés par une faille de sécurité qui peut conduire à un risque de traçage de l'appareil.

Dans [61], Seneviratne et al. se sont intéressés à la sémantique que peut apporter le *SSID* (nom du réseau). A l'aide d'une base de données d'environ 120 000 points d'accès *WiFi* associés à leurs position géographique, ils ont utilisé plusieurs métriques de similarité afin de faire la corrélation entre le nom d'un point d'accès *WiFi* et des sites connus (fast-food, bars, théâtre, cinéma, centre commerciaux ...). Les résultats montrent qu'avec un niveau de similarité le plus grand, le taux de précision entre le nom d'un point accès *WiFi* et un endroit atteignait plus de 97%.

Dans [62], Diluzio et al. ont utilisé les *SSIDs* contenus dans les *probes requests* pour inférer le lieu d'origine des participants de grands évènements (*i.e.* conférence) en utilisant la base de donnée publique fournis par *WiGLE*[63] qui permet d'associer une position géographique selon le nom d'un point d'accès.

Dans [3, 64], Cunche et al. ont présenté un mécanisme permettant de détecter des liens sociaux entre des individus en exploitant les *probes requests* émises par l'interface *WiFi* des *smartphones*. En analysant ces trames, il est montré que la liste des réseaux auxquels un *smartphone* s'est déjà connecté apparaissait en clair. De ce fait, une similarité entre plusieurs listes différentes induirait un possible lien social entre ces utilisateurs.

Ces mêmes auteurs ont publié dans [2] deux attaques permettant de récolter passivement l'adresse MAC *WiFi* et l'ensemble des réseaux auquel l'utilisateur s'est connecté et de lier ces informations à l'identité réelle d'une personne. La première se nomme *Beacon Replay Attack* qui va récolter, dans un premier temps, l'ensemble des réseaux auquel le *smartphone* s'est connecté en analysant les informations émises par les *probes requests*. L'attaque va ensuite répliquer et ré-émettre ces trames pour faire interagir le *smartphone* cible et ainsi pouvoir associer une personne réelle aux données précédemment collectées.

La seconde méthode, appelée *stalker attack*, a pour objectif :

- d'identifier une personne physique cible ;
- de sonder les canaux *WiFi* en collectant les *probes requests* envoyées et en stockant l'identifiant de chaque *smartphone* à savoir dans ce cas, l'adresse MAC *WiFi* ;
- de suivre la cible tout en s'assurant qu'elle soit à portée de la sonde ;
- chercher dans les données récoltées, l'identifiant qui est présent durant toute la capture ;

Après application de cet algorithme, le résultat va donc associer l'adresse MAC *WiFi* à une personne réelle.

Dans [65], Jamil et al. détectent le changement de comportement des utilisateurs de *smartphones* grâce à la fréquence d'envoi des *probes requests* générées par leur appareil. En effet, grâce à une classification par un arbre de décision, ils ont été capable de détecter lorsque le *smartphone* a son écran éteint ou allumé avec un taux de précision de 93% et jusqu'à 100% pour certains types de *smartphones*.

Dans [66], Rebondi et al. ont utilisé plusieurs méthodes de classification afin de déterminer si une *probe request* a été envoyée d'un *smartphone* ou d'un ordinateur portable en exploitant diverses informations comme le nombre de *probes requests* envoyées, la fréquence d'envoi, la puissance du signal ou encore l'*OUI* (fabricant du matériel).

Notons toutefois que la collecte et l'analyse des *probes requests* comme source d'information peut aussi avoir un impact positif et non-intrusif à la vie privée des utilisateurs comme dans [67], où les auteurs extraient des informations sur les canaux *WiFi* contenues

dans les *probes requests* pour améliorer la gestion du spectre afin d'éviter une surcharge ou un échec au bon fonctionnement des points d'accès.

1.2.2 *WiFi* et falsification de la position géographique

Dans [68], Cunche et al. mettent en avant une méthode permettant de faire le lien entre l'adresse MAC *WiFi* d'un *smartphone* et l'identité d'une personne grâce aux applications de réseaux sociaux qui envoient des informations géolocalisées à l'aide du *WPS (WiFi Positionning System)*. Ce papier explique les étapes nécessaires au bon déroulement de cette méthode qui utilise *Twitter* pour leur expérimentations :

- récupérer l'adresse MAC *WiFi* cible, et être à portée de la personne physique ;
- choisir un lieu géographique à modifier où se "teleportera" la cible au moment d'envoyer des informations via *Twitter* ;
- créer plusieurs faux point d'accès qui interagiront uniquement avec l'application cible ;
- au moment de *tweeter*, l'application va sonder les réseaux *WiFi* a portée, et va interroger un *WPS* pour avoir une position géographique. Le résultat affiché sera notre nouvelle position géographique précédemment choisie, car les réseaux *WiFi* sondés seront les faux points d'accès préalablement crées ;
- *Twitter* et d'autres applications similaire permettent grâce à son *API*, de récupérer tous les *tweets* selon des critères géographiques. Il suffira d'utiliser cet *API* pour filtrer les messages envoyés à la position géographique que nous avons choisie et de faire le lien entre l'adresse MAC *WiFi* et le profil *Twitter* de cette personne.

Tao et al. se sont intéressés aux méthodes par falsification de l'adresse MAC, une attaque difficile à détecter. Les travaux les plus récents utilisent principalement comme solution le traçage des numéros de séquences. Cependant, cette méthode présente des limites et est inutilisable avec des cartes réseaux qui ne suit pas le modèle des numéros de séquence proposé par le standard 802.11. De plus, les attaquants peuvent forger eux-même les numéros de séquence qui rendent leur attaques intraçables. Ces auteurs ont donc proposé *WISE GUARD (Wireless Security Guard)* pour détecter les périphériques qui falsifieraient leur adresse MAC. Leur méthode intègre trois techniques : le traçage des numéros de séquences, l'empreinte générée par l'utilisation d'un système d'exploitation et le traçage et la génération d'empreinte selon la puissance du signal reçu (RSS) [69].

Une possible solution pour protéger ses informations est l'utilisation du *geofencing* [70]. Cette méthode permet à un utilisateur d'activer son *WiFi* uniquement dans des zones de

confiances (*e.g.* à la maison ou au travail par exemple). Elle permet d'éliminer les attaques malveillantes qui voudraient capturer des informations privées dans des places publiques ou autre lieux. Cependant, cette méthode ne fonctionne pas lorsqu'un attaquant suit physiquement la cible ou lorsqu'il a accès aux endroits de confiance de la cible rendant la protection des informations par le *geofencing* obsolète.

1.2.3 Suivi des utilisateurs via le *WiFi*

Le suivi des utilisateurs au travers des sites webs ou des applications mobiles est devenu une pratique courante dans le monde digital. Cette pratique qui analyse l'activité des utilisateurs s'est récemment étendu dans le monde physique où les technologies sans fil et les vidéos peuvent dorénavant détecter, reconnaître et catégoriser de manière précise les activités humaines [71–73].

Dans le cas des réseaux sans fil et plus précisément du *WiFi*, la collecte de données peut se faire grâce à un analyseur qui décode les informations des *probes requests* émises par les *smartphones* pour en extraire un identifiant [74] . Ce dernier va ensuite être utilisé pour détecter la présence d'un individu et d'estimer sa mobilité.

Les systèmes de suivi des utilisateurs sont désormais déployés dans beaucoup de centres commerciaux [72], dans les systèmes de transport urbains et les autoroutes.

Cependant, la collecte des ces identifiants représente une information privée des utilisateurs. Malgré les précautions mises en place pour assurer la sécurité de ces données, la vie privée des utilisateurs est toujours menacée [75–77]. La menace est aggravée car le *Physical Tracking* rencontre un fort succès et les méthodes utilisées sont peu connus du grand public qui n'est pas conscient du fait qu'ils peuvent être *trackés* à tout moment par leur interface *WiFi*, y compris lorsqu'ils ne l'utilisent pas.

1.2.4 Contre mesures pour les informations liées à la vie privée des utilisateurs

Plusieurs travaux ont étudié le problème concernant les informations privées des utilisateurs. Ils visent soit à empêcher le suivi de la mobilité d'un utilisateur sans son consentement, soit à mettre en place des méthodes de suivi respectueuse de la vie privée.

1.2.4.1 Filtre de Bloom

Le filtre de Bloom représente une structure de donnée compacte pour représenter des ensembles. Cette structure probabiliste, permet avec certitude de savoir si un élément

est absent d'un ensemble et avec une certaine probabilité que l'élément peut être présent dans l'ensemble.

Dans le cas d'applications proposant des services basée sur la position géographique des utilisateurs, Paolo Palmieri et al. ont proposé une extension du filtre de Bloom qui intègre le domaine spatial et la position géographique [78, 79]. Leur solution permet de conserver la position exacte de l'utilisateur privé, tout en permettant au fournisseur de service d'apprendre quand l'utilisateur est proche de points d'intérêt spécifiques ou dans des zones prédéfinies.

1.2.4.2 Mixage de zones

La concept de mixage de zones a été proposé par Beresfort et al. dans le but de définir des zones appelées "zones mixées", où la position géographique de tous les utilisateurs à l'intérieur d'une zone ne vont pas être divulguées. Cette méthode repose sur le fait de n'envoyer que la position de la zone elle-même à l'entité collectant les données et non la position précise d'un utilisateur [80]. Cette technique peut être utilisé dans le contexte d'applications qui utilisent la position géographique d'un utilisateur pour lui proposer différents services. Palanisamy et Liu ont implémenté *Mobimix* en appliquant ce concept pour le réseau routier [81].

1.2.4.3 Anonymat basé sur *k*

La k-anonymité est un concept bien connu permettant la protection des informations privées. Cette méthode garantit que la cible est indiscernable parmi un ensemble de k utilisateurs.

Gruteser et Grunwald ont utilisé ce concept dans le domaine lié à la position géographique des utilisateurs [82]. En effet, l'idée derrière cette approche est d'envoyer à un client une zone "flou" de là où il se trouve contenant sa positon mais aussi des *k-1* autres personnes. Sa position précise est donc garder secrète. Par exemple, Bob est actuellement situé chez lui et fait une requête pour qu'on lui indique où se situe le centre de cardiologie le plus proche à un service qui va utiliser sa position géographique. Dans le cas où sa position exacte est traitée sans protection, il pourrait révéler des informations privées de Bob sur sa santé. En utilisant l'anonymat basé sur un ensemble k, Bob ne pourra pas être détecté parmi les autres *k-1* utilisateurs rendant impossible l'analyse des informations entre Bob et son lieu de destination.

1.2.4.4 Service de découverte respectueux des informations privées

Il est courant que les *smartphones* envoient périodiquement des *probes requests* dans le but de trouver des points connus à portée ou des points d'accès caché comme décrit dans la sous-section 1.1.3.2. Pour rappel, ce mécanisme révèle des informations privées du *smartphone* de l'utilisateur, en particulier, la liste des réseaux auxquels il s'est déjà connecté. Pour pallier à ce problème, Lindqvist et al. ont proposé dans [83] un protocole de découverte de points d'accès respectueux des informations privées qui utilise des clés secrètes partagées entre les clients et les points d'accès. Cette méthode requiert cependant des modifications du côté du client et des points d'accès pour qu'ils puissent toujours opérer avec les réseaux actuelles sans pour autant changer l'expérience utilisateur.

De manière similaire, Jang et al. ont présenté une nouvelle architecture appelé *Tryst*, qui permet d'utiliser le service de découverte de périphériques tout en conservant la sécurité sur les données sensibles avec le partage de clés secrètes entre un client et un point d'accès [84].

1.2.4.5 Introduction d'adresse MAC *WiFi* aléatoire

L'envoi de l'adresse MAC *WiFi* en clair dans les *probes requests* représente un identifiant unique permettant de le suivi des individus sans leur accord. La collecte de ces identifiants représente une menace pour la vie privée des utilisateurs.

Les premiers travaux connus, pour pallier ce problème fut l'utilisation d'algorithme de *hashage* pour anonymiser les données envoyés dans ces trames. Cependant, dans [85], Demir et al. ont démontré que ces techniques étaient insuffisantes et inefficaces en pratique.

Très récemment, les fabricants de matériels ont pris conscience de ce fléau et ont mis en place l'utilisation d'adresses MAC aléatoires dans les trames et qui changent périodiquement. En effet, Apple l'a introduite depuis *iOS8* [86], *Windows* avec l'arrivée de *Windows 10* [87], *Android* depuis *Android* version 6.0 [88] et *Linux* avec *Linux kernel* 3.18 [89]. De ce fait, collecter et analyser le contenu des *probes requests* ne suffisent plus pour identifier et étudier la mobilité d'un utilisateur.

Dans [90], Martin et al. ont analysé toutes les techniques d'anonymisation de l'adresse MAC utilisées par les systèmes d'exploitation, les constructeurs et les types de *smartphones*. Ils ont montré que l'anonymisation des adresses MAC n'était pas universellement utilisée par les différents périphériques *WiFi*. De plus, lorsqu'un périphérique est connecté à un point d'accès, il révèle son adresse MAC réelle. De ce fait, les attaques *Karma* qui

sont des attaques actives, permettent de placer un faux point d'accès qui est configuré pour avoir le même nom de réseau d'un réseau auquel le périphérique cible se connecte automatiquement [2]. Cette technique permet d'outrepasser la protection mise en place par l'anonymisation des adresses MAC. Elle nécessite cependant que l'attaquant soit actif et cible des utilisateurs ce qui limite le nombre d'utilisateurs suivis simultanément.

De plus, Demir et al. ont montré qu'il est possible de retrouver l'adresse MAC d'origine en calculant toutes les combinaisons possibles en l'espace d'une journée avec un ordinateur de récent. Plus loin encore, à l'aide d'une base de données réelle des adresses MAC *WiFi*, les auteurs ont montrés qu'il était possible de retrouver l'adresse réelle en l'espace de quelques minutes en se basant sur la structure de cette adresse. De ce fait, il est possible d'affiner la recherche et d'utiliser une attaque par dictionnaire pour retrouver l'adresse MAC d'origine en ciblant les OUI des vendeurs célèbres en priorité (*Apple, Samsung*) et leurs *NIC* associés [30].

Puis, Cunche et al. ont proposé une nouvelle attaqué basé sur le timing d'envoi des *probes requests* pour défaire l'utilisation d'adresses MAC aléatoires [91]. En effet, chaque *smartphone* possède un timing d'envoi de trames qui est consistent durant le temps et qui peut représenter un pseudo-identifiant unique. Les auteurs ont utilisé plusieurs métriques de distance et un algorithme permettant de regrouper des blocs de *probes requests* contenant des adresses MAC différentes mais appartenant à un même *smartphone*. Les expérimentations ont démontré que cette attaque était capable d'atteindre 75% d'efficacité.

Enfin, dans [92], Vanhoef et al. expliquent que l'utilisation des adresses MAC aléatoires n'est pas une mesure suffisante pour éviter que le traçage des utilisateurs. Ils démontrent que l'association de certaines informations contenues dans le champ IE (*Information Element*) peut représenter une empreinte unique d'un *smartphone*. Une autre attaque dans ce papier s'intéresse au standard 802.11u, communément appelé *Hotspot 2.0*. Les résultats ont montré que l'envoi de requêtes appelées *ANQP (Access Network Query Protocol)* uniquement sous *Windows* et/ou *Linux* utilisait l'adresse MAC réelle *WiFi* des *smartphones*.

1.3 Conclusion

Nous avons étudié dans cette partie les aspects techniques de la technologie du *WiFi* pour mieux comprendre les fonctionnalités qui permettaient de réaliser des méthodes de *Physical Tracking*. Nous avons vu qu'il suffit d'écouter sur un canal pour récolter les trames *WiFi*. Ces dernières possèdent dans l'en-tête une information sur la méthode de modulation utilisée pour la partie donnée. Il est donc possible de démoduler ces

trames et d'en extraire les informations. Nous avons ensuite vu que les *probes requests* contenaient deux informations personnelles d'un utilisateur : son adresse MAC *WiFi* (identifiant unique) et la liste des réseaux auquel son *smartphone* s'est déjà connecté. Ces informations sont périodiquement envoyé en clair sur le canal par les *smartphones* grâce au service de découverte actif des périphériques à portée. Dans un second temps, nous avons analysé tous les travaux scientifiques qui avaient un lien avec l'utilisation du *WiFi* et du *Physical Tracking*. Bien que l'introduction des adresse MAC aléatoires ait mis un frein aux méthodes utilisées pour le suivi de la mobilité des utilisateurs, certains articles mettent en avant que le *WiFi-based Physical Tracking* existe toujours.

Chapitre 2

classic Bluetooth

2.1 Aperçu du *classic Bluetooth*

Cette section présente la technologie du Bluetooth, plus précisément des premières versions déployées, référencée par l'appellation *classic Bluetooth*. Nous discuterons des points essentiels permettant de mieux comprendre les mécanismes et les failles utilisées pour collecter des informations personnelles et suivre la mobilité des utilisateurs. La sous-section2.1.1 présente le contexte avant de s'intéresser à la composition d'une adresse MAC *Bluetooth* qui représente un identifiant unique d'un *smartphone*. Nous étudierons les méthodes permettant de récupérer cet identifiant selon trois processus présents dans cette technologie : le processus de découverte de périphérique à portée, la phase d'appariement et la méthode communication. Un élément clé sera développé à la sous-section 2.1.6 permettant de comprendre le mécanisme de saut de canal de fréquence utilisé par des périphériques *Bluetooth* durant une communication.

2.1.1 Contexte

Le *Bluetooth*, crée par le groupe *Ericsson* en 1994, est un standard de communication qui a été développé pour répondre à un besoin important de pouvoir interconnecter des *smartphones* avec des périphériques à petite portée.

Le *Bluetooth* opère sur la bande de fréquence ISM 2.4 Ghz. Sur cette bande, 79 sous-fréquences sont allouées pour transmettre des données en utilisant le *FHSS (Frequency-Hopping Spread Spectrum)*. La modulation de la fréquence est effectuée grâce au *GFSK (Gaussian Frequency-Shift Keying)*. Elle est basée sur une architecture *master-slave* où le *master* peut communiquer et échanger des données simultanément avec 7 esclaves au

Classe	Transmission	Portée
Classe 1	100 mW (20 dBm)	100m
Classe 2	2.5 mW (4 dBm)	10m
Classe 3	1 mW (0 dBm)	1m

TABLE 2.1: Type de classes attribuée aux périphériques *Bluetooth* selon la puissance de transmission et la portée.

maximum. Les périphériques *Bluetooth* sont divisés en classe où leur critère de catégorisation se base sur la puissance de transmission et la portée du signal envoyé (2.1).

Pour pouvoir autoriser une communication entre plusieurs périphériques de différentes marques, les spécifications matériels et les méthodes de communication ne suffisent pas. Plusieurs protocoles doivent être spécifiés pour permettre d'envoyer différents types de flux (ex : audio, voix) qui pourront être interprétés par tous les périphériques. Pour pallier à ce problème, le *Bluetooth* a introduit le concept de *profil*. Les profils les plus utilisés sont *AD2P* qui est le protocole pour manipuler des données audio, *HFP* le protocole spécialisé pour les appels téléphoniques ou encore *FTP* utilisé pour l'échange de données. De ce fait, si un *smartphone* veut interagir avec un casque audio, ces deux périphériques doivent gérer le profil *AD2P* sinon ils ne pourront jamais communiquer sur ce type de donnée ensemble.

Avant que deux périphériques *Bluetooth* puisse communiquer, de nombreuses étapes préalables sont nécessaires. Tout d'abord, le processus de découverte de périphériques à portée, puis la procédure de pagination suivi du processus d'appariement qui va permettre de déclencher le début de la communication. Nous détaillerons chacune de ces étapes dans les sous-sections suivantes et analyserons les failles de sécurité utilisées pour collecter des informations personnelles des utilisateurs et de pouvoir connaître leurs déplacements.

2.1.2 Adresse MAC *Bluetooth*

L'adresse MAC *Bluetooth* constituée de 48 bits se décompose comme suit :

Non-Significant Address Part (NAP) : Les 2 premiers octets de l'adresse MAC.

Upper Address Part (UAP) : Le 3ème octet de l'adresse MAC. Combiné avec le NAP, il forme la partie Organizationally Unique Identifier (OUI) qui est assignée par l'Institute of Electrical and Electronics Engineers (IEEE).

Lower Address Part (LAP) : Les 3 derniers octets de l'adresse MAC. Il représente la partie Network Interface Controller (NIC) qui est désigné par le constructeur selon l'OUI.

2.1.3 Processus de découverte de périphériques à portée

Un périphérique *Bluetooth* possède principalement deux états : en connexion (*connection state*) et en réserve (*standby state*) et 7 sous-états. Le périphérique *master* va utiliser le sous-état de pagination (*page substate*) pour ajouter de nouveaux slaves à un *piconet*. Cependant, pour utiliser la procédure de pagination, l'horloge native et les 48 bits de l'adresse MAC du périphérique cible sont requis. Pour récupérer ces informations, le processus de découverte de périphériques à portée (ou *inquiry process)* est d'abord déclenché comme le montre la figure 2.1 (partie *inquiry process*).

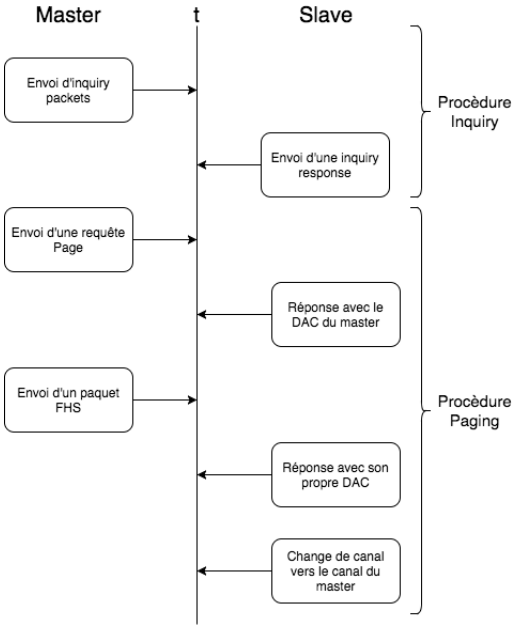


FIGURE 2.1: Description des échanges entre deux périphériques *Bluetooth* durant le processus de découverte de périphériques à portée et la procédure de pagination.

Un nœud entre dans le sous-état d'investigation (ou *inquiry substate*) pendant le processus de découverte de périphérique à portée lorsqu'il cherche à détecter la présence de d'autres nœuds pour former un *piconet*. Si un nœud est dans un sous-état de *scan* d'investigation (ou *inquiry scan substate)*, il sera à l'écoute des paquets envoyés par les nœuds en sous-état d'investigation. Un nœud à la recherche de périphérique à portée va envoyer périodiquement des paquets d'investigation appelés aussi *inquiry packets* pendant que les nœuds qui écoutent vont sonder les canaux à un rythme plus lent permettant aux deux appareils de se trouver relativement rapidement.

L'envoi des paquets et l'écoute de canaux se font à l'aide d'une séquence pseudo-aléatoire de canaux à visiter. Durant le processus de découverte, 32 canaux sont utilisés parmi les 79 au total, plus précisément les canaux de 0 à 5 et de 53 à 78. Le calcul de la séquence pseudo-aléatoire est déterminée par le *GIAC* (*General Inquiry Access Code*) qui est une valeur fixe durant tout ce processus (0x9E8B33) et par l'horloge native du périphérique.

Un périphérique en sous-état d'investigation va transmettre des *inquiry packets* sur deux séquences pseudo-aléatoires de 16 canaux que l'on nommera A et B. Au début de l'algorithme, le périphérique va choisir la séquence A. Un *slot* est composé de deux canaux. Si celui-ci est numéroté d'un nombre pair, le périphérique va émettre deux paquets sur deux canaux différents. Durant le *slot* impair suivant, celui va être à l'écoute de paquets sur ces deux mêmes canaux. Pour les deux *slots* suivants, il va choisir deux autres canaux selon la séquence dictée par A. Un périphérique fait une action (envoi ou écoute d'un paquet) sur chaque *slot* durant 625 micro-secondes, ce qui veut dire qu'en 10ms le processus aura parcouru les 16 canaux de la séquence de A. La spécification *Bluetooth* indique que ce processus doit être réitéré 256 fois pour une séquence. De ce fait, après 2.56s, le périphérique va changé de séquence et utilisera la séquence B de 16 canaux durant les 2.56 s suivantes et ainsi de suite. Ce processus est décrit par la figure 2.4. Pour les périphériques qui ne scannent pas et qui sont donc dans un sous-état de *scan* d'investigation, ces derniers vont écouter sur un canal parmi les 32 de leur séquence durant au moins 11.25ms. Ils changeront ensuite de canal tous les 1.28 ms comme indiqué par le figure 2.2.

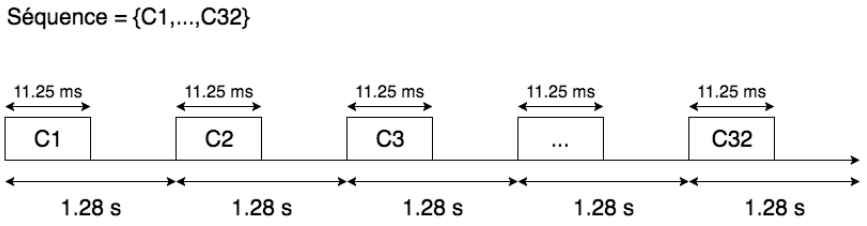


FIGURE 2.2: Écoute de canaux d'un périphérique en sous-état de *scan* d'investigation.

Après avoir reçu un paquet périphérique en sous-état d'investigation, le périphérique en sous-état de *scan* d'investigation va lui répondre par un paquet de réponse de type *FHS* (*Frequency Hopping Synchronization*) au *slot* suivant, soit après 625 micro-secondes et va entrer dans une période retrait qui va durer entre 0 et 1024 *slots*, soit entre 0 et 640ms (voir figure 2.3. Ce paquet de réponse contient l'adresse MAC de ce périphérique, la valeur de décalage de son horloge native et un code de contrôle d'erreur. Ces informations sont nécessaires pour établir un lien pour amorcer la procédure suivante : la pagination.

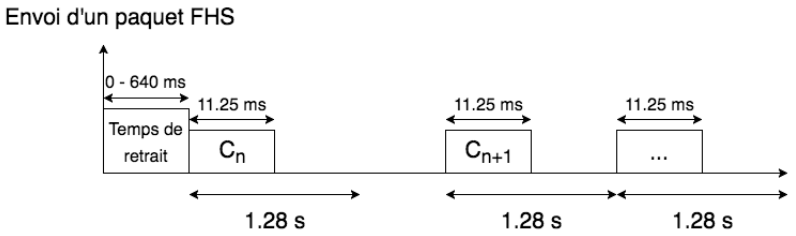


FIGURE 2.3: Après avoir envoyé un paquet de type *FHS*, le périphérique va avoir un temps de retrait qui varie entre 0 et 640ms. Ce temps d'attente permet essentiellement d'éviter les collisions.

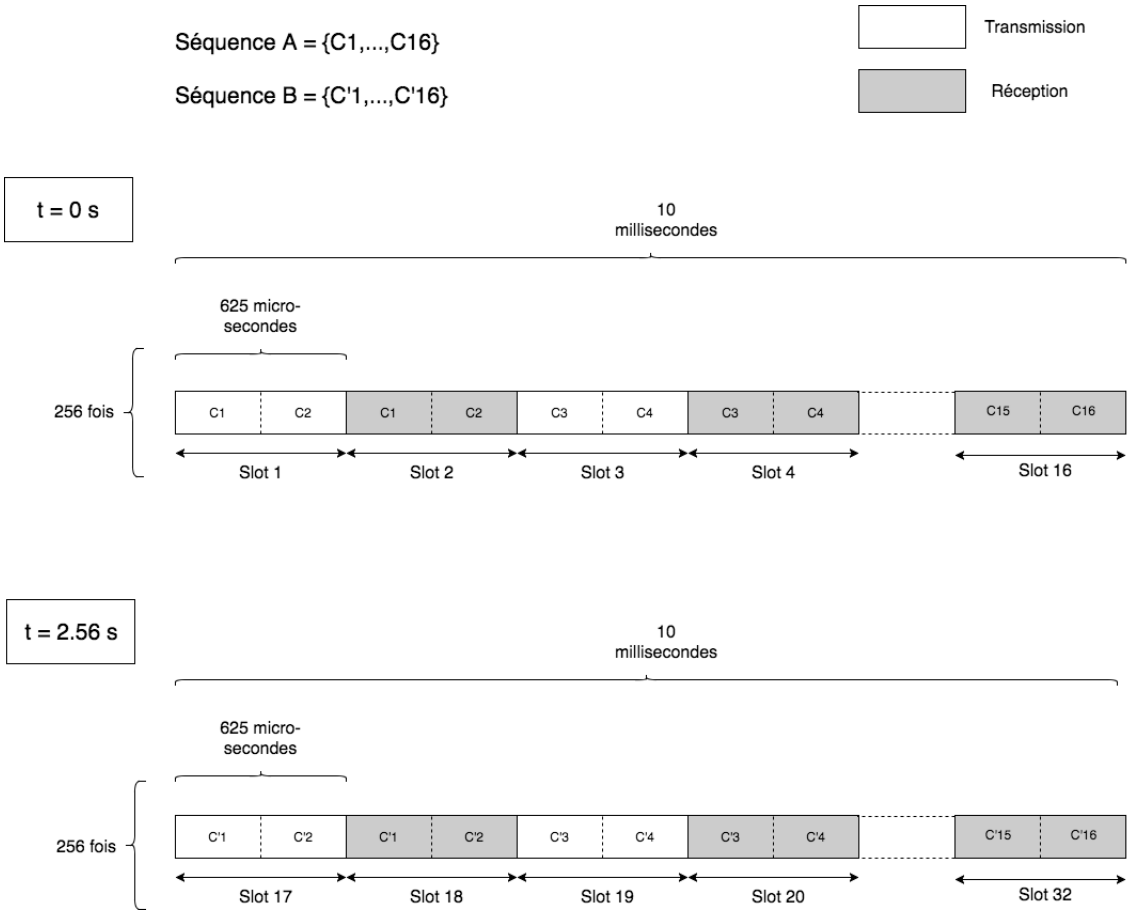


FIGURE 2.4: Transmission et écoute de canaux par *slot*, d'un périphérique *Bluetooth* en sous-état d'investigation *inquiry substate*.

Un élément important à prendre en compte est le mode qu'utilise les périphériques qui sont à l'écoute des *inquiry packets*. Il existe deux modes : le mode découvrable et non-découvrable. Un périphérique à portée va répondre aux *inquiry packets* par une *inquiry response* si et seulement si celui-ci est en mode *découvrable* comme décrit dans la figure 2.5).

Dans un contexte de *PT*, les *inquiry packets* contiennent l'adresse MAC du périphérique émetteur et sont envoyés en clair sur les canaux. De plus, en cas de réponse de la part du périphérique récepteur, ce dernier enverra un paquet envoyé en clair sur le réseau *Bluetooth* contenant des informations sensibles (notamment son adresse MAC). Il suffirait de collecter passivement ces paquets en écoutant un canal choisi arbitrairement en déclenchant périodiquement des processus de découverte pour détecter et identifier la présence de périphériques à portée en mode *découvrable*.

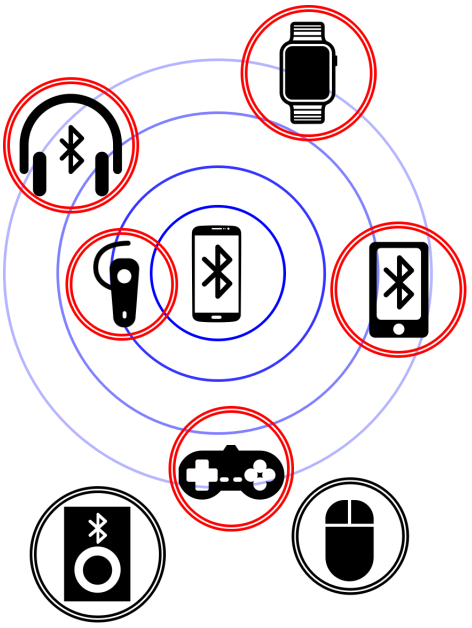


FIGURE 2.5: Le processus de découverte de périphériques à portée détecte les appareils à portée qui sont en mode *découvrable*

2.1.4 Procédure de pagination

La procédure de pagination a pour but de mettre en place tous les éléments nécessaires avant la phase de communication. Avant que celle-ci soit initiée, chaque périphérique doit connaître l'adresse MAC de l'autre.

Après réception d'un paquet de réponse en lien avec un *inquiry packet* envoyé, le master de la communication envoie une demande de pagination à l'aide d'une *paging request* (voir la figure 2.1, partie *procédure paging*). Le périphérique qui sondait les canaux (le slave) va accepter cette demande en renvoyant un paquet avec le DAC (*Device Access Code*) du master calculé grâce à son adresse MAC récupéré dans les *inquiry packets*. Après réception et vérification du DAC par le master, celui va transmettre le *FHS (Frequency Hopping Synchronization) packet* qui va contenir l'horloge native et l'adresse MAC *Bluetooth* du *master* afin d'entamer un processus de communication. Le slave réceptionne ce paquet et va calculer son propre *DAC* et le renvoyer au *master*. Après une dernière vérification de la part du *master*, le slave qui possède l'horloge native et l'adresse MAC du *master* peut maintenant générer la séquence de canaux que visite le *master* pour échanger de canal et se rendre sur le même canal que celui-ci. Les deux périphériques sont maintenant synchronisés sur la même séquence des différents canaux à visiter. La prochaine étape avant le début de la communication peut s'enclencher : le processus d'appariement.

Dans notre contexte, le paquet *FHS* contient des informations privées sur les périphériques des utilisateurs. Elles ne sont pas protégées et sont envoyées en clair. Un attaquant qui collecte de type d'informations aura accès à l'identifiant des deux périphériques qui

vont entamer une communication. De plus, il aura aussi accès à la valeur de l'horloge du périphérique *master*, qui, couplé avec son adresse MAC, permet de générer la séquence de canaux à visiter durant la communication.

2.1.5 Processus d'appariement

Lorsque deux périphériques *Bluetooth* veulent communiquer ensemble à long terme, ils peuvent être liés (appairé) ensemble. Les périphériques liés se connectent automatiquement lorsqu'ils sont à portée l'un de l'autre. La création de ce lien se fait au travers d'un processus d'appariement.

Lorsque deux périphériques s'appairent, ils s'échangent leur adresse MAC *Bluetooth* réelle, le nom ou encore les différents types de profils supportés pour généralement les stocker en mémoire. Ils s'échangent aussi une clé secrète commune, qui permettra la liaison automatique des deux périphériques et le chiffrement des paquets de données échangés.

Une phase d'authentification est requise où l'utilisateur doit valider la connexion entre les périphériques. Le mécanisme d'authentification varie et dépend généralement de l'interface des appareils.

Ce processus peut utiliser la méthode *Just Works* qui est adaptée pour les périphériques qui n'ont pas d'interface graphique (ex : un casque audio). Le clique d'un bouton de la part de l'utilisateur suffit à appairer les périphériques. Il peut aussi utiliser la méthode d'association *Numeric Comparison*, qui affiche un même code de 6 digits sur les deux périphériques. Enfin, pour les versions 2.0+, le processus d'appariement utilise la méthode *Passkey Entry*, qui demande à l'utilisateur d'entrer un code PIN pour finaliser le procédé. Ce code peut varier en longueur et en complexité à partir de quatre nombres (par exemple « 0000 » ou « 1234 ») à une chaîne alphanumérique à 16 caractères.

Une fois les périphériques appairés, ils peuvent amorcer le processus de communication et s'échanger des données.

Les failles de sécurité sont dépendantes de la méthode d'appariement utilisée. L'information sensible dans ce processus est principalement la clé secrète échangée. Celle-ci pourrait permettre de déchiffrer le contenu des paquets de données. Dans un contexte où l'adresse MAC n'a pas pu être récupérée avant, il est possible d'analyser le contenu des paquets pour en extraire un identifiant unique permettant de suivre la mobilité de l'utilisateur du périphérique ciblé.

2.1.6 Communication

Lorsque deux ou plusieurs périphériques *Bluetooth* communiquent, ils utilisent le *Frequency Hopping Spread Spectrum (FHSS)* une méthode de saut de fréquence qui consiste à changer de canal périodiquement de manière synchrone. Plus précisément, contrairement aux procédures d'investigation et de pagination, les périphériques du *piconet* vont sur la totalité des canaux (79) mise à disposition. Ces derniers vont changer de canal tous les 625 micro-secondes (2.6).

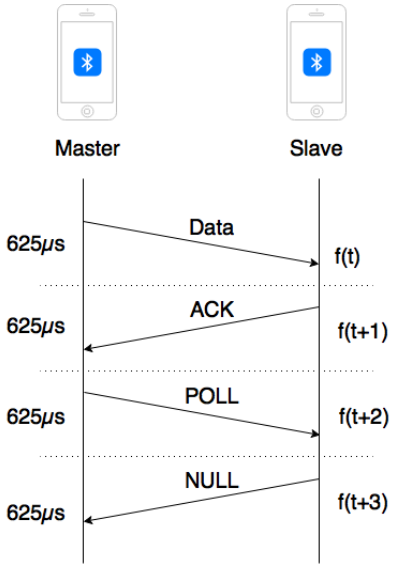


FIGURE 2.6: Processus de communication entre deux périphériques *Bluetooth*.

Dans chaque périphérique *Bluetooth*, il existe un matériel intégré appelé *Hop Selection Kernel*. Cette pièce matériel est responsable de la détermination de la séquence de sauts à visiter par les appareils durant une communication. Elle prend en paramètre l'adresse MAC *Bluetooth* et l'horloge interne du *master* du *piconet* comme le montre la figure 2.8. Dans [93], l'auteur propose une analyse complète de cet algorithme et met en avant une attaque permettant de retrouver la séquence de saut à visiter en obtenant la valeur de l'horloge et l'adresse MAC du *master* du *piconet*.

Pour rester synchronisé et corriger les dérives des horloges, les périphériques continuent à s'échanger des paquets de contrôles (paquets *NULL/POLL*) dans les cas où ils n'ont plus de paquets de données à s'échanger. La partie intéressante dans ce processus de communication est la construction des paquets de données envoyés par les périphériques du *piconet* comme le montre la figure 2.7.

L'*Access Code* est une valeur qui identifie le *piconet* et est unique durant toute la phase de communication. Elle peut être utilisée dans un contexte de *PT* afin d'identifier un utilisateur jusqu'à ce que la connexion se termine, car cette donnée n'est ni brouillée

ni chiffrée et est envoyée en clair sur les canaux. De plus, le calcul de cette valeur est basée sur le *LAP* du *master* du *piconet*, ce qui veut dire qu'il est éventuellement possible de récupérer une partie de son adresse MAC en inversant le processus de génération de l'*Access Code*.

Par suite, *l'HEC* (*Header Error Check*) qui est une somme de contrôle pour vérifier l'intégrité de l'en-tête contenue dans le paquet, est calculé avec un registre à décalage à rétroaction linéaire qui prend comme graine *l'UAP* du *master*. Cependant, cette donnée requiert une plus grande investigation pour pouvoir la récupérer d'autant plus que le *HEC* est brouillé à l'envoi du paquet.

Pour conclure, bien que l'*Access Code* peut être utile en tant qu'identifiant, ce dernier possède des limites dans le temps. Le *LAP* et l'*UAP* du maître sont des identifiants plus robustes et il est possible de les récupérer. Ainsi, même durant une communication, un utilisateur est exposé à divulguer des informations privées qui permette de suivre sa mobilité.

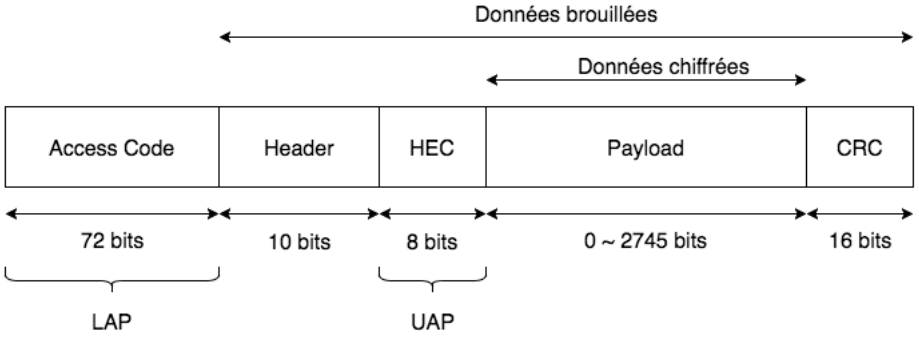


FIGURE 2.7: Composition d'un paquet de donnée *Bluetooth* échangé durant une communication.

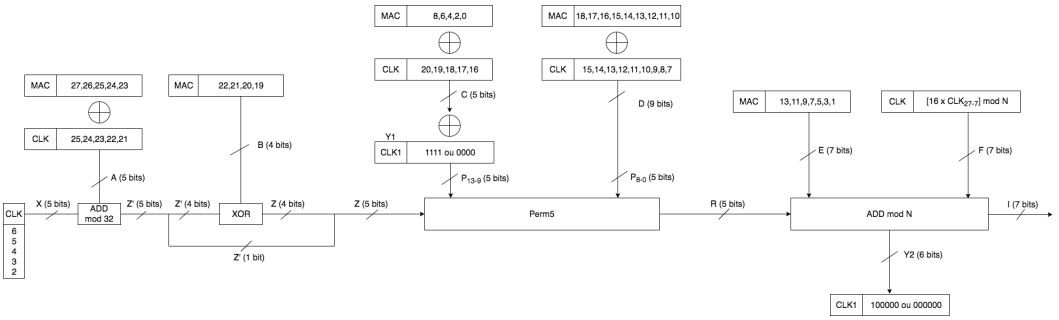


FIGURE 2.8: Algorithme de l'*Hop Selection Kernel*

2.2 État de l'art

L'état de l'art présente les différents travaux effectués en lien avec la version classique du *Bluetooth* et le *PT*. La première partie met en avant les différentes méthodes de

création d'empreinte *Bluetooth* permettant d'identifier les utilisateurs. Par la suite, nous nous intéresserons aux travaux présentant des systèmes de suivi de mobilité avec cette technologie accompagné d'expérimentations. Le principale objectif était de pallier l'inefficacité du *GPS* notamment pour la localisation d'objets à l'intérieur de bâtiments. D'autres articles vont coupler des technologies existantes (*WiFi*,*GPS*) avec le *Bluetooth* afin d'améliorer les performances en terme de temps de détection et de suivi des déplacements des utilisateurs. Enfin, la dernière partie se consacrera aux contre-mesures qui propose diverses solutions pour protéger les informations liées à la vie privée des utilisateurs.

2.2.1 Empreinte *Bluetooth*

L'empreinte *Bluetooth* représente un moyen d'identifier de façon unique une entité. Cette sous-section présente différentes méthodes de création d'empreintes basée principalement sur le *RSSI*, la collecte de l'adresse MAC ou encore la dérive de l'horloge.

2.2.1.1 *RSSI*

Le *Received Signal Strength Indication* ou *RSSI* est une mesure de la puissance en réception d'un signal. Dans le but d'établir une empreinte, l'idée est d'utiliser cette valeur comme une mesure de l'atténuation du signal. Cependant, il n'existe pas de relation déterministe entre la distance et la valeur du *RSSI*. Il y a des tendances qualitatives [94], où la relation peut varier selon différents facteurs[95].

Les méthodes d'empreinte avec le *RRSI* consiste en deux phases :

- phase de collecte de données
- phase d'estimation de la position

La première étape consiste à créer une base de donnée en récoltant les différentes valeurs du *RSSI* obtenues dans des positions distinctes d'une pièce.

La seconde étape est de collecter la valeur du *RSSI* émise par le périphérique d'un utilisateur qui se déplace et d'estimer sa position selon les valeurs dans la base de donnée créée en phase 1[96].

Liu et al. ont établi un état de l'art des différentes méthodes et systèmes pour le positionnement à l'intérieur de bâtiment. Ils ont montré que la valeur du *RSSI* peut être affectée par diffraction, réflexion, et par dispersion dans la propagation des ondes[97].

Rodriguez et al. ont analysé les performance de cette méthode d'empreinte avec trois différents algorithmes permettant de calculer la valeur du *RSSI* : la fonction de distribution cumulative qui utilisent directement les valeurs du *RSSI*, l'utilisation d'un classificateur de machines à vecteur de support qui va permettre de prédire la valeur du signal reçu et une analyse discriminante linéaire, qui à l'aide de variables prédictives, va conjecturer la valeur du *RSSI*. Les expérimentations ont démontrées que l'utilisation des machines à vecteur de support avait un meilleur de taux de réussite (un peu moins de 89%)[98].

2.2.1.2 Adresse MAC en mode découvrable

Dans [99–101], les auteurs décrivent les principales failles de sécurité présentes dans les premières version de cette technologie, à savoir la possibilité d'identifier une personne et de récupérer sa position géographique.

En effet, comme décrit dans la section 2.1.3, lorsqu'un périphérique a son interface *Bluetooth* active, il peut être soit en mode découvrable ou non-découvrable. S'il est en mode découvrable, l'appareil va répondre en émettant des paquets aux périphériques qui interagissent avec lui. De ce fait, la collecte de ces paquets permet d'extraire l'adresse MAC *Bluetooth* et de constituer une empreinte unique pour un périphérique.

L'article propose donc une attaque qui consiste à placer des périphériques *Bluetooth* dont la position géographique est connue qui vont interagir avec chaque appareil qui sont à portée de lui grâce aux informations remontés en sondant les canaux. En effet, avec un nombre assez conséquent de points d'écoute, il sera possible de géolocaliser la cible tout en ayant des informations sur sa mobilité.

2.2.1.3 Adresse MAC en mode non-découvrable

Lorsque un périphérique est en mode non-découvrable, il ne répond pas aux *inquiry packets* et de ce fait, ne révèle pas son identité.

Cependant, Whitehouse a proposé un nouvel outil appelé *RedFang* [102] permettant d'outrepasser cette protection sous certaines conditions. Son approche est basé sur le fait que des périphériques en mode non-découvrable répondront à des paquets dont l'adresse de destination est fixée à leur adresse. Elle est donc parfaitement valable dans les cas où l'adresse MAC de la cible est connu et où l'attaquant souhaite suivre un nombre de périphérique limité. Dans le cas où l'adresse cible est inconnue, cela laisse une opportunité de réaliser une attaque qui va essayer toutes les combinaisons possibles de l'adresse MAC cible. Cependant, cette attaque possède des limites car le temps de réponse de la cible peut varier et prend généralement plusieurs secondes. Si la cible est en mouvement, les

chances de réussite diminuent encore. Il serait possible de déclencher plusieurs processus en parallèle, mais il engendrerait un coût conséquent en terme de coût matériel, de consommation énergétique et de complexité d'implémentation.

Pour améliorer la viabilité de cette attaque, Haataja et al. ont proposé une méthode permettant de réduire le nombre de combinaisons à effectuer pour retrouver la bonne valeur de l'adresse MAC d'une cible . En effet, la structuration d'une adresse MAC (voir section 2.1.2) est composé de l'*OUI*, qui identifie le fabricant du matériel. Si cette valeur est connu, alors l'espace d'adressage diminue et induit un nombre de tests à réaliser beaucoup moins conséquents. Dans le cas échéant, il est possible d'effectuer une démarche de recherche plus appropriée en testant d'abord les valeurs pour les fabricants de matériels les plus connues (Apple, Samsung, Huawei).

Lors d'une expérimentation, 2000 adresses MAC ont été sondées en 174 minutes par un capteur, ce qui estimerait qu'une moyenne d'1 minute et 4 secondes en serait requise pour recouvrir l'adresse MAC d'un seul périphérique. Cross et al. ont proposé une méthode pour améliorer les performances de cette attaque en utilisant 79 capteurs qui sonderaient les canaux en parallèle et des paquets condensés pour améliorer le temps de réponse du périphérique cible. Les expérimentations ont montré qu'il est possible de retrouver l'adresse MAC réelle d'un périphérique en mode non-découvrable en moins de 24 heures[103].

Enfin, une technique peu commune qui ne requiert pas l'utilisation d'équipements ou l'analyse de paquets peut être utilisé pour établir une empreinte unique en récupérant l'adresse MAC *Bluetooth* : l'aspect visuel. Wright et al. ont illustré l'efficacité de cet aspect dans un supermarché où ils ont été capable de localiser des périphériques en mode non-découvrable et de déterminer leur adresse MAC réelle qui était tout simplement affiché sur une étiquette collée au périphérique. L'application de cette méthode dépend fortement du contexte mais peut être une approche à prendre en compte pour identifier et suivre la mobilité des utilisateurs[104]. Ces mêmes auteurs ont discuté d'un cas plus particulier où l'adresse MAC WiFi et Bluetooth d'un périphérique ont un lien. Dans le cas des *smartphones* de la marque Apple, les adresses MAC *WiFi* et *Bluetooth* sont assignés séquentiellement par les constructeurs, ce qui induit qu'incrémenter ou décrémenter la valeur de l'adresse MAC *WiFi* nous donne la valeur de l'adresse MAC *Bluetooth*.

2.2.1.4 Protocole de découverte de services

Le protocole de découverte de service (*Service Description Protocol (SDP)*) est utilisé dans le but d'identifier certains services aux autres périphériques. Sa principale fonction est d'indiquer la démarche à suivre pour avoir accès aux différents services proposés par

un périphérique. Chaque service proposé possède des propriétés qui peuvent être utiliser pour construire un identifiant. Herfurt et al. ont proposé *Blueprinting*, un algorithme qui va utiliser deux valeurs (*RecHandle* et *RFCOMM Channel*) de chaque service pour en calculer un *hash*, qui peut constituer une empreinte. De plus, l'utilisation de cet algorithme permet d'avoir des informations sur le nom du fabricant du matériel, le modèle du périphérique mais aussi la version de son *firmware*[105].

2.2.1.5 Champs dérivés de l'adresse MAC pendant une communication

Pour identifier un périphérique *Bluetooth*, certains articles se sont intéressés aux paquets de données échangés durant une communication. Pour pouvoir analyser ces paquets, un analyseur de paquets *Bluetooth* est requis. Les premiers équipements prévus à cet effet étaient très coûteux comme le *Cisco Spectrum Analyzer* (3000 USD) de *Cisco* ou encore le *Frontline Test Equipment* (10000 USD) et n'étaient pas adaptés pour un déploiement réel dans un scénario où plusieurs stations d'écoute collecteraient des paquets de données.

Cependant, en 2013, D.Spill et al. présente l'outil *Ubertooth* [106], un analyseur de paquets *Bluetooth* qui permet de collecter et de décortiquer les paquets de données échangés durant des connexions *Bluetooth*. Par la suite, dans [107, 108], les auteurs soulignent que la construction des paquets de données était réalisé grâce à certaines parties de l'adresse MAC *Bluetooth*. Ils démontrent donc qu'il est possible de récupérer les valeurs de l'*UAP* et du *LAP*, qui représenteraient 32 bits de l'adresse MAC. Ce sous-ensemble peut constituer une empreinte afin d'identifier les utilisateurs.

Une des contributions de cette thèse est la mise à profit du *LAP* et de l'*UAP* afin de créer une méthode passive de suivi des utilisateurs appelé *BPM* pour *Bluetooth Passive Monitoring* 4 puis d'évaluer les performances de cette méthode.

2.2.1.6 Dérive de l'horloge

Dans le cas du *Bluetooth*, la valeur de l'horloge a un rôle important, notamment durant le processus de communication où la séquence de canal à visiter est calculée en partie grâce à l'horloge interne du *master*, comme décrit dans 2.1.6. La bande passante est divisée en un ensemble de 79 canaux pour l'envoi de paquets. Le maître d'un *piconet* envoie un paquet par canal avant d'en visiter un autre 625μs plus tard.

Basée sur l'aspect temporel des sauts de canaux, Huang et al. proposent une méthode d'empreinte appelé *BlueId* qui va extraire la dérive de l'horloge du *master*. En effet, l'intervalle de temps entre chaque envoi de paquet est de $625 \times k\mu$s où k est une valeur positive. Idéalement la valeur de k serait de 1. Cependant, dû à la dérive de l'horloge,

l'intervalle observée dérive petit à petit. En mesurant la vitesse de cette dérive, *BlueId* va estimer la dérive de l'horloge du *master* qui va constituer une empreinte. Dans le cas des esclaves du *piconet*, il sera plus difficile des les identifier car ces derniers calibrent continuellement leur horloge afin de rester synchroniser avec le *master*[109].

2.2.2 *Tracking*

Haase et al. ont proposé *BlueTrack* ,[110] un système de suivi des utilisateurs grâce à la collecte passive des *inquiry packets*. Leur expérimentations ont pu identifier 5000 périphériques durant une collecte de données d'une semaine. Ils ont aussi démontré qu'au moins 1% des périphériques identifiés contenaient des informations personnelles tel que le nom réelle de l'utilisateur inscrit dans le champ *device name* des *inquiry packets*, ce qui laisse une opportunité d'associer une personne physique à un périphérique identifié.

Versichele et al. ont évalué les performances de la méthode passive de collecte de données en étudiant la mobilité des utilisateurs durant le festival de Ghent qui a accueilli plus d'un million et demi de visiteurs pendant 10 jours. Les expérimentations montrent qu'en couvrant 22 zones par des sondes *Bluetooth*, l'algorithme a analysé 152 487 trajectoires générées par le mouvement de 80 828 visiteurs, montrant que cette méthode est fiable en terme d'analyse des mouvements des utilisateurs de *smartphones* dans un environnement vaste avec cette technologie [111].

Pels et al. ont utilisé cette même méthode avec des capteurs fixes dans trois stations de train et ont pu identifier près de 4 000 périphériques uniques. Cependant, ils soulignent un certain nombre de problèmes à la localisation d'utilisateurs dans le train en raison de la vitesse de déplacement et des caractéristiques architecturales de la gare. De ce fait, ils mettent en avant que l'environnement et la mobilité des utilisateurs sont des facteurs à prendre en compte pour modéliser des systèmes de suivi performants[112].

Delafontaine et al. ont analysés les données collectées et ont extrait des modèles de mouvements similaires entre les individus détectés afin de pouvoir établir un lien dans leur comportement [113].

Dans un contexte de géolocalisation à l'intérieur de bâtiment, des auteurs présentent un système de traçage d'appareils à l'intérieur d'un bâtiment pour pallier l'inefficacité du *GPS* où ils décrivent une architecture composée de points d'accès *Bluetooth* connecté à un serveur central qui détecte le mouvement des individus grâce aux informations émises par leur interface *Bluetooth*[114, 115].

Simon et al. proposent une alternative aux méthodes de découverte de périphérique à portée à l'intérieur de bâtiment. Leur technique repose sur un système de traçage d'appareils, dont l'identité du *smartphone* cible est connu au préalable, qui utilise des demandes de connexions. En effet, si l'on connaît l'adresse MAC *Bluetooth* d'un *smartphone*, il est possible d'amorcer une connexion à l'aide des couches les plus basses du protocole qui ne requiert pas d'authentification. De ce fait, ils présentent une architecture où plusieurs points d'écoute sont disposés au sein d'un bâtiment. Périodiquement, les stations de base vont émettre une demande de connexion sur un ensemble d'adresses préalablement identifiées. Si la connexion aboutit, il en résulte que le périphérique est présent dans la zone du point d'écoute. Les expérimentations montrent que cette technique est viable dans un contexte de suivi d'utilisateurs [116].

D'autres travaux ont montré l'apport en terme de précision des méthodes de suivi des utilisateurs avec le *Bluetooth*. Dans [11, 13], ils utilisent le *WiFi* et les stations de base *GSM* pour co-localiser les utilisateurs. Cette méthode peut couvrir une grande zone géographique mais présente une imprécision conséquente sur le temps de détection des individus. De ce fait, dans [7, 11], Chaintreau et al. ont utilisé l'interface *Bluetooth* couplée à la technologie *WiFi* et aux réseaux *ad-hoc* pour inférer la présence des utilisateurs à portée. Les résultats mettent en avant que l'utilisation de la technologie *Bluetooth* apporte une précision accrue sur la mobilité des utilisateurs comparés aux méthodes précédemment citées.

Enfin, dans le but d'avoir un outil statistique sur la mobilité des utilisateurs, les auteurs dans [117] ont présenté *Deeploc* qui calcule statistiquement les lieux où des personnes se croisent fréquemment

2.2.3 Contre mesures pour les informations liées à la vies privées des utilisateurs

La méthode qui collecte les adresses MAC des périphériques en mode découvrable a longtemps été utilisé pour étudier la mobilité des utilisateurs, notamment dû à sa simplicité de déploiement. Cependant, dans l'optique d'apporter une réduction de la consommation énergétique et pour des questions de sécurité des informations issues de la vie privée des utilisateurs, les *smartphones* ne restent dorénavant que peu de temps en mode découvrable, rendant les systèmes de traçage des individus obsolètes [118].

De plus, les récentes mise à jour du *Bluetooth* ont aussi corrigé les problèmes sur les connexions sans autorisation et sur les mécanismes d'appariements dont les failles sont décrites dans [119–121].

Ainsi, les attaques sur la découverte d'adresses MAC en mode découvrable, non découvrables et sur les services *SDP* ne sont désormais que d'une efficacité très limitée. Les performances présentées dans la section *Tracking* (sec. 2.2.2) ne sont plus valides et le *Bluetooth* ne peut plus être utilisé pour faire du suivi.

La méthode basée sur les dérives d'horloges ne permet pas de discriminer parmi un grand nombre de périphériques et ne peut donc pas être utilisée pour faire du PT à large échelle.

Le développement et l'utilisation d'une méthode de PT basée sur les informations qui peuvent être obtenues lorsque deux périphériques sont connectés a donc un intérêt tout particulier.

2.3 Conclusion

Nous avons discuté dans cette partie des aspects technologiques du *classic Bluetooth* afin de mieux visualiser les différents failles des mécanismes *Bluetooth* que l'on pourrait exploiter afin de mettre en place des méthodes de PT basé sur le *Bluetooth*. Nous avons détaillé les différents processus de cette technologie au travers du processus d'investigation, de pagination, d'appariement et de communication tout en commentant les informations privées que l'on pourrait collecter. Ces informations privées sont décrites comme des empreintes qui peuvent identifier de manière unique un utilisateur.

L'état de l'art présentée, a mis en avant les travaux permettant de générer ces empreintes avec le *Bluetooth*. La principale méthode était de collecter l'adresse MAC *Bluetooth* réelle à partir du service actif de découverte de périphériques à portée. Plusieurs articles scientifiques ont utilisé cet identifiant pour présenter des systèmes de suivi de mobilité, car le processus de la collecte de l'adresse MAC *Bluetooth* était simple d'utilisation. Pour protéger les utilisateurs, la principale solution mise en oeuvre a été de limité le temps durant lequel un périphérique est découvrable. Cela a eu pour conséquence de limiter fortement les performances du *Physical Tracking* basé sur le *Bluetooth*.

Pour conclure, le lien du *classic Bluetooth* et du PT s'est principalement axé sur les processus en amont de la communication permettant de collecter l'adresse MAC *Bluetooth* des périphériques. Ces méthodes de suivi sont désormais inefficaces. Par ailleurs, peu d'informations sont disponibles sur la possibilité d'exploiter les informations qui peuvent être collectées lorsque deux périphériques sont connectés. De ce fait, nous avons étudié ce cas et avons analysé la construction des paquets échangés durant la phase de communication. C'est dans ce contexte qu'une des premières contributions de cette thèse présente *BPM Bluetooth Passive Monitoring* 4, une méthode de *Physical Tracking* passive, basée sur la collecte de données échangés durant une communication. Elle permet de suivre les

personnes lorsqu'elles sont connectées à leur kit audio, à leur véhicule ou autres objets connectés, y compris lorsqu'il n'y a pas de données échangées.

Chapitre 3

Bluetooth Low Energy

3.1 Aperçu du *Bluetooth Low Energy*

Cette section présente le *Bluetooth Low Energy*, une des dernières versions de la norme *Bluetooth* ajoutée en 2010. Bien que similaire de nom à sa version classique, cette technologie présente plusieurs différences que nous mettrons en avant. Nous étudierons par la suite l'impact positif ou négatif qu'engendrent ces modifications dans un contexte de *PT*. Nous analyserons ensuite les nouvelles fonctionnalités ajoutées à la composition de l'adresse MAC *Bluetooth*, au processus de découverte de périphériques à portée, au processus d'appariement et à la communication. Par suite, un état de l'art sera présenté pour donner un aperçu des travaux réalisés qui lient le *BLE* au *PT*.

3.1.1 Contexte

Le *Bluetooth Low Energy*, aussi appelé *BLE*, est l'une des dernières améliorations apportées à la technologie *Bluetooth*, qui a été ajoutée à la version 4.0 de la norme. Elle a été conçue dans le but de réduire fortement la consommation énergétique. Les périphériques qui utilisant ce standard, outre les *smartphones*, sont généralement des objets connectés qui peuvent opérer longtemps sur la durée avec leur petite pile ou petite batterie sans devoir les recharger ou les changer.

Elle opère dans la bande *ISM* 2.4 GHz (usage Industriel, Scientifique, Médical). Cette technologie est basée sur une radio à saut de fréquence pour réduire l'interférence et éviter les collisions. Elle utilise 40 canaux physiques espacés de 2 MHz chacun. Trois d'entre eux sont des canaux dits d'avertissement (*advertising channel*) utilisés durant la phase de découverte de périphérique à portée et tous les autres sont des canaux pour l'échange de données durant la phase de communication.

Cette technologie, dans l'optique de réduire fortement sa consommation énergétique, possède une architecture moins complexe que sa version classique bien que celle-ci garde la base d'une architecture *maître-esclave* où le maître possède un contrôle unidirectionnel sur un ou plusieurs esclaves qui exécutent les actions dictées par le maître. Cependant, le *BLE* n'autorise pas l'utilisation de *scatternet i.e.* la combinaison de plusieurs *piconets*. Cela induit qu'un périphérique *BLE* ne peut pas avoir le rôle de *master* et de *slave* en même temps. De plus, un périphérique ne peut pas avoir plus d'une connexion vers un *slave* ce qui veut dire qu'une seule connexion vers un dispositif *BLE* à la fois est autorisé, pour des raisons de réduction d'énergie.

La couche liaison de données du *BLE* a introduit un ensemble d'état qu'un périphérique *BLE* peut avoir selon ses différentes actions. Il est regroupé en deux états principaux : l'état connecté et non-connecté. L'état connecté indique qu'un périphérique est en phase de communication. Pour l'état non-connecté, un périphérique peut avoir quatre sous-états : *advertising, scanning, standby, initiating*. L'état *standby* est l'état par défaut où un périphérique BLE ne peut ni envoyer et recevoir des données. L'état *advertising* permet à un périphérique d'alerter de sa présence aux autres dispositifs à proximité. L'état *scanning* permet d'interagir avec des dispositifs *BLE* à portée en envoyant un paquet de requête pour avoir plus d'informations sur celui-ci. Enfin, l'état *initiating* indique qu'un périphérique va amorcer une demande de connexion. La figure 3.1 présente cet ensemble et quel chemin un dispositif *BLE* peut utiliser pour passer d'un état à un autre. Cette simplification du système d'état induit qu'un périphérique *BLE*, s'il est connecté, ne peut pas en même temps avertir de sa présence aux autres périphériques, car il ne peut pas être à le fois à l'état connecté et l'état d'avertissement (*advertising state*).

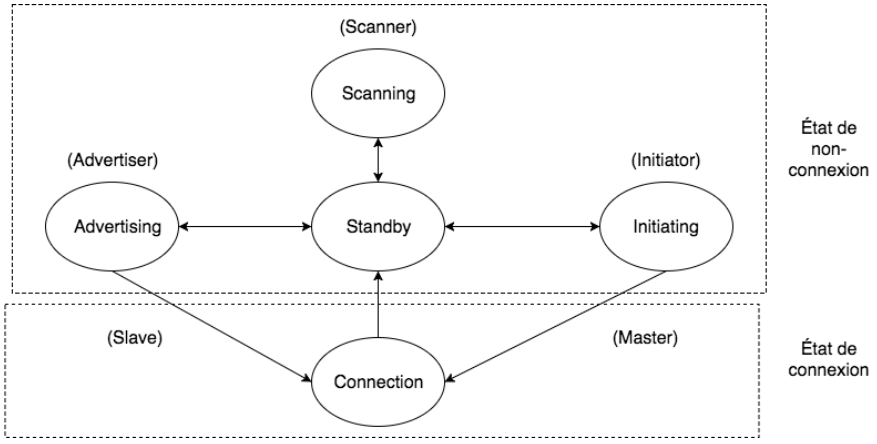


FIGURE 3.1: Les différents états qu'un périphérique *BLE* peut avoir.

De plus, plusieurs fonctionnalités ont été implémentées par rapport à la version classique, notamment sur les types d'adresses MAC utilisées, l'intégration de nouvelles clés secrètes durant le processus d'appariement, et des modifications sur la génération de séquence de

canaux à visiter durant la communication. Tous ces mécanismes et leur impact dans un contexte de *PT* seront détaillés dans les sous-sections suivantes.

Enfin, depuis ces dernières années, un réel engouement s'est créé autour de cette technologie. En effet, son utilisation plus intensive est notamment liée à l'émergence des objets connectés (les bracelets, chaussures, lunettes ou des montres) qui permettent d'apporter des services aux utilisateurs en collectant des informations diverses telles que la fréquence cardiaque, le nombre de kilomètres parcourue, la vitesse, le nombre de calories brûlée ou encore la qualité du sommeil. Le caractère ubiquitaire de ces périphériques ainsi que leur nombre nécessitait de prendre des mesures pour protéger la vie privée des utilisateurs, justifiant ainsi l'ajout des nouveaux types d'adresses.

3.1.2 Adresse MAC *BLE*

Une des principales fonctionnalités ajoutées au *BLE* est l'introduction de différentes types d'adresses MAC. Il en existe trois :

Adresse public Cette adresse représente l'adresse réelle du périphérique, assignée par les constructeurs.

Adresse aléatoire statique Cette adresse est générée aléatoirement après l'initialisation du périphérique. Tant que celui fonctionne, cette adresse ne change pas. Après remplacement de la batterie et/ou ré-initialisation du périphérique, une nouvelle adresse aléatoire statique est générée.

Adresse aléatoire privée Cette adresse est générée aléatoirement et change de façon périodique. De plus, elle est peut-être *résolvable*, ce qui veut dire que les périphériques précédemment appairés peuvent l'identifier grâce à une clé secrète échangée durant la phase d'appariement. Dans le cas contraire, l'adresse est dite *non-résolvable*.

L'adresse public est la même utilisée par la version classique, elle représente l'adresse MAC *Bluetooth* réelle du périphérique et n'apporte aucune protection pour protéger cet identifiant. Cependant, les adresses aléatoires ont été développés dans le but d'éviter que tout utilisateur puisse être tracé.

Dans un contexte de *PT*, l'utilisation d'adresses aléatoires privées résolvables pose un frein à la collecte d'identifiants et au suivi de la mobilité des utilisateurs car cette valeur change périodiquement au cours de temps. Cependant, nous verrons plus tard dans ce

manuscrit, que la plupart des dispositifs *BLE* ne tirent pas profit de ces nouvelles fonctionnalités et utilisent encore des adresses publiques ou des adresses aléatoires statiques, qui sont envoyés en clair durant la phase de découverte de périphérique à portée.

3.1.3 Phase de découverte de périphérique à portée

Comme nous l'avons décrit dans 3.1.1, un périphérique *BLE* va périodiquement alerter de sa présence en envoyant des paquets d'avertissement (aussi appelé *advertising packets*), en passant dans l'état d'*advertising*. Ce processus est appelé la phase de découverte de périphérique à portée.

En *BLE*, le format d'un paquet est unique quel que soit l'état actuel d'un périphérique. En effet, ce dernier est composé d'un préambule sur 1 octet, d'une *Access Address* sur 4 octets, des données qui varient de 2 à 39 octets et une somme de contrôle d'erreur sur 3 octets. L'adresse MAC utilisée est contenu dans la partie donnée. La valeur de l'*Access Address* est toujours la même pour le processus de découverte mais change pour chaque communication entre périphériques *BLE*.

Concernant le processus de découverte, les périphériques *BLE* utilisent uniquement trois canaux durant cette phase. Les périphériques appelés *advertiser* débutent périodiquement des *advertising events (ADV_EVT)*, dans lequel ils vont envoyer successivement des *advertising packets (ADV_PKT)* sur chacun des canaux à disposition (37, 38 et 39), Après avoir reçu un *ADV_PKT*, les appareils appelés *scanners* (généralement un ordinateur ou un *smartphone*) peuvent obtenir des informations supplémentaires en envoyant une *scan request(SCAN_REQ)*. L'*advertiser* peut lui répondre par une *scan request(SCAN_RSP)*. La figure 3.2 décrit le mécanisme d'un *advertising event* durant ce le processus de découverte de périphérique à portée. L'*advertiser* va émettre des *ADV_EVT* sur le canal 37, puis 38 et enfin 39. Sur le canal 38, il reçoit une *SCAN_REQ* d'un *scanner* auquel il renvoie une réponse de type *SCAN_RSP*.

Dans un contexte de suivi de mobilité, deux types de méthodes pour collecter des *ADV_PKT* se distinguent. La première serait de se placer sur un des trois canaux disponibles et de collecter passivement les *ADV_PKT* afin d'en extraire des informations sensibles. Dans le cas échéant, l'envoi actif de paquets de type *SCAN_REQ* serait une possible solution. La figure 3.4 résume le fonctionnement de ces deux méthodes.

Pour un attaquant, l'utilisation de méthode de collecte passive est l'idéal et nécessite peu d'effort. L'identifiant contenue dans les paquets envoyés par un périphérique est représenté par les différents types d'adresse MAC décrit en 3.1.2. Cette méthode n'est

viable uniquement si l'identifiant utilisée est une adresse MAC publique ou une adresse aléatoire statique.

Dans le cas contraire, si le périphérique *BLE* cache sa vraie adresse MAC en utilisant périodiquement des adresses aléatoires, il existe néanmoins une solution pour un attaquant de pouvoir essayer de tracer les mouvements d'un utilisateur. Il devra utiliser une méthode actif qui enverra périodiquement des *SCAN_REQ* et analysera la réponse obtenue dans les *SCAN_RSP* du périphérique cible. Généralement, ce paquet de réponse peut contenir diverses informations, tel que le niveau de batterie, la version du *firmware* utilisée ou encore le nom associé au périphérique.

Prenons l'exemple d'un individu qui possède un bracelet connecté utilisant des adresses aléatoires comme identifiant dans les *ADV_PKT*. L'attaquant qui ne peut pas tirer profit de la collecte passive des *ADV_PKT*, va envoyer activement une *SCAN_REQ* et attendre une *SCAN_RSP*. Le nom du périphérique récupéré dans le paquet de réponse peut déjà servir d'identifiant selon sa valeur. En effet, si le nom est de type "LeBraceletDeTaher", il y a peu de chances qu'un autre périphérique utilise ce même nom et l'attaquant pourrait être quasi-sûr que les adresses privées qu'il voit dans les *ADV_PKT* et le nom du périphérique contenu dans les *SCAN_RSP* appartiennent à la même personne. Dans le cas où le nom serait couramment utilisé (par exemple « Fitbit »), l'individu s'expose toujours à un risque d'être suivi. En effet, si l'attaquant a connaissance que cet individu est le seul à utiliser ce type d'objet dans une zone donnée, il pourra facilement détecter sa présence car il collectera plusieurs fois la même *SCAN_RSP* après l'envoi de ses *SCAN_REQ*. Ce cas de figure requiert des connaissances préalables de la part de l'attaquant mais l'utilisation du *PT* sous certaines hypothèses reste applicable.

Pour apporter un peu plus de sécurité face à ce type d'attaque, le *BLE* a introduit plusieurs d'*ADV_PKT* qu'un périphérique peut utiliser. Il sont au nombre de quatre : *Advertising Indication (ADV_IND)*, *Directed Advertising Indication (ADV_CONN_IND)*,*Nonconnectable Advertising Indication (ADV_NONCONN_IND)* et *Scannable Advertising Indication (ADV_SCAN_IND)* décrit par le tableau 3.3.

L'utilisation d'*ADV_PKT* de type *ADV_IND* est le cas par défaut, où le périphérique *BLE* répond à toutes les requêtes envoyés (paquet de type *SCAN_REQ* et *CONNECT_REQ*). Par exemple, un thermomètre pourrait être placé dans un bâtiment et envoie périodiquement des paquets pour indiquer sa présence (*ADV_IND*). Un *smartphone* lui envoie une *SCAN_REQ* pour lui demander s'il indique la température en *fahrenheit*. Le thermomètre lui répond favorablement avec une *SCAN_RSP*. Le *smartphone* va se connecté au thermomètre pour récupérer la valeur de la température actuelle (*CONNECT_REQ*).

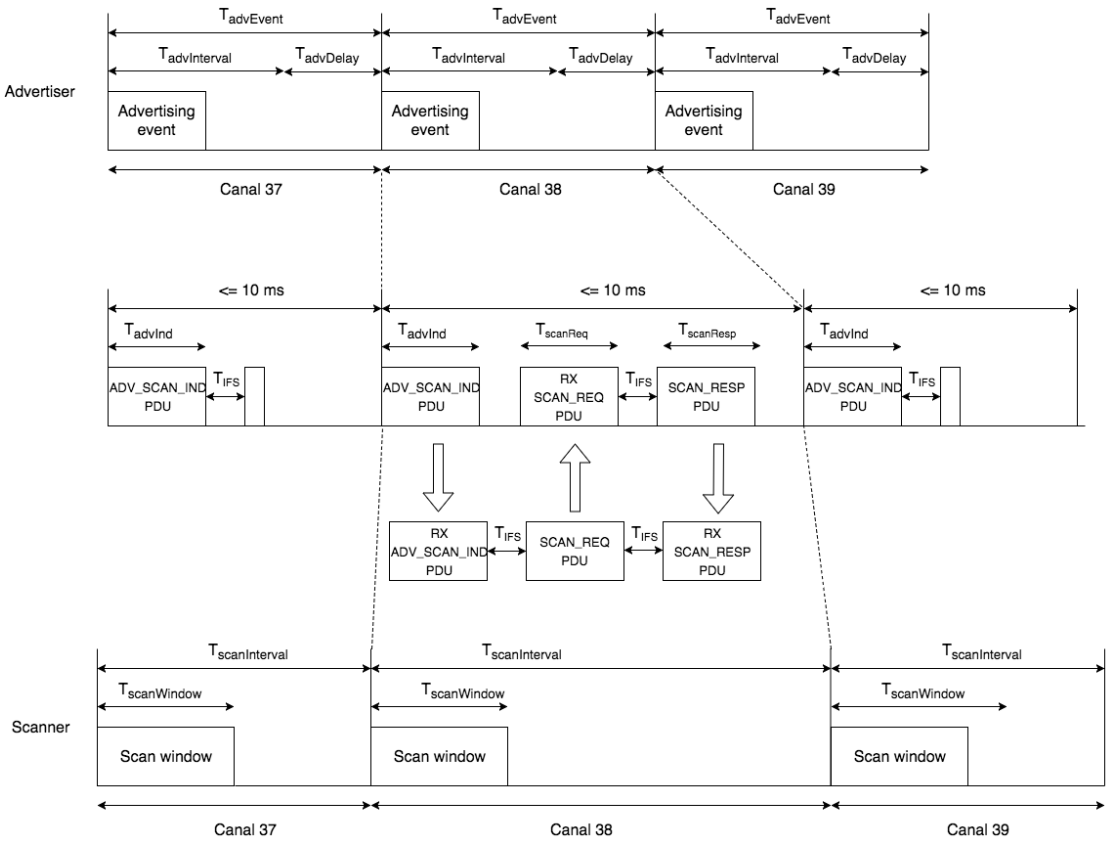


FIGURE 3.2: La description d'un *advertising event* durant la phase de découverte de périphérique à portée

Les périphériques qui utilisent des paquets de type *ADV_CONN_IND* ne répondent à aucune *SCAN_REQ* et uniquement au *CONNECT_REQ* du périphérique auquel il s'est précédemment appairé.

Quant aux paquets de type *ADV_NONCONN_IND* utilisés par les périphériques, ces derniers ne répondent à aucune requête ni aux demandes de connexion. Prenons l'exemple d'un micro-onde qui envoie des paquets pour dire que la nourriture est prête. Les utilisateurs dans sa proximité qui reçoivent ce message devront agir en conséquence.

Enfin, les paquets de type *ADV_SCAN_IND* répondent à tout paquet de type *SCAN_REQ* avec l'impossibilité d'amorcer une demande avec celui-ci. On pourrait prendre en exemple, l'utilisation d'objets connectés qui enverraient des informations sur la position où il se trouve (pièce d'un musée). Les utilisateurs pourront demander plus d'informations sur cette pièce en envoyant une *SCAN_REQ* par le biais de leur *smartphone*.

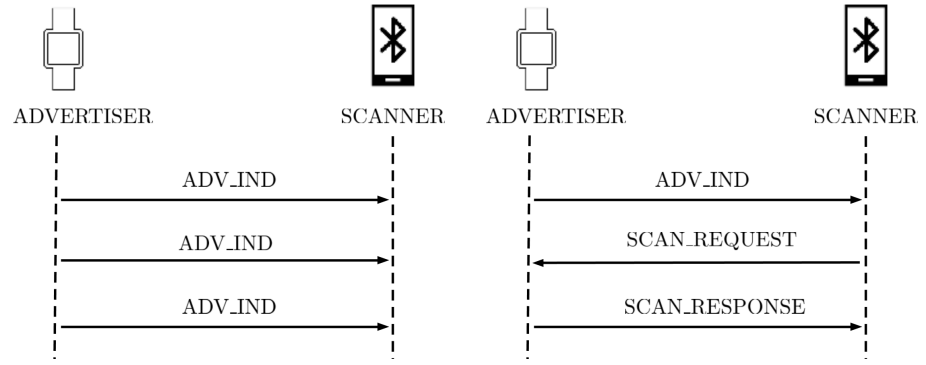
Pour revenir à notre cas de figure précédent, afin d'éviter qu'un attaquant puisse détecter la présence d'un utilisateur seul dans une zone donnée, l'utilisation des *ADV_EVT* de type *ADV_CONN_IND* avec des adressées privées aléatoires s'avère être une solution

Type d'*ADV_PKT*	Réponse au *SCAN_REQ*	Réponse au *CONNECT_REQ*
ADV_IND	Oui	Oui
ADV_DIRECT_IND	Oui	Oui (uniquement du périphérique appairé)
ADV_NONCONN_IND	Non	Non
ADV_SCAN_IND	Oui	Non

FIGURE 3.3: Interactions entre un périphérique qui envoient différents types d'*ADV_PKT* avec un *scanner* (envoi de *SCAN_REQ*) et un *advertiser* (envoi de *CONNECT_REQ*)

fiable car vu l'absence de réponse aux *SCAN_REQ* envoyé par l'attaquant, ce dernier ne sera pas en mesure de détecter activement la mobilité de l'utilisateur cible.

Contrairement à sa version classique, l'utilisation des nouvelles fonctionnalités offertes par le *BLE* améliore la protection des données liées à la vie privée d'un utilisateur et ainsi de rendre plus difficile la détection de sa présence durant le service de découverte de périphérique à portée.



(A) Processus passif quand un périphérique collecte uniquement des *ADV_PKT*.

(B) Processus actif quand un scanner envoie une *SCAN_REQ* après avoir reçu un *ADV_PKT*.

FIGURE 3.4: Processus de découverte de périphérique à portée actif et passif en *BLE*

3.1.4 Processus d'appariement

Le processus suivant le service de découverte de périphérique à portée se caractérise généralement par une demande de connexion. Le périphérique *master* va initier une connexion en envoyant au périphérique esclave un paquet de type *CONNECT_REQ*. Avant de pouvoir communiquer, un processus d'appariement est déclenché. Il se déroule principalement en trois phases. La première étape consiste à l'envoi des informations sur les capacités d'entrée et de sorties des périphériques, de la méthode d'appariement à utiliser et du type de clés secrètes à échanger. La seconde phase va générer une première clé secrète appelé *TK* (*Temporary Key*) qui va chiffrer la session courante en utilisant la méthode d'appariement la plus appropriée selon les données échangées à la phase 1. Enfin,

la dernière étape va permettre aux périphériques de s'échanger des clés de sécurités tels que la *LTK* (*Long-Term Key*) et l'*IRK* (*Identity Resolving Key*). L'*IRK* a pour but de résoudre les adresses aléatoires résolvables pour révéler l'identité réelle d'un périphérique. La *LTK* est chargée de chiffrer la connexion actuelle et les futures connexions ainsi que la re-connexion automatique entre deux périphériques *BLE*. La figure 3.5 décrit toutes les étapes de ce processus.

Dans un contexte de *PT*, les données échangées durant la troisième phase de ce processus sont des données sensibles notamment l'*IRK* qui permet d'outrepasser la sécurité apportée par l'introduction des adresses aléatoires résolvables. Cependant, pour un attaquant, la phase 2 est celle à cibler car elle est indispensable pour pouvoir déchiffrer les clés envoyés à la phase 3. Le protocole qui gère les différentes interactions du processus d'appariement se nomme *Security Manager*. Il définit des méthodes d'appariements semblable à sa version classique : *Just Works, Passkey Entry,* et *Out of Band*2.1.5. Cependant, ces méthodes ne garantissent aucune protection pour les attaques de type *passive eavesdropping i.e.* qui consiste pour un attaquant d'écouter et de collecter passivement tous les paquets échangés sur les canaux durant ce processus quelque soit la méthode d'appariement utilisée. De ce fait, il aura accès à toutes les clés secrètes échangées entre les périphériques ce qui représente un risque potentiel pour la vie privée des utilisateurs ciblés. L'utilisateur peut être aussi confronté à un autre type d'attaque : les attaques de type *MITM (Man-In-The-Middle)* aussi appelé *active eavesdropping*. Elle consiste à un attaquant de se placer entre les périphériques qui communiquent en relayant les paquets échangés sans que ces derniers ne soient au courant. Cette attaque peut permettre d'injecter tout type de paquet, d'usurper l'identité d'un périphérique légitime ou encore de forcer une fin de connexion entre ces périphériques. Néanmoins, la méthode *Passkey entry* qui va consister à afficher une clé sur l'interface d'un des périphériques et l'utilisateur de l'autre périphérique devra entrer ce mot de passe via une interface graphique. S'il y a une correspondance, le processus d'appariement est un succès. Dans ce contexte l'attaquant n'a aucune information sur la clé qui est affiché car elle ne transite pas en clair sur le canal. Il ne sera pas en mesure d'effectuer activement des modifications sur les paquets échangés. La méthode *Out of Band* possède aussi cette protection mais cette méthode s'applique généralement pour les périphériques qui possèdent des capteurs *NFC* (*Near Field Communication*), utilisés principalement pour les applications « sans contact »(paiement sans contact par exemple), et qui n'entre pas dans le *scope* des applications déployées dans notre contexte de *PT*. Comme décrit en début de cette sous-section, la méthode utilisée dépend de la capacité d'interface des périphériques qui veulent s'appairer. Dans le cas où un périphérique ne possède pas la capacité d'afficher ou d'entrer un nombre, la méthode *Just Works* sera utilisée. Cette dernière est une méthode moins sécurisée que le *Passkey entry* car aucune information n'est échangée au niveau

de l'interface graphique des deux périphériques. De ce fait, elle est sensible aux deux types d'attaques précédemment citées, le *passive et active eavesdropping*. Pour les périphériques d'anciennes générations qui ne supporterait pas la méthode *Just Works*, une solution existe via la méthode *No Security*. Comme son nom l'indique, aucun chiffrement n'est réalisé et c'est est une méthode à éviter car elle n'apporte aucune protection sur les données sensibles qui transitent entres les périphériques durant ce processus.

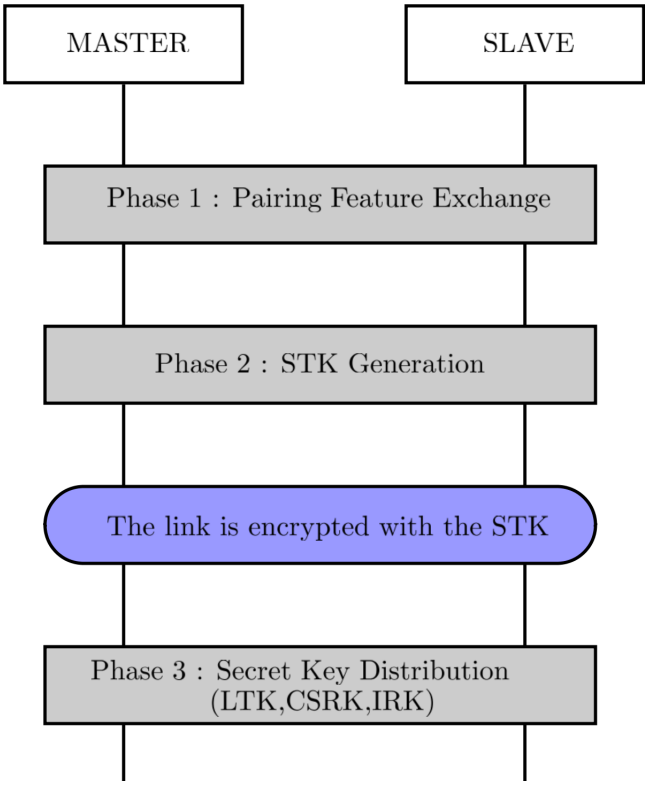


FIGURE 3.5: Le processus d'appariement entre deux périphériques *BLE*

Bien conscient qu'aucune de ces méthodes n'est viable contre une attaque de type *passive eavesdropping*, les développeurs de la norme *Bluetooth* ont apporté une possible solution, avec l'arrivée de la version 4.2 qui introduit une nouvelle méthode d'appariement utilisant l'algorithme de *Diffie-Hellman*.

Cet algorithme permet de générer une clé secrète entre une source et une destination de tel sorte que cette clé ne puisse pas être observée et échangée durant la communication. Le point principal dans ce contexte n'est pas d'échanger une clé mais de créer une même clé ensemble par les deux parties qui communiquent. Cette méthode s'avère très efficace dans le cas où un attaquant collecterait passivement les données échangées entre deux périphériques, car en analysant chacun des paquets échangés, il ne sera pas capable de retrouver la clé secrète car celle-ci n'a jamais été échangée.

3.1.5 Communication

Une fois que le processus d'appariement a été effectué, la communication entre les périphériques appairés débute. Tout comme la version classique, le processus de communication utilise une séquence de canaux à visiter entre le *master* et le *slave*. Dans le cas du *BLE*, 37 des 40 canaux sont utilisés pour la communication.

Avant le début du processus d'appariement, le *master* envoie un paquet de configuration (appelé CONNECT_REQ_PDU) au slave qui contient deux paramètres importants pour générer la séquence de canaux à visiter : le *hop Increment* et le *hop Interval*.

La valeur du *hop Interval* est utilisée pour calculer l'intervalle de temps (en ms) entre la visite de deux canaux consécutifs :

$$\Delta t = 1.25 * hopInterval$$

.

Le *hop Increment* est utilisé pour calculer le numéro du prochain canal à visiter. Le canal de départ est attribué par le master :

$$nextChannel = currentChannel + hopIncrement(modulo 37)$$

Enfin, la construction des paquets de données n'utilisent aucune parties ou sous-parties de l'adresse MAC réelle du périphérique.

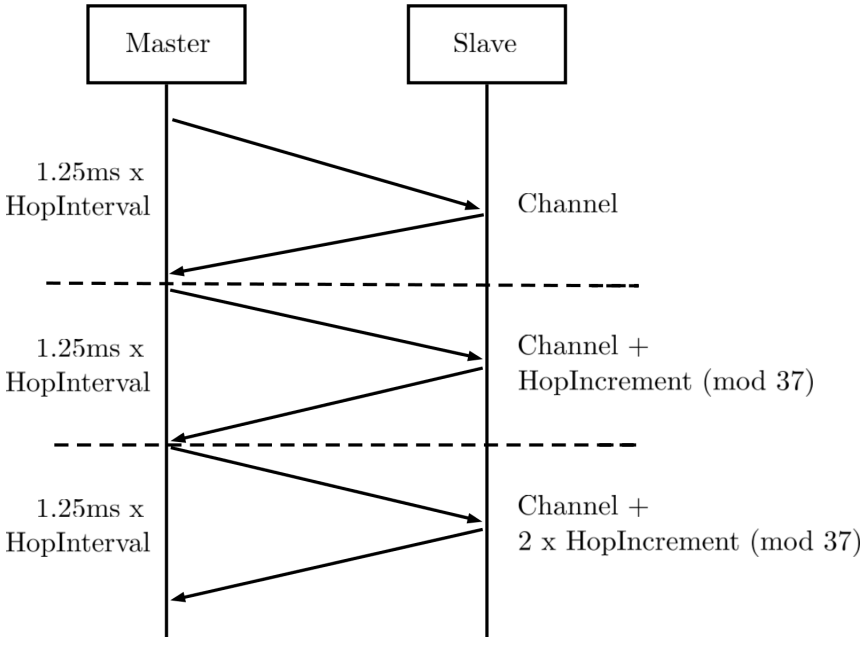


FIGURE 3.6: Le processus de communication entre deux périphériques *BLE*

La figure 3.6 décrit le processus de communication entre deux périphériques *BLE*.

Comme décrit ci-dessus, le mécanisme de communication introduit pour le *BLE* a été simplifiée par rapport aux versions précédentes. Cependant, les modifications apportées sur la construction des paquets échangés vont dans le bon sens en terme de sécurité de données privées de risque de suivi de la mobilité des utilisateurs dans notre contexte de *PT*. En effet, ni l'*Access Address* ni la séquence de saut n'est dérivé de l'adresse MAC et il impossible en décortiquant les paquets échangés durant une communication de retrouver l'adresse MAC d'origine du *master*. Pour un attaquant, la phase de communication représente un frein à la collecte d'informations privées. Ce dernier peut essayer de récupérer les valeurs de l'*Hop Increment* et l'*Hop Interval* afin de recalculer la séquence des canaux à visiter. Il peut ensuite entamer une phase de brouillage des canaux pour forcer une déconnexion entre les périphériques qui communiquent afin de pouvoir collecter les *ADV_PKT* émis par l'un des deux périphériques lorsqu'il repassera en phase de découverte de périphérique à portée suite à sa déconnexion. Pour conclure, le risque zéro n'existe pas mais l'attaque précédemment décrite, en plus d'être une attaque active, dépend fortement du contexte et nécessite une proximité physique de l'attaquant vers sa cible. Nous constatons que le *BLE* prend en considération les risques liées à la vie privée des utilisateurs et agit en conséquence. Nous voyons que la modification apportée au processus de communication en *BLE* par rapport à sa version classique engendre beaucoup plus d'efforts de la part d'un attaquant pour atteindre son objectif.

3.2 État de l'art

Après avoir étudié l'ensemble des fonctionnalités apportées au *BLE*, cette section présente les différents travaux effectués en lien avec la version du *Bluetooth Low Energy* et le PT. La première partie s'intéresse aux différentes méthodes de création d'empreinte *BLE* permettant d'identifier les utilisateurs au travers des adresses MAC *Bluetooth*, du processus de communication ou en utilisant le concept d'*honeypot*. La seconde partie se focalise sur les systèmes de suivi de mobilité qui mettent en avant leur fonctionnement et un retour sur expérimentations pour en évaluer les performances. Enfin, la dernière partie se consacrera aux contre-mesures qui propose diverses solutions pour protéger les informations liées à la vie privée des utilisateurs avec la technologie du *BLE*.

3.2.1 Empreinte *BLE*

Semblable au *WiFi* et au textitclassic Bluetooth, l'empreinte *BLE* identifie une entité tout en garantissant son unicité. Cette sous-section détaille les différentes méthodes de

création d'empreintes en s'appuyant sur les failles des mécanismes *BLE* qui interagissent avec des informations sensibles et privées des utilisateurs .

3.2.1.1 Adresse MAC publique et statique

La norme *BLE* a introduit l'utilisation de plusieurs types d'adresses MAC à utiliser lors de la phase de découverte de périphériques à portée comme décrit dans 3.1.2.

Dans [122], Das et al. ont analysé le type d'adresse MAC utilisées par plusieurs objets connectés de marques différentes. Il en résulte que la majorité de ces périphériques utilisent soit leur adresse mac réelle soit une adresse aléatoire statique. De ce fait, il est possible de collecter ces adresses qui constitueront une empreinte associée à un périphérique.

3.2.1.2 Communication

Ryan et al. ont montré qu'il existait des failles de sécurité avec le *Bluetooth Low Energy* pendant une communication. L'objectif de leur attaque est de permettre de suivre une connexion *Bluetooth* entre deux périphériques cibles. En effet, lorsque des appareils *BLE* s'échangent des données, ils utilisent une séquence de nombre représentant le numéro du canal à visiter de manière synchrone [123].

Cet article montre qu'il est possible de récupérer les valeurs du *Hop_ Increment* et du*Hop_ Interval*, indispensables pour réussir à recouvrir la séquence de saut de canaux.

La formule utilisée pour calculer le prochain canal à visiter parmi les 37 est décrite de la façon suivante : $NextChannel = currentChannel + HopIncrement(mod 37)$ Grâce à l'analyseur de paquets *Ubertooth*, la première étape consiste à collecter les paquets de données et à récupérer la valeur de l'*Access Address*, qui identifie de manière unique une communication.

Ensuite, un cycle de sauts se termine toutes les 37 x 1.25 x *Hop_ Interval* millisecondes. De ce fait, pour récupérer cette dernière valeur, il suffit de rester sur un canal fixe et de calculer le temps écoulé (nommé Δt) entre la visite consécutif de ce canal par les périphériques cibles. Il en découle la formule suivante : $Hop\_Interval = \frac{\Delta t}{37x1.25(ms)}$ Cette valeur indique combien de temps restent les périphériques sur un canal avant de passer au suivant.

La valeur du *Hop_ Increment* est calculée en regardant le temps inter-arrivée des paquets entre deux canaux d'indice 0 et 1. Si le temps entre ces deux canaux est égale à la valeur du *Hop_ Interval* précédemment calculée, alors ces canaux se suivent selon la séquence de

sauts. On réitère ces opérations jusqu'à trouver la séquence complète et ainsi récupérer la valeur du *Hop_ Increment*.

De plus, il est possible d'outrepasser les méthodes de chiffrement et de récupérer les valeurs des clés secrètes échangées grâce à l'outil *Crackle* [124] si l'on réussi à collecter tous les paquets échangés durant la phase d'appariement entre périphériques *BLE*.

Enfin, pour pouvoir identifier un périphérique *BLE* et lui associer une empreinte unique, les étapes suivantes sont à suivre :

1. Collecter les paquets de données et récupérer la valeur de l'identifiant de la communication (valeur de l'*Access Address*)
2. Retrouver les valeurs du *Hop_ Increment* et *Hop_ Interval*
3. Injecter un paquet qui force la fin de la communication et oblige une nouvelle phase d'appariement.
4. Lorsque les deux périphériques vont se reconnecter, un paquet nommé *CONNECT_ REQ PDU* sera envoyé en clair et contiendra entre-autre les informations suivantes :
   - L'adresse MAC réelle de l'origine et de la source
   - La valeur du *Hop_ Increment*
   - La valeur du *Hop_ Interval*
5. Lors du processus d'appariement, collecter tous les paquets et les différentes clés secrètes échangées.

3.2.1.3 Concept d'*honeypot*

L'utilisation d'adresses aléatoires ne permet plus d'établir une empreinte unique pour un périphérique. Les informations sensibles sont uniquement échangées durant la phase d'appariement. De ce fait, un moyen de récupérer l'adresse MAC réelle d'un périphérique est de l'inciter à amorcer un processus d'appariement avec un objet connecté. Le concept d'*honeypot* consiste à placer à des points stratégiques des sondes *BLE* qui proposent un service aux utilisateurs lorsqu'ils se connectent à lui. Avant la phase de communication, une phase d'appariement sera établie et permettra de récolter l'identifiant du périphérique qui se connecte en plus des clés secrètes (IRK,LTK).

3.2.2 Tracking et localisation indoor

Au niveau applicatif, les principaux travaux sur le *BLE* se sont axés sur les méthodes de géolocalisation à l'intérieur de bâtiments. En effet, dû a la faible précision du GPS et du fait que la technologie Bluetooth (avant la 4.0) ne permet plus aux périphériques de rester en mode découvrable très longtemps, les objets connectés en *BLE* apportent une solution viable pour pallier à ces restrictions.

L'établissement d'un système de traçage se déroule en deux phases : le calcul du positionnement des périphériques cibles suivi de la collecte d'identifiants. La première phase permettra de géolocaliser les périphériques tandis que la seconde permettra d'identifier les utilisateurs. Ces deux phases couplées permettront de suivre la mobilité de chaque utilisateur.

Deux principales méthodes de calcul de la position d'une entité à l'intérieur de bâtiment se distinguent avec le *BLE* : la méthode par empreinte du *RSSI* et la triangulation. Les objets connectés sont placés à des endroits fixes au sein d'une zone. Ils sont en mode *beacon*, ce qui veut dire qu'ils émettent périodiquement des paquets. Plusieurs objets connectés utilisant ce mode accompagnés de leur protocole existent sur le marché comme par exemple *iBeacon d'Apple*, *Eddystone* de *Google* ou encore *AltBeacon de Radius Network*.

La méthode par empreinte, collecte dans une base de donnée, les valeurs du *RSSI* émise par chaque objet connecté fixe ainsi que leur identifiant. Le couple (*RSSI*, identifiant) définie une empreinte permettant de distinguer une position précise d'une zone. Ces valeurs sont stables au cours du temps étant donné que la fréquence d'émission et la mobilité de l'objet connecté ne changent pas. Pour calculer la position d'un périphérique cible, les valeurs du *RSSI* des objets connectés fixes et de ce périphérique sont comparés et convertis en distance à l'aide d'un modèle de propagation[125]. Différentes méthodes existent pour calculer la distance en fonction du *RSSI* et sont discutées dans [97].

Enfin, Chai et al. propose un algorithme basé sur la triangulation des valeurs du *RSSI* récoltées. La valeur du *RSSI* obtenue est modifiée en supprimant les valeurs aberrantes et en calculant une moyenne selon la mobilité du périphérique cible. Ensuite, la distance est estimée selon la valeur du *RSSI* précédemment calculée en utilisant un filtre de *Kalman*. Enfin, la triangulation va calculé la position de l'objet connecté cible en utilisant les distances pré-calculées par 3 nœuds ou plus. Les expérimentations montrent qu'il y a une marge d'erreur de positionnement comprise entre 0.2m et 0.4m.

La seconde phase, après calcul de la position du périphérique cible, est d'envoyer vers un serveur central son identifiant. Ce dernier peut-être la valeur de l'adresse MAC (si

c'est une adresse publique ou statique), un *UUID* (identifiant unique généralement sur 128 bits) ou encore un identifiant générée par une application.

3.2.3 *Crowd localization*

Il existe des systèmes de traçage qui ne nécessitent pas d'infrastructure. En effet, le *crowd-localization* est une méthode qui permet de récolter des données grâce aux informations envoyées par les utilisateurs d'une communauté. Cette pratique s'utilise notamment pour retrouver ses objets perdus où plusieurs acteurs ont développé leur propres objets connectés et applications (TrackR, Wistiki, Chipolo). Périodiquement, l'application va sondé les canaux *BLE* et va envoyé l'identifiant de l'objet connecté de l'utilisateur et de ceux à portée de lui accompagné de sa position géographique actuelle. Le serveur peut retracer la mobilité de ses clients grâce à ces informations remontées. Cette pratique induit notamment des problèmes relatifs à la vie privée des utilisateurs. Un des contributions de cette thèse est la proposition d'architecture de *crowd-localization* préservant les informations privées et personnelles des utilisateurs.

3.2.4 Contre mesures pour les informations liées à la vies privées des utilisateurs

L'émergence des objets connectés a eu un impact non-négligeable sur l'utilisation plus intensive du *Bluetooth Low Energy*. La norme 4.0 a introduit les d'adresses mac aléatoire pour apporter plus de sécurité et éviter le suivi de la mobilité des utilisateurs. De plus, contrairement au *classic Bluetooth*, lors d'une communication, les champs de contrôle des paquets ne sont pas dérivés de l'adresse MAC *Bluetooth* réelle du *smartphone* ou de l'objet connecté *BLE*[126]. Cependant, beaucoup de constructeurs n'appliquent pas cette nouvelle norme et implémentent encore des adresses publiques ou statiques à l'objet connecté.

Pour pallier ce problème, Snader et al. propose une première approche avec *CryptoCop*, un protocole de confidentialité et de chiffrement permettant de protéger les informations liées à la vie privée, de sécuriser l'échange de données entre deux périphériques et éviter d'être tracé par un individu malveillant.

Pour le chiffrement des données contenues dans les paquets envoyés, ce protocole utilise une clé de chiffrement matériel envoyé par USB en connectant l'objet connecté à un ordinateur. Cette clé n'est donc pas envoyé entre le *smartphone* et l'objet connecté via le processus d'appariement. De plus, pour se protéger d'un attaquant qui aurait un accès

physique à l'objet, il a été prévu une rotation périodique de la clé de chiffrement en supprimant la clé de chiffrement matériel initialement envoyée.

Pour empêcher à un attaquant d'identifier un utilisateur au travers des paquets envoyés, *CryptoCop* va utiliser une adresse aléatoire à chaque paquet envoyé par l'objet connecté. La génération de cette adresse est effectuée par le *hash* du tuple de donnée composé d'une valeur secrète (connu par le périphérique appairé) et d'un compteur de synchronisation appelé *CTR* situé dans chaque en-tête d'un paquet. Pour apporter plus de protection, le compteur peut être précédé d'une valeur arbitraire pour éviter la répétition d'une même valeur de *CTR*. Enfin, un attaquant peut malgré tout détecter la présence d'un utilisateur en analysant la fréquence de transmission des paquets de données. Pour pallier ce problème, *CryptoCop* génère des paquets à une fréquence constante, mais ajoute une instabilité aléatoire aux temps de transmission pour empêcher un attaquant d'identifier ce périphérique.

3.3 Conclusion

Dans cette partie, nous avons mis en avant les principales fonctionnalités de la technologie du *Bluetooth Low Energy*. Bien que similaire à sa version classique, cette dernière présente des différences technologiques qui ont un impact dans un contexte de *PT*. L'ajout des adresses aléatoires (résolvables ou non) et la modification de l'*Access Address* et de la séquence de saut du *FHSS* qui ne dépendent plus de l'adresse MAC du périphérique renforcent significativement la robustesse du *BLE* face au suivi des utilisateurs. Cependant, certaines failles existent et peuvent être exploitées dans un contexte de suivi de mobilité bien qu'il requiert plus d'efforts de la part d'un attaquant si le périphérique cible utilise toutes ces nouvelles fonctionnalités mises à sa disposition. En effet, beaucoup d'appareils ou d'objets connectés compatibles au *BLE* ne prennent pas en compte ces innovations et exposent les utilisateurs à des risques sur sa vie privée bien supérieurs à ceux du *classic Bluetooth*. Une des contributions de cette thèse, est de lister (de manière non-exhaustive) les caractéristiques de périphériques *BLE* et de leurs fonctionnalités associées, pour se rendre compte de la proportion d'objets sensibles à ces attaques.

Deuxième partie

Contributions

Chapitre 4

BPM : *Bluetooth Passive Monitoring*

4.1 Introduction

Ce chapitre, comme les suivants s'intéresse aux méthodes de suivi des utilisateurs, que ce soit à leur insu ou pour la collection de traces de contacts ou de mobilité. Les possibilités pour la collecte de traces de contact et/ou de mobilités sont nombreuses, diversifiées et complémentaires.

Traces issues de la position géographique inférée par l'*OS* : Les applications installées sur les *smartphones* des utilisateurs peuvent périodiquement relever leur position géographique, laquelle est fourni par l'*OS* du *smartphone*. Lorsque que deux utilisateurs sont suffisamment proches, on peut considérer qu'ils sont en contact. Si la position remontée par l'*OS* est fournie par le *GPS*, la consommation énergétique requise est malheureusement significative, impactant l'usage normal du *smartphone*. De plus, elle est soumise à des imprécisions, en particulier lorsque les utilisateurs sont situés en milieu urbain. Si l'objectif est de suivre de manière grossière le parcours de l'utilisateur, cela peut être satisfaisant. A l'inverse, si les traces ont pour objectif de déterminer les opportunités de contacts (*e.g.* pour en dériver les performances d'un réseau *DTN*), les traces inférées sur la base de position *GPS* sont peu réalistes. Lorsque la position du *smartphone* est inférée par l'*OS* via le WPS (*Wifi Positioning System*), le surcoût de consommation énergétique est négligeable mais la précision obtenue demeure tributaire de la finesse du maillage de points d'accès *WiFi*. Celui-ci demeure en deux dimensions, ce qui créé des contacts artificiels entre des utilisateurs qui seraient à des étages différents (ou points

d'intérêts) et qui ne sont pas à portée. En sus du problème de précisions et de consommation, un autre inconvénient majeur est de convaincre les utilisateurs d'installer une application qui suivra leurs déplacement, ce qui limite la taille des populations suivies.

Traces inférées par co-localisation *WiFi/GSM* : Une autre méthode utilisée par les opérateurs de téléphonie mobile ou les propriétaires de larges réseaux *WiFi* (*e.g.* campus, centres commerciaux) consiste à considérer que deux utilisateurs sont en contact lorsqu'ils sont connectés en même temps à la même station de base ou au même point d'accès. Cette méthode souffre d'un évident problème de précision, en particulier dans le cas des réseaux mobiles ou les cellules sont de grande taille et ou deux utilisateurs peuvent être déclarés comme en contacts alors qu'ils sont très éloignés et largement hors de portée. Il en va de même pour la co-localisation sur les points d'accès *WiFi* qui souffre en plus d'un problème de précision temporelle, la présence d'un utilisateur étant généralement inférée sur la base de son authentification ou de l'obtention du bail *DHCP*. Celui-ci n'est pas fait immédiatement lors de l'arrivée de l'utilisateur et le moment du départ est difficile à déterminer car le bail *DHCP* est rarement relâché et l'utilisateur ne se dé-associe pas explicitement. Cette méthode présente par contre l'avantage de pouvoir inclure un très grand nombre d'utilisateurs, a priori sans limite de durée.

Traces collectées par périphérique dédiés : Pour inférer les opportunités de contacts d'une population mobile, l'option la plus précise reste de recruter des participants et de leur confier des sondes dédiés qui relèvent périodiquement les autres utilisateurs à portée ainsi que la qualité de signal vers ces participants. Cette méthode présente cependant des inconvénients évidents en termes de coûts et, par conséquent, du nombre participants qui peut être suivi, affectant ainsi la représentativité des traces collectées.

Traces de contact issues des découvertes active du *classic Bluetooth* : Le service actif de découverte de périphériques à portée du Bluetooth a longtemps été considéré pour sa facilité de déploiement et sa précision. Une application sur le *smartphone* des participants scannent périodiquement les périphériques à portée. Les équipements *Bluetooth* à proximité qui sont en mode découvrables répondent avec leur adresse MAC et nom de périphérique. Les *smartphones* sont parfois partiellement remplacés par des périphériques dédiés tels que les *intel-imote2*. Dans [6, 7, 11], les auteurs ont utilisé l'interface *Bluetooth* pour détecter la présence d'utilisateurs à proximité et en dériver des traces de contacts pour l'évaluation des *DTNs* et autres *MANETs*. Ces traces sont ensuite utilisées pour la simulation ou pour dériver des modèles de mobilité synthétiques. Comparé aux traces issues des points d'accès *WiFi* ou des stations de base *GSM*, le *Bluetooth* a eu l'avantage d'améliorer significativement la précision des traces. Les noeuds qui effectuent

les scans actifs doivent être équipés d'applications dédiées, ceux qui sont simplement détectés doivent simplement être en mode découvrable, ce qui était le cas par défaut pour peu que l'interface *Bluetooth* soit activée. La nécessité de recruter des utilisateurs pour effectuer les scans limite le nombre de périphériques inclus dans les traces. De plus les contacts entre les noeuds qui sont découvrables (mais ne font pas de scans) ne sont pas visibles dans les traces. La consommation énergétique générée par les procédures périodiques de découverte de périphériques à portée avec le *Bluetooth* affecte la batterie des *smartphones* des utilisateurs et limite leur volonté de participer à la collecte des traces de mobilité dans la durée. Cette méthode de collecte de traces de mobilité n'était certes pas dénuée de d'inconvénients mais permettait un compromis intéressant entre la précision des traces et la facilité de collecte, le *Bluetooth* étant une interface largement disponible sur les périphériques des utilisateurs. Par ailleurs, en raison de la consommation d'énergie excessive et de problèmes de confidentialité, les fabricants ont changé leurs politiques et les *smartphones* sont maintenant moins susceptibles de répondre aux scans *Bluetooth* [118]. Cette méthode de collecte de traces est donc désormais peu performante, que ce soit pour la collecte de traces de contacts entre des volontaires ou le suivi d'utilisateurs à leur insu.

Nous pouvons constater qu'il n'existe pas de méthode de collecte parfaite. Le suivi de la mobilité et des contacts entre utilisateurs, fut-il pour une campagne ponctuelle ou effectué en continu, doit donc reposer sur un ensemble de technologies complémentaires en sélectionnant celles qu'il est opportun d'utiliser. Ainsi, les méthodes basées sur la détection des contacts entre des sondes de mesure et les utilisateurs (recrutés ou non) sont très utiles. En sus de la détection via la découverte active des périphériques *Bluetooth*, le *WiFi* permet des mesures similaire via les *probes requests* de la procédure de découverte des points d'accès (voir la le chapitre 1.1.3.2). Cette méthode avait l'avantage supplémentaire d'être passive (*i.e.* le point de mesure se contente d'écouter les canaux *WiFi* et n'a pas besoin d'interagir avec le noeud détecté) et de pouvoir suivre la quasi totalité des périphériques *WiFi*, c'est-à-dire l'ensemble des *smartphones* et *laptop*. Ces propriétés la rendaient particulièrement intéressante pour le suivi de masse à l'insu des utilisateurs. Cependant, pour les raisons de vie privée détaillées dans 1.1.3.2, les adresses MAC publiques n'apparaissent plus dans les *probes requests* et cette méthode est également rendu in-effective.

Dans ce chapitre nous proposons *BPM (Bluetooth Passive Monitoring)*, un outil de collecte de traces de contacts passif qui permet de reconsidérer le *Bluetooth* comme une technologie efficace pour suivre la mobilité des utilisateurs. Dans BPM, des noeuds écoutent un ou plusieurs canaux *Bluetooth* et calculent une empreinte pour chaque périphérique détecté.

BPM présente l'avantage d'être une méthode de détection passive et peut donc suivre les utilisateurs à leur insu sans être détectée. L'*AC (Access Code)* sur lequel elle repose est envoyé avec une modulation robuste et permet une grande distance de détection qui peut être couplée avec le *RSSI* pour estimer la proximité et la qualité du signal. Elle requiert pour l'instant d'utiliser l'*Ubertooth*, un périphérique d'écoute dédié pour chaque point de mesure dont le coût avoisine les 40€. Un autre désavantage est qu'elle ne peut suivre que les périphériques qui ont une connexion *Bluetooth*. C'est par exemple le cas des utilisateurs connecté à leur véhicule, à leur kit oreillette, des dispositifs médicaux ou autres objets connectés en *classic Bluetooth* qui ont une connexion établie quasiment en permanence, même si il n'y a pas de données échangées. Les *smartphones* qui disposent du *Bluetooth* mais qui ne sont pas connectés, ne font pas de scans et ne sont pas découvrables et ne peuvent être détectés par BPM.

Pour évaluer l'adéquation de BPM, nous avons montré les empreintes obtenues permettait d'identifier une population large avec un faible taux de collision. Nous avons effectué des expérimentations réelles sur des périphériques équipés d'une interface *classic Bluetooth*. Nous avons mesuré le temps de détection d'un utilisateur actif comme une fonction comprenant les paramètres suivants : la taille de l'empreinte, le mode *Bluetooth* utilisé, le type de profil *Bluetooth* et la distance entre la sonde *Bluetooth* et le périphérique cible.

La suite de ce chapitre est organisée de la manière suivante. La section 4.2, présente la méthode d'obtention du *LAP* et de l'*UAP*, les deux parties de l'empreinte collectée. La section 4.3 décrit les expérimentations que nous avons menées pour caractériser ces empreintes, aussi bien en de temps d'obtention, en sensibilité aux modes d'utilisation, aux types de périphériques ou encore la distance entre les participants. La section 4.4 présente les résultats pour le *LAP* tandis que la section 4.4 présente le résultat pour l'*UAP*. La section 4.6 présente ensuite comment améliorer les performances et enfin la section 4.7 conclut le chapitre sur BPM et discute des recommandation, aussi bien pour la collecte de traces que pour se protéger du risque de suivi.

4.2 Empreinte *Bluetooth*

Cette section traite de l'empreinte construite par BPM afin d'identifier un utilisateur. Cette méthode fonctionne uniquement lorsque des périphériques *Bluetooth* sont en communication. Dans ce contexte, nous étudions uniquement la technologie du *classic Bluetooth* dont la section 2 présente les détails.

4.2.1 Sonder les canaux *Bluetooth*

Écouter un canal spécifique avec l'espoir de collecter un sous-ensemble de paquets échangés n'est pas aisé en *Bluetooth* car aucune interface *classic Bluetooth* ne le permet et modifier le *firmware* n'est pas une option abordable. Les analyseurs *Bluetooth* coûtent plusieurs milliers d'euro et la communauté recherche se tourne généralement vers des radio programmables tels que l'*USRP* couplé au logiciel *GNU radio* dont le coût dépasse néanmoins les 2000€.

De cette observation, les auteurs dans [107] ont récemment développé l'*Ubertooth*, une plate-forme logicielle et matérielle *open-source* qui a pour but de sniffer les trames *classic Bluetooth* et *Bluetooth Low Energy*. Ce périphérique peu coûteux peut être configuré pour rester sur un canal donné et récolter le flux binaire *GFSK* démodulé. La libraire `libtbb` permet de rechercher des trames *Bluetooth* à l'intérieur de ce flux.

Si l'on considère le service de découverte de réseau du *WiFi* sur les *smartphones* d'anciennes générations (voir 1.1.3.2), il est très simple d'obtenir les 48 bits de l'adresse MAC qui font office d'identifiant. Excepté dans le cas de quelques fabricants peu fiables, une adresse MAC devrait être unique. L'obtention d'une trame générée par un périphérique est suffisante pour identifier de manière unique un *smartphone* et son propriétaire. Le cas du *Bluetooth* n'est pas aussi simple pour plusieurs raisons. Tout d'abord, la plate-forme *Ubertooth* permet de remonter les trames collectées sur un seul canal *Bluetooth* alors que le périphérique du *piconet* peut utiliser l'un des 79 canaux. Deuxièmement, l'en-tête d'un paquet de donnée *Bluetooth* contient un champ d'adresse sur 3 bits qui identifie seulement les périphériques à l'intérieur d'un *piconet*. Ce champ n'est pas lié à l'adresse MAC réelle. Pour obtenir un identifiant unique ou un quasi-identifiant des trames *Bluetooth*, il est nécessaire d'examiner plus en profondeur la structure des paquets de données *Bluetooth* échangées durant une communication.

4.2.2 Inférer les 24 bits de l'identifiant

L'Access Code (AC) *i.e.* les 72 premiers bits d'un paquet de donnée *Bluetooth* décrit à la figure 4.1 est utilisé pour identifier à quel *piconet* appartient la trame. L'AC est dérivé à partir du *LAP* du *master* et une séquence pseudo-aléatoire connue. Le résultat est encodé avec un code expurgé (64,30) sur lequel est ajouté 4 bits du préambule et du trailer. Toutes ces opérations peuvent être inversées pour obtenir le *LAP* du *master* qui représente les 24 derniers bits de l'adresse MAC. Pour rappel, comme décrit dans 2.1.2, les 48 bits de l'adresse MAC *Bluetooth* sont divisés en deux parties principales. Les 24 premiers bits représentent l'*OUI (Organisational Unique Identifier)* qui est assigné par

l'IEEE. Les 24 derniers bits, appelé le *NAP*, est spécifique au *NIC (Network Interface Controller)*, c'est-à-dire qu'il est prévu de ne pas avoir plus d'un *NIC* avec pour un même couple *NAP* et *OUI*.

Outre le fait que les 24 bits du *LAP* peuvent supporter 16777216 *NIC*, les différents périphériques vus dans les mesures peuvent venir de constructeurs différents conduisant ainsi à une probabilité de collision non négligeable.

Bien que le *LAP* ne soit pas un identifiant unique, il devrait cependant suffire pour la plupart des scénarios de mesures de mobilité. De plus, même s'il y a plusieurs périphériques qui partagent le même *LAP*, celui-ci peut être facilement détecté car leur mobilité ne sera pas plausible, par exemple, ils seront détectés simultanément à différents endroits ou se déplaceront trop rapidement.

4.2.3 Inférer 8 bits de plus : l' *UAP*

Comme décrit dans 4.1, l' *LAP* n'est pas le seule partie de l'adresse MAC qui est impliquée dans une trame de donnée. Les 8 derniers bits de l'en-tête est le *HEC*, une somme de contrôle d'erreur calculée en utilisant un registre à décalage à rétroaction linéaire initialisé avec les 8 bits de l'*UAP* du *master*. Si l'en-tête est reçu sans erreur, les 10 premiers bits de l'en-tête et les 8 bits de l'*HEC* sont suffisants pour obtenir la valeur de l'*UAP*.

Malheureusement, avant la transmission, l'en-tête et les données sont brouillées, c'est-à-dire qu'un *XOR* est affecté avec une séquence générée par un registre à décalage à rétroaction linéaire initialisé avec les 6 derniers bits de l'horloge interne du *master*, appelé *CLK1-6*. Le *master* partage la valeur son horloge au début de la connexion et ce dernier est incrémenté toutes les 625 micro-secondes. Chaque horloge évolue au même rythme, de ce fait, la valeur du *CLK1-6* va être connue par tous les membres d'un *piconet* mais pas par notre sonde *Bluetooth*. De ce fait, nous avons 64 valeurs potentielles de *CLK1-6* à partir desquelles nous pouvons dériver 64 contenues de l'en-tête et par la suite, 64 valeurs candidates de l'*UAP*.

Les auteurs de [108] ont proposé deux façons de réduire le nombre de valeurs candidates. Si la trame de donnée contient un *CRC* à la fin de la section de données, il est possible de calculer le *CRC* pour toutes les 64 valeurs candidates du *CLK1-6*. Celui qui correspond permettra de récupérer l'*UAP*.

Cependant, dans certains profils *Bluetooth*, le *CRC* n'est pas obligatoire. De plus, la modulation utilisée pour l'en-tête n'est pas nécessairement la même que pour la *payload* et le *CRC*. Si elles sont différentes, le *CRC* ne sera d'aucune utilité car *Ubertooth* ne peut pas décoder deux modulations simultanément. Dans ce cas, il est possible d'éliminer la

valeur candidate du *CLK1-6* si elles ne donnent pas un contenu d'en-tête plausible. Il est très improbable qu'une seule trame soit suffisante pour réduire le nombre de candidats pour la valeur de l'*UAP* à un. Comme *Ubertooth* fournit l'intervalle de temps qui sépare deux trames, nous pouvons utiliser les trames reçues à la suite pour réduire davantage le nombre de candidats qui atteindront éventuellement un.

Notons que si l'en-tête reçu contient des erreurs, cette méthode peut conduire à avoir zéro candidat pour l'*UAP*. S'il n'y a pas assez de trafic, le temps requis pour trouver un *UAP* devient grand et l'horloge de l'*Ubertooth* peut s'éloigner de celle du *master*. De ce fait, cela pourrait entraîner la découverte la découverte d'un *UAP* dont la valeur n'est pas la bonne.

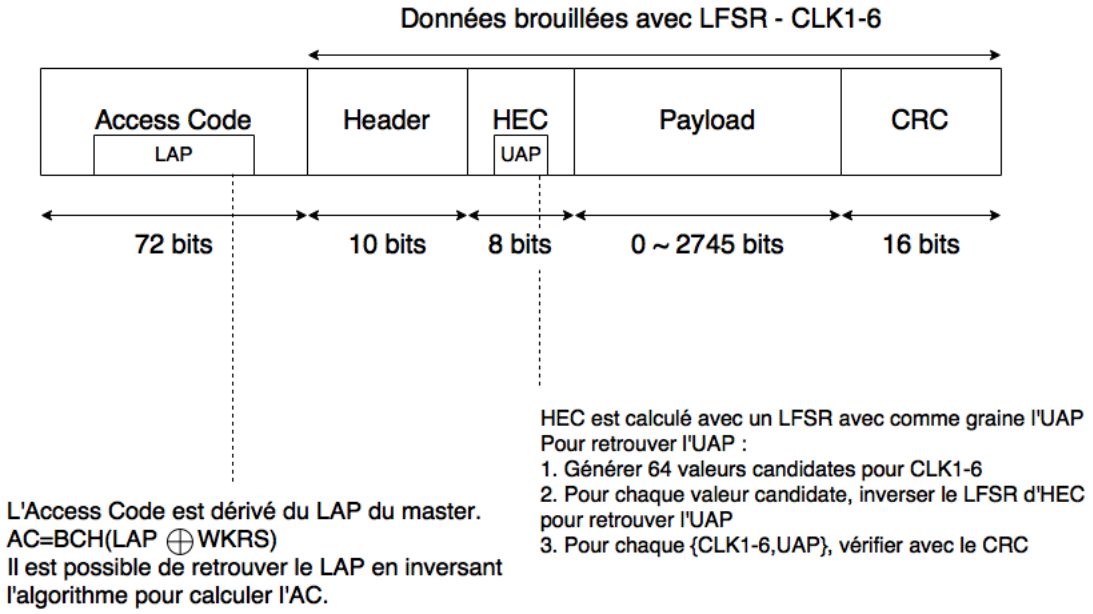


FIGURE 4.1: Informations pertinentes d'une trame de donnée *Bluetooth* : L'*Access Code* permet de récupérer le *LAP* ; L'en-tête qui n'est pas brouillé permet de récupérer la valeur de l'*UAP* ; *CLK1-6* permet de supprimer le brouillage de l'en-tête, il peut être inférer après avoir collecter plusieurs trames.

4.3 Expérimentations

Cette sous-section présente les résultats des expérimentations effectuées afin d'évaluer les performances de BPM dans un contexte de collection de trace de mobilité. Nous nous sommes intéressés à deux analyses de mesures : l'intervalle de temps entre deux *LAP* consécutifs collectés (appelé temps inter-*LAP*) et le temps nécessaire pour récolter l'*UAP*.

4.3.1 Détails des expérimentations

Pour mener à bien nos expérimentations et pour comprendre l'impact des différents périphériques utilisés, nous avons utilisé trois différents *smartphones* : un *iphone 6* sous iOS9, un *samsung GTI90000* sous android 2.2 et *Wiko* sous android 4.2. Nous avons évalué ces *smartphones* avec les trois principaux profils *Bluetooth* (voir 2.1.1). Le profil *HFP* est utilisé pour les appels téléphoniques et est associé avec une oreillette *Bluetooth*. Le profil *AD2P_ AVRCP* est utilisé pour le contenu audio et est associé avec un haut-parleur *Bluetooth*. Le profil *FTP* est principale utilisé pour les transferts de fichiers et les *smartphones* étaient associés avec un ordinateur portable. Les *smartphones* étaient tous *master* de leur connexion *Bluetooth*.

Pour tous les *smartphones* et profils, nous avons effectué cinq captures de cinq minutes pendant que l'application générait du trafic sur le lien. Nous faisons référence à ces mesures avec le mot-clé *Trafic*.

Pour *HFP* et *AD2P_ AVRCP*, nous avons effectué deux captures d'une heure chacune où les *smartphones* étaient connectés mais ne généraient aucun trafic. Pour *AD2P_ AVRCP*, le haut-parleur était connecté mais aucun contenu audio n'était joué. Pour *HFP*, l'oreillette était connectée, mais il n'y avait pas d'appel entrant et sortant pendant toute la durée de l'expérimentation. Notons qu'il n'était pas possible de connecter deux périphériques avec le profil *FTP* tout en ne générant aucun trafic car la connexion *Bluetooth* se termine aussitôt que le fichier a été envoyé à la destination. Nous faisons référence à ces mesures avec le mot-clé *No-Trafic*.

Pour chacun de scénarios (*smartphone*, profil, trafic), le nombre total de trames capturées pendant les cinq captures est resté aux alentours de 10% de leurs valeurs moyennes.

Pour configurer nos sondes pour la collecte de données, nous avons utilisé l'outil *Ubertooth*. Ce dernier a été configuré pour collecter les trames de données sur le canal 0 et stocker le flux binaire démodulé par *GFSK*. Nous avons analysé ultérieurement les trames démodulées et nous avons recherché le *LAP* dans ces différentes trames. Nous avons aussi appliqué l'algorithme pour trouver l' *UAP* d'un *piconet* à partir du début de la capture. Si nous avons un résultat positif, nous sauvegardons le temps requis, réinitialisons l'algorithme et nous recommençons le processus avec le paquet suivant trouvé dans le fichier de capture. Il nous permet d'avoir assez d'échantillon pour tracer la distribution.

Toutes nos expérimentations ont été effectuées dans notre laboratoire. D'autres périphériques utilisaient le bande *ISM* 2.4 GHz durant nos captures. Seul l' *UAP* et le *LAP* qui correspondaient à nos *smartphones* ont été considérés dans nos résultats.

	Wiko	Samsung	iphone6
AD2P avec trafic	0.38	1.96	0.85
HFP avec trafic	0.37	0.46	0.54
FTP/OBEX avec trafic	0.39	0.52	NA
HFP sans trafic	23.48	37.13	13.86
AD2P sans trafic	26.67	34.93	35.17

TABLE 4.1: 95-ème percentile du temps inter-*LAP* en secondes pour chaque *smartphone* avec différents profils en trafic et sans trafic.

4.4 Temps de détection du *LAP* d'un *smartphone*

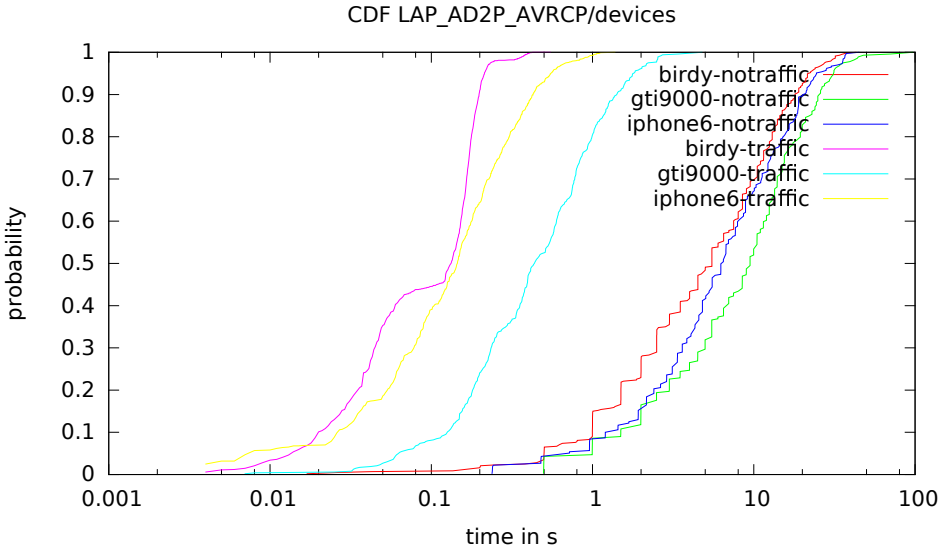
Les figures 4.2 et 4.3 montrent la distribution du temps inter-arrivée d'un *LAP* mesurée par notre sonde. Sur chaque sous-figure, le tracé distingue le cas où il y a (resp. il n'y a pas) de trafic échangé entre un *smartphone* pour les trois profils (*AD2P_ AVRCP, HFP, FTP*). Pour faciliter la lecture la 4.2 n'affiche qu'un profil par figure tandis que la figure 4.3 affiche un *smartphone* par figure. Les deux figures sont issues des mêmes données. L'axe des x a été fixé selon une échelle logarithmique.

Le tableau 4.1 montre le 95-ième percentile du temps inter-arrivée d'un *LAP* des *smartphones* pour chaque profil.

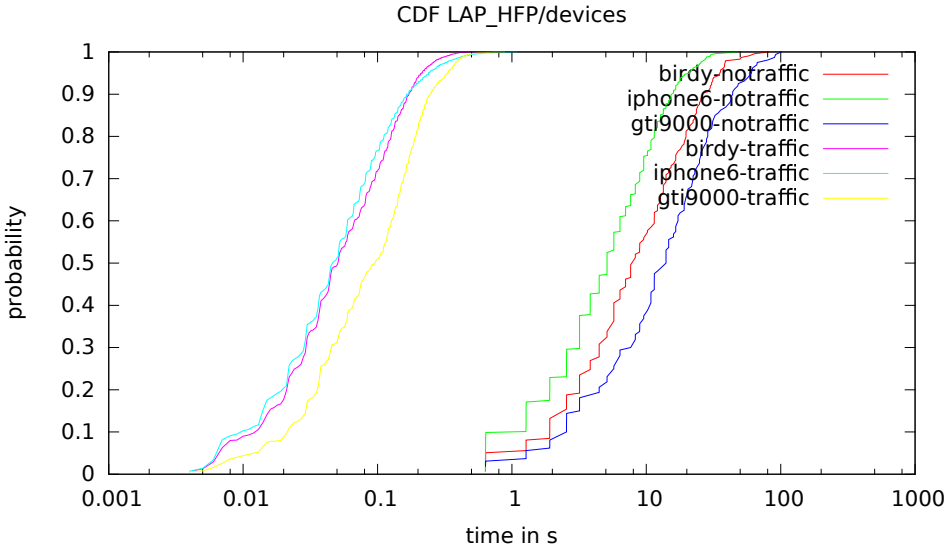
Nous pouvons voir que pour chaque *smartphone*, les résultats différent concernant le temps inter-arrivée du *LAP*. Tout d'abord, nous pouvons voir que le type de *smartphone* a un impact sur le temps inter-arrivée du *LAP*. Par exemple, pour un même profil (*AD2P_ AVRCP*), lorsqu'il y a du trafic, le Wiko a une probabilité de 95% de collecter un *LAP* en moins de 0.38 seconde. Il a de meilleurs résultats que le gti9000 (1.96 secondes) et l'iphone 6 (0.85 seconde). Par la suite, nous constatons que le type de profil utilisé a aussi un impact sur le temps inter-arrivée du *LAP*. En effet, pour un même *smartphone* (gti9000), lorsqu'il y a du trafic, nous collectons un *LAP* en moins de 0.46 seconde avec une probabilité de 95% avec le profil *HFP* qui a de meilleures résultats que le profil *AD2P_ AVRCP* (1.96 secondes) et le profil *FTP* (0.52 seconde). De plus, selon la norme *Bluetooth*[126], le service actif de découverte de périphériques à portée a besoin au minimum de 10.24 secondes pour récupérer l'adresse MAC entière. Par conséquent, en trafic, notre méthode passive peut collecter un *LAP* en moins d'une seconde pour la plupart des *smartphones* avec une probabilité de 95%.

Nous pouvons conclure que notre méthode améliore jusqu'à 10 fois les performances du service actif de découverte de périphériques à portée.

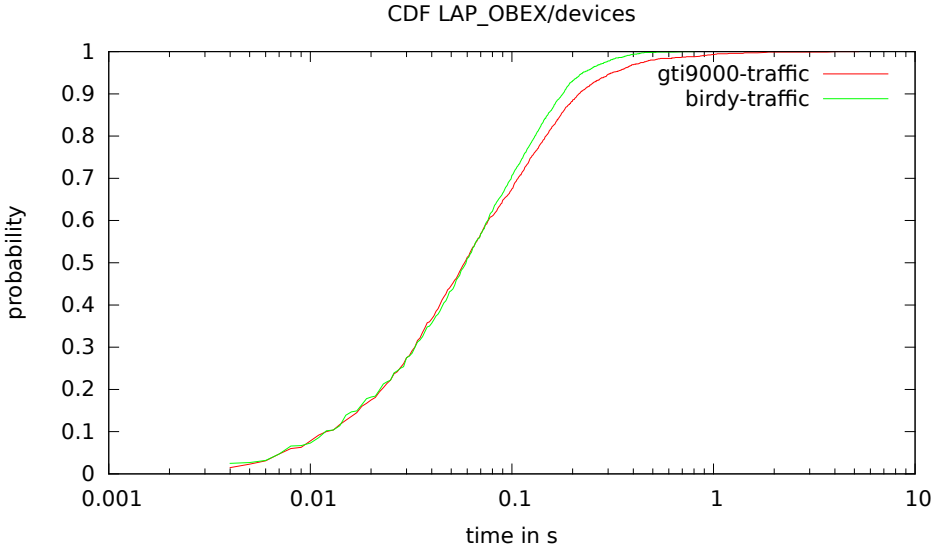
Enfin, dans un contexte de *Physical Tracking*, nous savons que la plupart des utilisateurs ne communiquent pas constamment avec leur périphérique *Bluetooth*. Sans trafic,



(A) AD2P_AVRCP

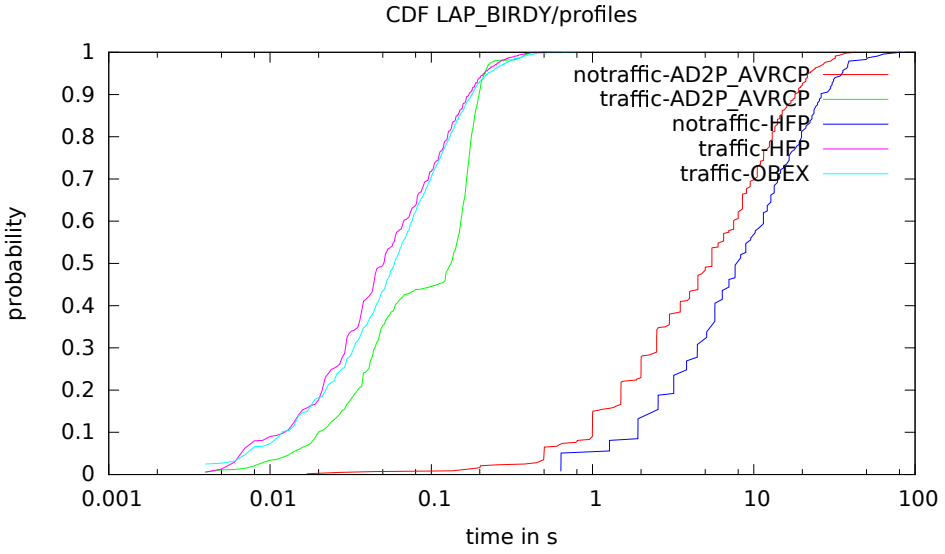


(B) HFP

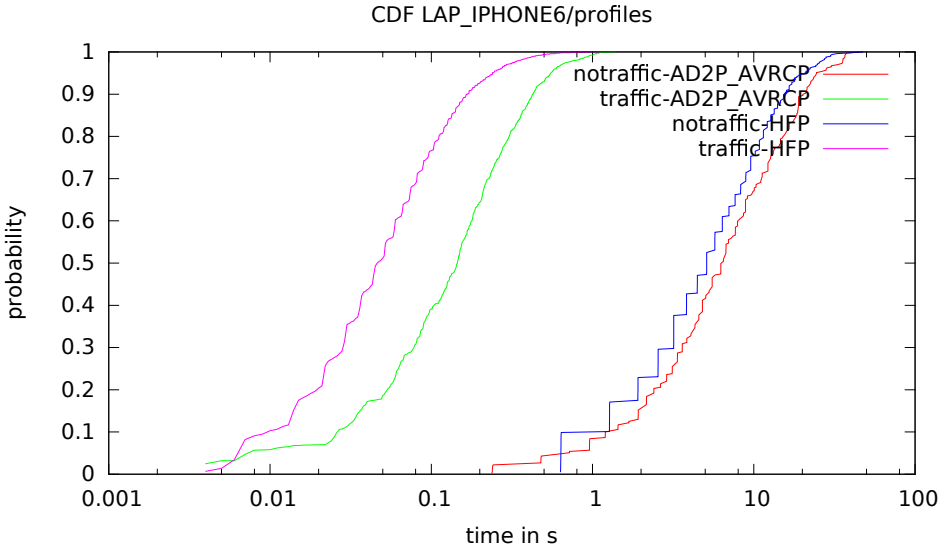


(C) FTP

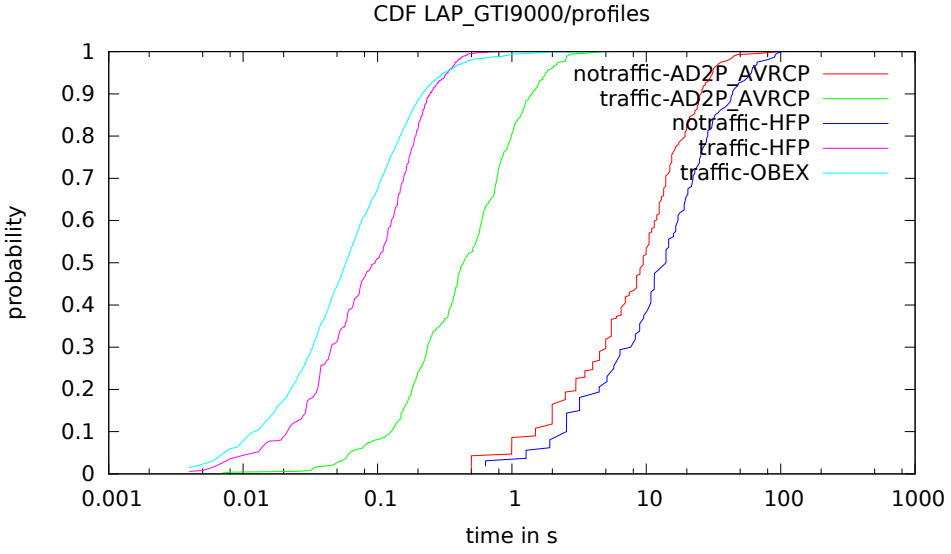
FIGURE 4.2: Distribution du temps inter-arrivée du *LAP* pour les différents *smartphones*. Chaque figure représente un profil particulier.



(A) Wiko



(B) iPhone 6



(C) GTI9000

FIGURE 4.3: Distribution du temps inter-arrivée du *LAP* selon les différentes profils *Bluetooth*. Chaque figure représente un *smartphone* en particulier.

le temps inter-arrivée du *LAP* augmente significativement. En effet, en général, ce temps augmente près de 20 fois lorsqu'il n'y a pas de trafic. En comparaison avec la méthode de découverte actif, notre méthode passif est, dans le pire des cas, trois fois moins efficace avec une probabilité de 95%.

Pour conclure, le type de profil et de *smartphone* ont un impact sur le temps inter-arrivée du *LAP*. Lorsqu'il y a du trafic, BPM a de meilleures performances (jusqu'à 10 fois) que les méthodes actives existantes. Sans trafic, les résultats en terme de temps de détection sont similaires en moyenne que les méthodes actives.

4.5 Recouvrir l'*UAP*

Les figures 4.4 et 4.5 montrent la distribution du temps requis pour décoder la valeur d'un *UAP* mesurée par une sonde. Sur chacun des sous-figures, le tracé distingue le cas où il y a (resp. il n'y a pas) de trafic échangé entre le *smartphone* pour chacun des trois profils *Bluetooth*(*HFP, AD2P_ AVRCP, FTP*). Pour une meilleure lisibilité, l'axe des x a été fixé selon une échelle logarithmique.

Comme discuté dans 4.2, le processus pour décoder l'*UAP* est théoriquement plus long que pour la partie *LAP*. En général, l'*UAP* requiert plusieurs trames de données comparé à une pour le *LAP*. Le tableau 4.2 montre le 95-ième percentile du temps requis (en secondes) pour décoder l' *UAP* pour chaque *smartphone* avec chaque profil. Les paquets de données pour décoder l'*UAP* sont dérivés de ceux du *LAP*. Premièrement, en pratique, nous pouvons constater que le temps pour décoder l'*UAP*, dans la plupart des cas, est trois fois supérieur au temps inter-arrivée du *LAP*. Deuxièmement, les résultats sont pertinents car le type de *smartphone* et de profil ont un impact sur le temps requis pour décoder l'*UAP* tout comme la partie *LAP*. Troisièmement, lorsqu'il y a du trafic, les performances de la combinaison du *LAP* et de l'*UAP* demeurent plus efficaces en comparé aux méthodes actives avec en plus une forte probabilité d'obtenir la bonne valeur de l'*UAP* décrit dans le tableau 4.3. Enfin, lorsqu'il n'y a pas de trafic, le temps pour récupérer l'*UAP* augmente drastiquement. De plus, la probabilité de recouvrir sa bonne valeur est très bas (moins de 40% dans la plupart des cas). Par conséquent, les performances de la combinaison du *LAP* et de l'*UAP* ne sont pas suffisantes dans un contexte de *Physical Tracking*. Comparé aux méthodes actives, la moyenne de temps pour l'*UAP* et le *LAP* est dix fois supérieur. Cependant, nos expérimentations se sont déroulés en écoutant uniquement un canal. Nous pouvons avoir de résultats plus précis si nous pouvons collecter plus de paquets de données lorsqu'il n'y a pas de trafic. Le procédé est d'utiliser plus d'un *Ubertooth* sur un canal différent et de modifier le code de la partie logicielle pour analyser et extraire cette nouvelle charge de données. De ce

fait, d'une part, nous pourrons améliorer le temps d'obtention de l'*UAP* et d'autre part, récupérer sa bonne valeur. Ce processus est aussi disponible pour améliorer le temps inter-arrivée du *LAP*.

Pour conclure, les résultats montrent que de façon similaire à la partie du *LAP*, le type de *smartphone* et de profil ont un impact sur le temps requis pour décoder l'*UAP*. Lorsqu'il y a du trafic, le temps de détection est meilleur que pour les méthodes actives connues. Sans trafic généré, la faible confiance que l'on peut apporter aux *UAP* inférés nécessite d'être prudent sur l'usage qui en est fait.

	Wiko	Samsung	iphone6
AD2P avec trafic	0.93	6.04	3.01
HFP avec trafic	1.77	1.36	7.31
FTP/OBEX sans trafic	1.26	1.11	NA
AD2P sans trafic	264.36	284.79	303.00
HFP sans trafic	187.04	183.63	128.58

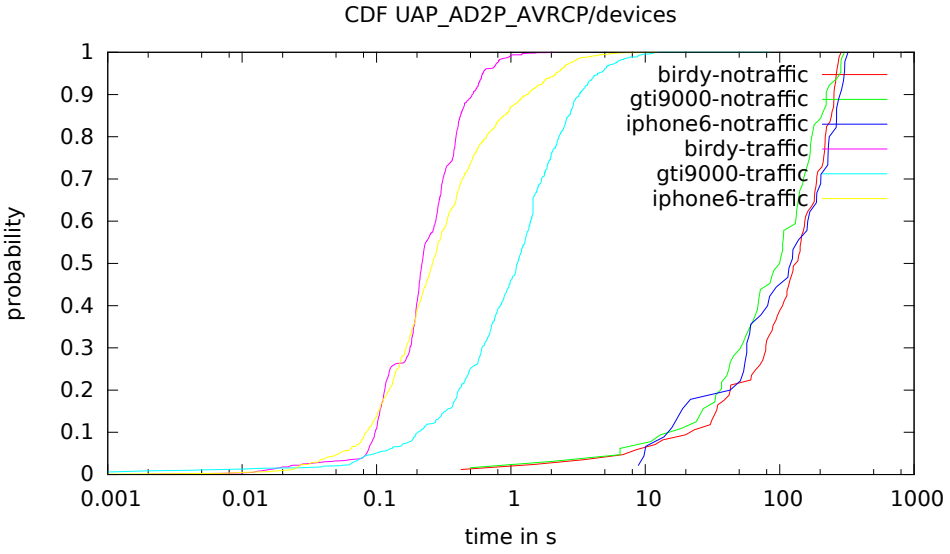
TABLE 4.2: Temps requis en secondes pour décoder la valeur de l'*UAP* pour chaque *smartphone* selon chaque profil lorsqu'il y a du trafic et lorsqu'il y en a pas.

	Wiko	Samsung	iphone6
AD2P avec trafic	99%	98%	88%
AD2P sans trafic	45%	40%	0%
HFP avec trafic	99%	99%	86%
HFP sans trafic	46%	49%	20%
FTP via OBEX avec trafic	69%	70%	NA

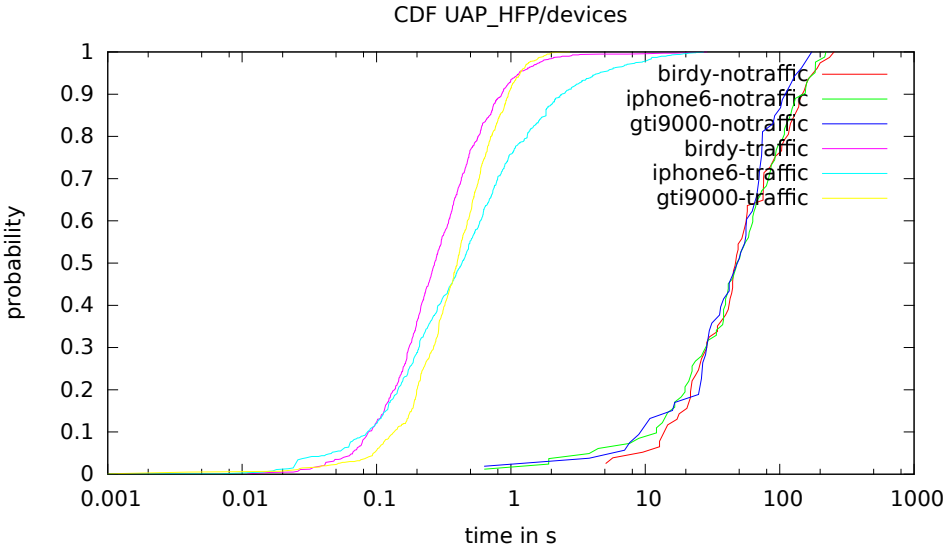
TABLE 4.3: Probabilité de décoder la bonne valeur de l'*UAP* pour chaque *smartphone* selon chaque profil lorsqu'il y a du trafic et lorsqu'il y en a pas.

4.6 Amélioration de BPM lorsqu'il n'y a pas de trafic

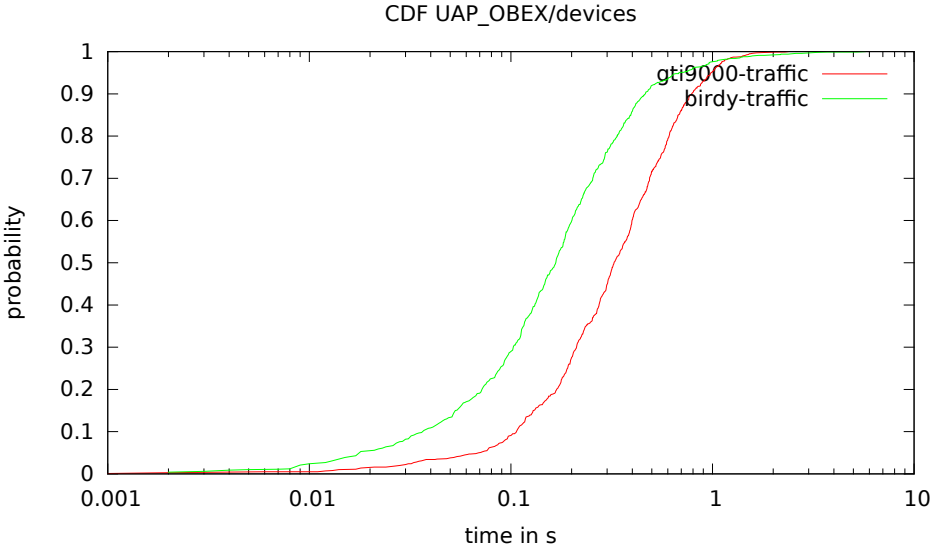
Nos expérimentations ont montré que les résultats pour le couple *LAP* et *UAP* lorsqu'il n'y a pas de trafic n'est pas applicable dans un contexte de *Physical Tracking*. Nous proposons d'améliorer les performances de notre méthode passive lorsque les périphériques n'échangent pas de données avec une analyse théorique. Le but est de réduire le temps de collecte du *LAP* et de l'*UAP* en écoutant sur plusieurs canaux en même temps avec de la simulation. Durant nos tests, nous avons montré que la distribution du temps inter-arrivée du *LAP* suivait une loi exponentielle. Ce résultat est démontré par la propriété de l'algorithme *HSK (Hop Selection Kernel)*, qui est la pièce matérielle responsable pour déterminer la séquence de sauts de canal d'un système *Bluetooth* [93, 127]. Par suite, basée sur une loi de Poisson, nous serions capable de déterminer le ratio « gain de temps



(A) AD2P_AVRCP

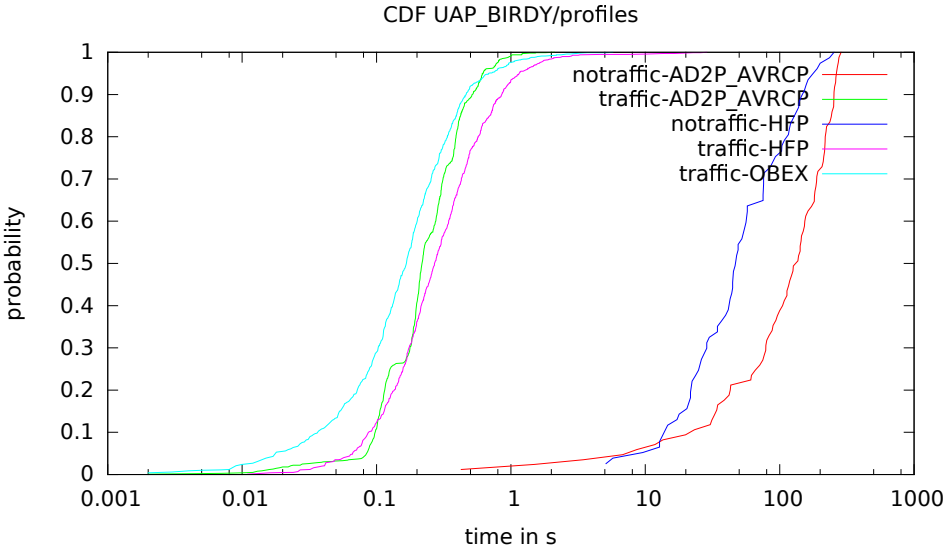


(B) HFP

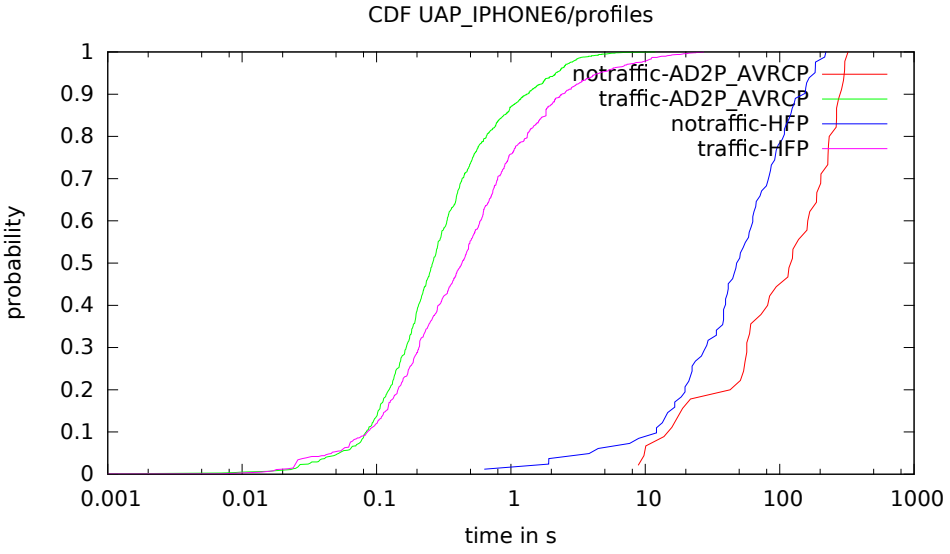


(C) FTP

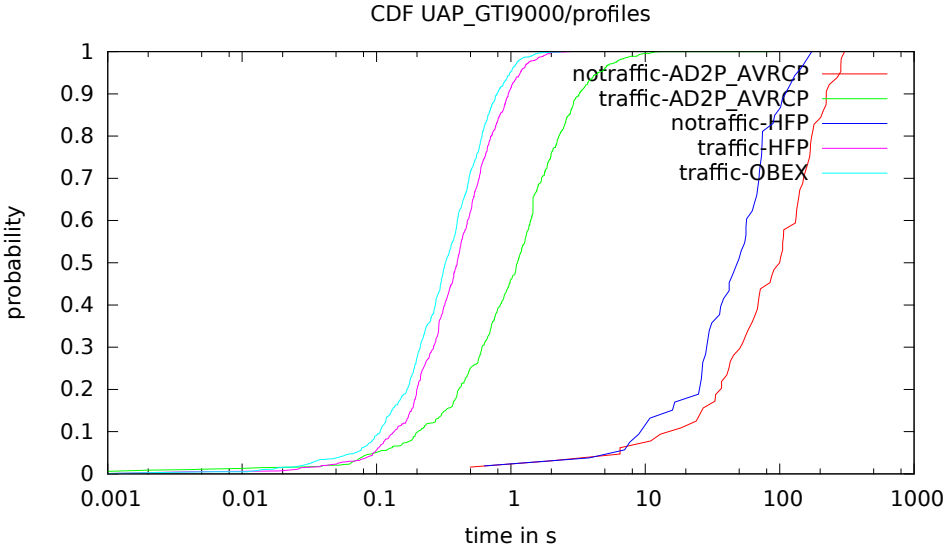
FIGURE 4.4: Distribution du temps requis pour recouvrir la valeur de l'*UAP* pour les différents *smartphones*. Chaque figure représente un profil particulier.



(A) Wiko



(B) iPhone 6



(C) GTI9000

FIGURE 4.5: Distribution du temps requis pour recouvrir la valeur de l'*UAP* selon les différentes profils *Bluetooth*. Chaque figure représente un *smartphone* en particulier.

pour collecter le LAP et l'UAP selon le nombre de canaux utilisé ». Enfin, cette méthode ne donne pas d'indication sur la probabilité d'avoir collecter la bonne valeur de l'UAP. Il faudra donc prévoir une étape supplémentaire qui consistera à vérifier l'intégrité de cette valeur.

4.7 Conclusion et recommandations

Pour conclure, nous avons reconsidéré la technologie du *classic Bluetooth* comme une technologie pouvant être utilisée dans un contexte de PT ou de collecte de trace de mobilité. Bien que pour le *classic Bluetooth* le service actif de découverte de périphérique à portée soit devenu moins performant, ces dernières années, nous avons démontré qu'il était toujours possible d'utiliser le *Bluetooth* pour suivre la mobilité des utilisateurs qui communiquent avec un dispositif *Bluetooth* en présentant BPM. Nos résultats montrent que dans la plupart des cas, notre méthode est plus rapide en terme de temps de détection que la méthode de découverte de périphériques à portée. De plus, elle met en avant les risques liés à la vie privée des utilisateurs car la collecte du LAP suffirait dans la plupart des scénarios à identifier un utilisateur et à suivre ses déplacements. Pour s'en protéger, la principale recommandation est de limiter les échanges de données entre un *smartphone* et le dispositif en *Bluetooth*. De façon pratique, il conviendrait de se connecter à son objet connecté uniquement lorsque le besoin se fait sentir et de se déconnecter immédiatement après. Par exemple, un utilisateur écoute de la musique avec un casque connecté en *Bluetooth*. Lorsque la musique n'est plus jouée et qu'il est mis en veille, le bon réflexe serait d'amorcer une déconnexion entre le *smartphone* et le casque et de s'assurer que les périphériques ne restent pas en mode découvrable, ensuite car le fait de garder les deux équipements connectés induit l'échange de paquets de synchronisation qui pourrait révéler à un attaquant la présence de l'utilisateur au travers du LAP contenu dans ces paquets.

Chapitre 5

BLEB : *botnet BLE*

Cette section considère le problème de suivi des utilisateurs au travers des empreintes générées par leur différentes interfaces sans fil, plus particulièrement, le *Bluetooth Low Energy (BLE)* qui est devenu de plus en plus populaire grâce à une meilleure gestion de la consommation énergétique et à une sécurité renforcée.

L'utilisation d'adresses MAC aléatoires et les différents changements dans le processus de communication étaient attendus pour prévenir le suivi d'utilisateurs au travers des informations émises par les objets connectés qu'ils portent (bracelet connecté, montre connecté).

Nous avons trouvé que les périphériques *BLE* mal-configurés induisaient un risque significatif pour le suivi d'utilisateurs et avons proposé BLEB, une attaque qui permet à un adversaire de suivre la mobilité des utilisateurs à grande échelle. Il s'appuie sur les adresses MAC et les informations divulguées dans les paquets de contrôle. Ces paquets peuvent être passivement ou activement collectés au travers d'une application mobile s'exécutant sur un *smartphone* non-modifié. Déployer de telles applications ou une librairie incluant ces fonctionnalités de suivi pourrait permettre de rassembler une grande quantité de périphériques (*i.e. un botnet BLE*) et facilement couvrir une large zone géographique avec une grande précision. Les propriétés d'un suivi par *botnet BLE* sont les suivantes :
– *Furtif* : Plusieurs méthodes de collectes sont disponibles, dont une passive, ce qui permet à un membre du BLEB de ne pas être détecté.

– *Passage à l'échelle* : Il n'y pas de limitation sur le nombre de périphériques à suivre.

– *Précision* : La précision dépend de la taille du *botnet*, des configurations des périphériques *BLE* cibles et de leur localisation. Or, contrairement aux attaques de suivi par *WiFi* ou *BPM* (présentées dans le chapitre précédent) qui nécessitaient que les mesures

soient effectuées depuis un ordinateur exécutant un *OS* classique (ordinateur, *Raspberry-pi.*.), les fonctionnalités nécessaires au suivi d'un noeud *BLE* sont disponibles sur les *smartphones Android* et *iOS*, lesquels sont bien plus nombreux, ubiquitaires améliorant ainsi la précision géographique.

– *Peu coûteux* : Il suffit qu'une application soit installée sur un *smartphone* pour que son utilisateur fasse parti du BLEB et collecte des données. De plus, cette application ou le code nécessaire au BLEB peut être caché dans une application qui parait tout à fait légitime. Il est donc possible de recruter des membres du BLEB à leur insu. C'est d'autant plus possible qu'une telle application ne consomme que peu d'énergie.

Il en résulte que BLEB permettrait de collecter des traces de mobilité de manière bien plus efficace que les précédentes méthodes et, en l'état actuel de son déploiement, le *BLE* serait donc plus menaçant pour la vie privée des utilisateurs que le *WiFi* ou le *classic Bluetooth* lors de leur déploiement initiaux, avant que des correctifs y soient apportés.

Le reste de ce chapitre va s'organiser de la manière suivante. Dans la section 5.1, un bref rappel sur les méthodes de suivi via le *Bluetooth* et *WiFi* permettra de mieux apprécier l'apport de BLEB. La section 5.2 détaille comment ce nouveau protocole est utilisé et configuré sur un ensemble de périphériques *BLE* populaires. Par la suite, la section 5.3 abordera l'architecture et les différentes composantes de BLEB. Enfin, une discussion sur les possibles applications et les extensions de BLEB sera présentée.

5.1 Rappels sur le suivi via le *WiFi* et le *classic Bluetooth*

Cette section remet BLEB dans le contexte des autres méthodes de suivi. Elle reprend les éléments présentés dans la section 1.1.3.2 et 2.

Comme discuté dans 1.1.3.2, les périphériques *WiFi* envoient périodiquement des *probes requests* à la recherche de point d'accès à portée. Le contenu de ces requêtes n'est pas chiffré et peut être facilement collecté par des périphériques *WiFi* à portée supportant le *mode moniteur*. Parmi les autres informations, ces paquets contiennent l'adresse MAC des *smartphones*. Comme cette adresse est unique, cela est suffisant pour identifier un périphérique et inférer la présence de son utilisateur. Cela résulte qu'un adversaire peut passivement suivre la mobilité des utilisateurs au travers de point d'accès compromis [128] ou des périphériques dédiés à des mesures. De telles attaques de suivi d'utilisateurs peuvent rester anonymes vu qu'elles ne requièrent pas que l'attaquant envoie une seule trame.

Utiliser de telles attaques avec le *classic Bluetooth* peut sembler plus difficile de par la couche physique qui utilise les sauts de fréquences (voir 2.1.6) où un périphérique va changer périodiquement de canal parmi les 79 proposés. Cependant, pour la découverte de périphériques à portée, les appareils doivent envoyer une requête sur un sous-ensemble des canaux. Les appareils à portée configurée pour être *découvrable* vont éventuellement écouter cette requête et y répondre avec une trame contenant son adresse MAC *Bluetooth*. Un adversaire peut sonder périodiquement les canaux *Bluetooth* pour détecter la présence des périphériques à portée et leur mobilité.

Les menaces sur la vie privée des utilisateurs avec l'utilisation de ces attaques ont été détaillées pour le cas du *WiFi* et du *Bluetooth* dans [12, 128]. Cela à conduit les constructeurs a changé la configuration des périphériques. Dans le cas du *WiFi*, ils ont supprimé la liste des réseaux auxquels le *smartphone* s'est déjà connecté. Ils ont aussi ajouté le fait de pouvoir utiliser des adresses MAC aléatoires durant la phase de découverte de réseaux [86–89].

Un adversaire qui écoute les canaux *WiFi* est toujours capable de collecter les *probes requests* mais ces dernières ne contiennent plus d'adresse MAC unique et ne pourront plus être utilisées dans un système de traçage de la mobilité des individus (voir 1.2.4.5).

Dans le cas du *classic Bluetooth*, les constructeurs ont choisi de restreindre le temps qu'un périphérique reste en mode *découvrable* (*e.g.* $60s$) et requiert que les utilisateurs activent explicitement ce mode. Ces récentes modifications induisent que les attaques de suivi des utilisateurs qui pouvaient être utilisées dans les débuts du *WiFi* et du *classic Bluetooth* sont dorénavant moins utiles ou trop coûteuses à déployer pour un attaquant ou pour une campagne de mesure de mobilité. L'augmentation de la sensibilisation sur les informations liées à la vie privée des utilisateurs a également eu un impact positif sur la conception des protocoles les plus récents. Par exemple, le *Bluetooth Low Energy* inclut nativement la possibilité d'utiliser des adresses MAC aléatoires (voir 3.1.2). L'utilisation de cette technologie est prévue pour plusieurs applications dans le domaine de l'Internet des Objets, des capteurs d'activités ou des dispositifs portatifs.

5.2 Périphériques *BLE* et adresses MAC *BLE* utilisées

Comme discuté dans 3.1.2, la norme *BLE* permet d'utiliser plusieurs types d'adresses MAC dépendant du niveau de sécurité des informations privées. A partir de la description du tableau 5.1, nous pouvons espérer que les périphériques fréquemment portés par les utilisateurs utilisent une adresse privée résolvable ou non-résolvable car les adresses publiques et aléatoires statiques devraient être utilisées uniquement par des périphériques

Type	Possibilité de suivre un utilisateur	Commentaires
Public	*OUI*	L'*OUI* donne des informations sur le modèle et le constructeur.
Aléatoire statique	Oui	Possibilité de collision d'adresses.
Aléatoire résolvable	Non	Possible si l'adversaire possède l'*IRK*.
Aléatoire non-résolvable	Non	Il y a un risque dépendant de le mise à jour de la fréquence de saut de canaux.

TABLE 5.1: Type d'adresses *BLE* et le risque d'être suivi.

qui ne sont pas sensibles au suivi de la mobilité tels que les capteurs domotiques, les objets connectés pour de la localisation à l'intérieur de bâtiment ou des objets connecté qui distribuent des coupons de promotion.

5.2.1 Périphériques *BLE* étudiés

Nous avons choisi d'étudier plusieurs objets connecté *BLE* dans le but de déterminer si les constructeurs configurent leurs périphériques avec une adresse MAC appropriée. Comme le décrit la norme 4.2 du *Bluetooth* [126], les mécanismes de sécurité et adresses choisies par un périphérique dépendent notamment de la possibilité d'afficher ou entrer une clé partagée sur celui-ci. Le standard *BLE* distingue trois capacités d'affichage d'entrées : *No Input*, *Yes/No* et *Keyboard*, c'est-à-dire que l'objet connecté a au moins un clavier numérique (0 à 9) et deux boutons pour *yes* et *no*. Concernant les capacités d'affichage en sortie, l'objet connecté a au moins la capacité d'afficher et/ou de communiquer un nombre décimal à 6 chiffres.

Nous avons choisi un ensemble de périphériques selon ces différentes catégories : trois bracelets connectés (un *Xioami MiBand*, un *FitBit Flex* et un *FitBit Charge 2*), une montre connectée *ZeCircle*, trois objets connecté *BLE* qui empêchent de perdre ses objets (deux de marque générique et un *TrackR*, le modèle le plus populaire) et un moniteur de fréquence cardiaque *Geonaute*. Tous ces périphériques sont souvent portés par les utilisateurs.

	Type
Geonaute	Public
TrackR	Aléatoire statique
Marque non-connue Tracker 1	Public
Marque non-connue Tracker 2	Public
ZeCircle	Public
Fitbit Flex	Aléatoire statique
Fitbit Charge 2	Aléatoire statique
Xiami MiBand	Public

TABLE 5.2: Type d'adresses utilisées par les objets connectés *BLE*.

5.2.2 Mise en place des mesures

Tous ces périphériques restaient connectés aux *smartphones* pour une période de temps limitée. Lorsqu'ils n'étaient pas connectés, les objets connectés envoyaient périodiquement des paquets de contrôles, appelé ADV_PKT (voir 3.1.3).

Nous avons collecté ces paquets en utilisant *Ubertooth* [106], un analyseur de paquets de données *classic Bluetooth* et *Bluetooth Low Energy*. Cet outil peu coûteux peut être configuré pour rester sur un canal donné et récupérer le flux binaire démodulé. La librairie `libtbb` permet de chercher des trames *Bluetooth* à l'intérieur de ce flux.

Pour nos expérimentations, nous avons collecté des paquets de contrôles lorsque les périphériques n'étaient pas connectés, durant 7 jours, 10 minutes par jour pour chaque objet connecté *BLE*. Cela a permis de détecter les changements d'adresse MAC à court et à long terme sur plusieurs cycles de charge.

Le tableau 5.2 décrit les types d'adresses utilisés lorsqu'ils sont en phase d'avertissement. Il montre que malgré les efforts de la norme *Bluetooth* concernant les adresses MAC privées, tous nos objets connectés utilisent soit une adresse publique soit une adresse aléatoire statique, ce qui implique qu'ils sont traçables. De plus, aucune adresse aléatoire n'a changé au cours d'une semaine alors que les dispositifs ont subi plusieurs cycles d'énergie, par exemple lors du redémarrage de l'objet connecté, lorsqu'ils n'avaient plus de puissance ou lors d'un changement de batterie.

5.2.3 Remarque sur les adresses privées résolvables

Malheureusement, aucun de nos périphériques utilise une adresse MAC privée résolvable. Cette section se focalise sur l'exploitation de cette faille. Cependant, cela ne veut pas dire que l'utilisation d'une adresse MAC privée résolvable n'est pas sans risque. En effet, un adversaire peut toujours suivre un utilisateur s'il connaît l'adresse MAC d'origine et

l'*IRK* échangée durant la troisième phase du processus d'appariement (décrit dans 3.1.4). Obtenir ces informations requiert un peu plus d'effort. Le moyen le plus simple serait d'implémenter un *honeypot* qui offrirait des services attractifs pour inciter les *smartphones* a y entamer un processus d'appariement via le *BLE*. L'attaquant peuplerait ainsi une base de données des *smartphones* qui auraient interagi avec l'*honeypot* et serait composée d'adresse MAC publique et de la clé *IRK* associées. Pour détecter la présence d'un *smartphone* utilisant une adresse résolvable, il suffit de collecter une trame contenant une telle adresse générée par le *smartphone* cible. Si celui-ci utilise une application qui déclenche un *scan* actif, il enverra un paquet nommé *SCAN_REQ*. Dans certains cas, le *smartphone* peut aussi être à l'écoute passivement des *ADV_PKT*. S'il détecte un périphérique dont il a été connecté avec, il lui enverra un paquet nommé *CONNECT_REQ* pour ré-établir une connexion.

5.2.4 L'adresse MAC comme identifiant : impact des adresses aléatoires

L'utilisation d'adresses aléatoires peut créer un risque de collision entre périphériques *BLE*. Cependant, cette probabilité reste négligeable car il y a 2^{46} adresses possibles. Cela résulte que la menace sur la sécurité des informations privées engendrée par l'utilisation d'une adresse publique est similaire à la menace causée par une adresse aléatoire statique ou le manque de rotation d'une adresse privée résolvable. De plus, dans certains cas, l'adresse MAC n'est pas le seul identifiant que nous pouvons collecter. En effet, ces paquets peuvent contenir plusieurs informations dans le champ *advertising data*, plus précisément un identifiant unique sur 128 bits appelé *UUID*. Par exemple, les objets connectés *Fitbit* mettent la valeur d'un *UUID* directement dans le champ *advertising data* d'un paquet.

5.3 Proposition de BLEB

Comme décrit dans 5.2.2, tous nos périphériques de test utilisent soit une adresse publique, soit une adresse aléatoire statique qui peuvent être utilisées pour suivre les utilisateurs. Cela induit que chaque fois qu'une trame *BLE* est émise, celle ci révèle la présence d'un périphérique et de son utilisateur. Si le périphérique *BLE* n'est pas connecté, il restera en mode *avertissement* et enverra fréquemment des *ADV_PKT*. Avec une sonde de mesure, un adversaire qui collecte ces *ADV_PKT* peut détecter la présence d'un individu. Plusieurs sondes réparties dans une zone (un bâtiment, une ville, un pays ou le monde entier) permet de suivre la mobilité des individus. Comme décrit plus haut,

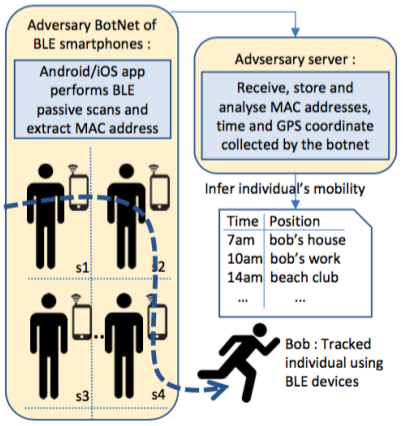


FIGURE 5.1: Architecture de BLEB.

ces sondes peuvent être des *smartphones* sur lesquels sont installés une application qui collectent les données. Le code de collecte des données peut avoir été installé à l'insu des utilisateurs. Les *smartphones* infectés par ces applications font alors parti d'un *botnet* de collecte de données *BLE*, que l'on nomme BLEB. Le BLEB est contrôlé par l'entité qui souhaite suivre la mobilité des utilisateurs, ici appelé l'attaquant. Les *smartphones* du BLEB envoient les données collectées à un serveur controlé par l'attaquant.

Comme le montre la figure 5.1, les membres du *botnet* collectent périodiquement les ADV_PKT et envoient à un serveur de l'adversaire les adresses MAC ainsi que le *timestamp* et la position géographique actuelle. Le serveur de l'adversaire sauvegarde et analyse les données collectées pour extraire des traces de mobilité pour chaque individu inclus dans les mesures.

La précision ou la qualité d'une telle trace de mobilité va dépendre de 1) du nombre de sondes (membres du BLEB) et de la pertinence de leur positionnement ; 2) la probabilité de détecter un périphérique qui est près d'une sonde.

La prochaine section décrit les périphériques qui peuvent être intégrés dans BLEB, les *API* disponibles et l'effort que devra fournir un adversaire pour contrôler ce périphérique. La section 5.5 se focalise sur la probabilité de détecter un utilisateur marchant à proximité d'une sonde de mesure.

5.4 Sonder les canaux *BLE* pour récolter les adresses MAC

Comme décrit dans la section 3.1.3, les *scans BLE* peuvent être soit actif ou passif. Les *scans* passifs ne rassemblent que des informations obtenues par l'intermédiaire des ADV_PKT émis spontanément par les périphériques *BLE* tandis que les *scans* actifs

nécessitent un échange supplémentaire via l'envoi d'un paquet appelé $SCAN\_REQ$ et d'un paquet de réponse appelé $SCAN\_RSP$ après la réception d'un ADV_PKT initial. Cela veut dire qu'un *scan* actif et passif permet de détecter le même ensemble de périphériques *BLE*, la majeur différence est l'aspect furtif des *scans* passifs.

Malheureusement, les systèmes d'exploitation sur appareils mobiles tel que *iOS* et *Android* ont instauré des restrictions sur la quantité d'information qui peut être obtenue au travers de leur interface de programmation applicative.

5.4.1 *iOS*

Sur *iOS*, l'interface de programmation applicative disponible pour une application permet de déclencher des *scans* actifs lorsque l'application est en cours d'exécution et des *scans* passifs lorsque l'application tourne en arrière-plan.

Malheureusement, cette interface de programmation applicative ne donne pas l'accès à l'adresse MAC des périphériques. A l'inverse, elle retourne uniquement le nom du périphérique et génère un *UUID* qui est spécifique à la paire périphérique scannée et au périphérique qui exécute un *scan*. Cela signifie que deux périphériques *iOS* scannant un même objet connect" *BLE* vont générer deux *UUIDs* différents ; un périphérique *iOS* va toujours générer le même *UUID* au cours de plusieurs rencontres successives avec un objet connecté donné si son adresse MAC reste la même.

Cela voudrait dire qu'un adversaire recrutant un membre de BLEB en déployant une application *iOS* ou une libraire, n'aura jamais accès à l'adresse MAC de la cible. Cependant, un unique périphérique *iOS* membre de BLEB va pouvoir suivre la mobilité de périphériques au fil du temps grâce à son *UUID*. Bien qu'il soit difficile de fusionner les traces collectées par un appareil *iOS* avec les traces collectées par d'autres membres de BLEB (*iOS* ou *Android*), certaines stratégies peuvent être développées. Si deux nœuds de BLEB dénotés iOS_i et A_j sont co-localisés avec un périphérique ciblé d_x, nous pouvons deviner que l'adresse MAC $MAC(d_x)$ obtenue au travers de A_j fait référence au même périphérique d_x que l'$UUID(iOS_i, d_x)$ qui est obtenue par l'interface de programmation applicative de iOS_i.

5.4.2 Android

Alors qu'*Android* traite effectivement les ADV_PKT spontanément envoyés par les appareils voisins, nos tests ont montré que l'interface de programmation applicative ne montre que les résultats des *scans* actifs. Cela signifie que si l'on est à porté d'un périphérique cible d_x qui émet des ADV_PKT, l'API ne nous permet pas d'être informé sans

qu'elle génère automatiquement un *scan* actif (*SCAN_REQ* suivi d'un *SCAN_RSP*). Seul la réponse du *SCAN_RSP* est remonté par l'application. L'interface de programmation applicative fournit l'adresse MAC du périphérique détectée ainsi que son nom et son *UUID* (s'il y en a). Un adversaire recrutant un membre de BLEB au travers le déploiement d'une application *Android* ou une libraire partagée peut être capable de détecter activement les périphériques voisins. L'aspect négatif est que cette pratique de suivi des utilisateurs peut être détecté et les performances sont influencées par le temps requis pour compléter un *scan* actif.

5.4.3 *Linux*, *Ubertooth* et *smartphones* rootés

Pour les appareils *Linux*, la libraire *bluez* [129] permet de déclencher des *scans* passifs et actifs et fournit les données brutes obtenues à partir des *ADV_PKT* et des *SCAN_RSP*, incluant les adresses MAC. Sur *iOS*, l'interface de programmation applicative expose uniquement un *UUID* dérivé de l'adresse MAC par le système d'exploitation. Si le *smartphone* est rooté, la fonction qui convertit l'adresse MAC en *UUID* est exposée, permettant de retrouver l'adresse MAC au travers des *scans* actifs et passifs. De manière similaire, sur les *smartphones Android* rootés, il est possible de capturer le trafic *BLE* comme le fait *bluez* sur *Linux*. Cela permet d'avoir le contenu des *ADV_PKT* envoyés par les périphériques à portée. De telles périphériques peuvent ensuite être utilisés pour le suivi des utilisateurs en mode furtif.

5.5 Le temps inter-*ADV_EVT*

Lorsqu'un utilisateur passe à proximité d'un périphérique membre d'un BLEB, la probabilité que ce périphérique puisse détecter sa présence va dépendre de la fréquence d'envoi d'*ADV_PKT* d'un périphérique cible. Pour évaluer la performance des nœuds *BLE* lorsqu'ils utilisent des *scans* passifs, nous avons étudié la période des *ADV_EVT* (*advertising events*) de nos périphériques de tests présentés en 5.2.2.

Durant nos mesures, les périphériques n'étaient pas connectés et émettaient des *ADV_PKT* périodiquement. Nous avons utilisé l'outil *Ubertooth* configuré sur le canal 37 pour collecter des paquets sur deux périodes de 4 heures pour chaque objet connecté *BLE*. Dans le but de déterminer si les mouvements et vibrations avaient un impact sur le comportement des objets connectés *BLE*, ces derniers étaient statiques durant les 4 premières heures et étaient portés par un utilisateur mobile durant les 4 dernières heures.

Le tableau 5.3 montre la moyenne du temps inter-émission des *ADV_PKT* et le 95 ième percentile du temps inter-émission des *ADV_PKT* pour chaque objet connecté. Le

	Scenario	Moyenne	95ième percentile
TrackR	Mobile	1.6	3.057
TrackR	Statique	1.480	3.060
Marque non-connue TrackR 1	Mobile	4.588	9.557
Marque non-connue TrackR 1	Statique	4.221	9.010
Marque non-connue TrackR 2	Mobile	4.838	10.010
Marque non-connue TrackR 2	Statique	4.822	9.010
Xiami MiBand	Mobile	1.230	2.789
Xiami MiBand	Statique	1.160	2.585
ZeCircle	Mobile	1.6	3.057
ZeCircle	Statique	1.534	2.510
Fitbit Flex	Mobile	1.763	2.600
Fitbit Flex	Statique	1.830	2.678
Fitbit Charge 2	Mobile	1.657	2.690
Fitbit Charge 2	Statique	1.789	2.580

TABLE 5.3: La moyenne et le 95ième percentile du temps inter-émission des ADV_PKT. Le temps est en secondes.

95ième percentile représente la durée maximum qu'un membre de BLEB doit attendre avant de détecter la présence d'un périphérique cible dans 95% du temps, c'est-à-dire $P(DetectionTime < X) = 0.95$ avec X la valeur du 95ième percentile. Cette métrique permet d'exclure les valeurs maximales extrêmes, non représentatives et sans doute dues à des collisions.

Nous observons que le fait que les objets connectés soient mobiles ou statiques ne semble pas avoir d'impact sur le temps inter-ADV_EVT de nos objets connectés. A l'exception des *TrackR* sans marques connues, le 95ième percentile de nos périphériques se situent en dessous des trois secondes, un résultat applicable dans un contexte de *Physical Tracking*. Pour mettre les choses en perspective, si l'on considère une portée de 10 mètres pour le périphérique *BLE* (hypothèse basse), une vitesse de déplacement d'un utilisateur de 1.5m/s, la cible et le membre de BLEB marchant en direction opposée dans un couloir, le membre de BLEB est capable de détecter si un ADV_PKT est émis en 6.6 secondes après avoir été à portée l'un de l'autre. Cela laisse largement le temps au noeud de BLEB la possibilité de détecter le périphérique *BLE*.

5.6 Temps requis pour compléter un *scan* actif

Le *BLE* utilise les canaux 37, 38 et 39 pour l'envoi des ADV_PKT, de $SCAN\_REQ$ et $SCAN\_RSP$. Le *scanner* (le *smartphone*) et l'*advertiser* (objet connecté *BLE*) vont changer de canal périodiquement. Cela peut prendre un temps conséquent avant qu'ils restent sur un même canal assez de temps pour permettre l'échange d'ADV_PKT, de

SCAN_REQ et de *SCAN_RSP*. Par conséquent, les *scans BLE* peuvent être potentiellement long et variable en terme de durée.

Comme décrit dans [130], les performances des procédures de découverte actif peuvent avoir un impact sur les performances des attaques de suivi d'utilisateurs. Dans un scénario favorable avec uniquement un *advertiser i.e.* un périphérique qui envoient périodiquement des *ADV_PKT*, un *scanner i.e.* un périphérique qui est à l'écoute de ces paquets et qui va envoyer un paquet de requête, un *ADV_EVT* génère l'envoi successif d'*ADV_PKT* sur chacun des trois canaux (37,38,39), la probabilité pour un *scanner* de détecter un périphérique ($\approx 33\%$). Cela peut s'expliquer par le fait que les *scanner* et les *advertiser* changent périodiquement de canal de manière asynchrone. Si le nombre de *scanners* augmente, la probabilité de détecter un périphérique diminue, c'est-à-dire qu'avec dix *scanners* et cinq *advertisers*, la probabilité de détecter tous les périphériques à portée est en dessous de 5%. La raison principale est la collision des paquets de contrôles (tels que les *SCAN_REQ PDU* ou les *SCAN_RSP PDU*) durant ce processus. De plus, ils ont montré que la latence de découverte de périphériques augmente exponentiellement avec le nombre de périphériques. Par exemple, pour un *advertiser* et trois (resp. dix) *scanners*, la latence moyenne pour détecter les périphériques est aux alentours de 6 secondes (resp. 10 secondes). Le nombre de périphériques qui sonde les canaux simultanément augmente exponentiellement le délai moyen de découverte de périphériques. Enfin, l'augmentation uniquement d'*advertisers* n'a pas d'impact sur le temps de détection et sur la latence moyenne pour le processus de découverte de périphériques à portée.

5.7 Impact sur les performances de BLEB

L'utilisation d'un ordinateur portable sous *Linux* ou des *smartphones iOS* et *Android rootés* permet à BLEB de rester indétectable car il n'utilise que des *scans* passif sans perte de précision. En effet, le temps moyen requis pour compléter un *scan* actif est 2 à 4 fois supérieur au temps moyen requis pour détecter passivement un utilisateur. Pour le moment, il n'y a pas le besoin d'exploiter des informations supplémentaires qui peuvent être obtenues par les paquets de requêtes. Le point négatif est que le déploiement de tels nœuds BLEB requiert plus d'effort pour un adversaire vu qu'il y a beaucoup moins de *smartphones rootés* ou d'ordinateurs sous *Linux* que de *smartphones* non-modifiés.

Les *smartphones Android* non-modifiés avec une application ou une bibliothèque compromises par un adversaire permet à BLEB d'effectuer un suivi des utilisateurs à grande échelle. La proportion des utilisateurs qui vont être suivis pourrait être uniquement limité par le taux de pénétration des périphériques *BLE* mal-configurés (ceux qui utilisent un identifiant (MAC ou *UUID*) qui ne change pas au cours du temps). Le coté négatif

est que le comportement des ces nœuds BLEB peut éventuellement être détecté car les interfaces de programmation applicatives sous *Android* ne permettent de faire que des *scans* actifs. Un autre aspect négatif de ce type de *scan* est qu'il réduit la probabilité de détection lorsque les contacts sont limitées dans le temps.

5.8 Conclusion et recommandations

Nous avons étudié dans cette section la possibilité de suivre les déplacements des utilisateurs sur une plus grande échelle via l'utilisation d'un *botnet BLE* installé sur le *smartphones* des utilisateurs. Avec un coût de déploiement peu élevé de la part d'un attaquant, cette méthode présente un potentiel risque pour les données privées des utilisateurs. Pour s'en protéger, les utilisateurs de *smartphones* et d'objets connectés *BLE* est d'une part la désactivation de l'interface *Bluetooth* lorsque qu'ils ne sont pas en utilisation et d'autre part, de conserver une connexion entre le *smartphone* et l'objet connecté pour éviter que ce dernier émette périodiquement des ADV_PKT qui pourraient divulguer sa présence aux autres membres du *botnet*. Il faut néanmoins noter que lors d'une connexion, l'*Access Address* (AA) reste inchangé jusqu'à la prochaine connexion. Un attaquant qui aurait capté l'AA d'une connexion peut donc suivre l'utilisateur. De manière similaire il faut s'assurer que les données échangées soient chiffrées car celles-ci pourraient contenir des identifiants et autres données sensibles.

Plus en amont, lors de l'installation d'applications, les utilisateurs doivent vérifier les permissions demandées par ladite application et s'assurer qu'aucune incohérence existe. Par exemple, une application qui propose de faire des achats en ligne et qui demande la permission d'interagir avec l'interface *Bluetooth* est potentiellement suspecte. De plus, il faudrait avoir le réflexe de surveiller la consommation énergétique générée par chaque application installée, une consommation anormalement élevée pourrait être synonyme de réveil périodique du téléphone pour des opérations de *scans* ou la réception d'*intents* de *scans BLE* auxquels l'application se serait abonnée.

En outre, il conviendrait aux utilisateurs de n'acheter que des périphériques qui sont correctement configurés *i.e.* qui utilisent des adresses privées pour empêcher qu'un identifiant unique soit continuellement envoyé en clair sur le réseau. Pour le processus d'appariement avec cette technologie, il ne faudrait pas s'appairer avec n'importe quel périphérique qui pourrait stocker les clés secrètes (notamment l'IRK) donnant ainsi la possibilité d'outrepasser la protection des adresses privées. Par suite, pour les constructeurs, la principale recommandation est l'utilisation de toutes les fonctionnalités de sécurité proposées par la norme *Bluetooth*. En effet, chaque objet *BLE* embarqué par un utilisateur doit utiliser des adresses privées résolvables pour éviter que ce dernier soit détectable dans la plupart

des cas. Enfin, les OS mobiles ont déjà pris de bonnes initiatives en bannissant la possibilité de réaliser des *scans* passifs par le biais des interfaces de programmation actives et certains ont bloqué les *scans* actifs en *BLE* lorsque l'application tourne en tâche de fond. La recommandation serait de limiter la possibilité pour les applications de cloner les ADV_PKT émis par un objet connecté et de les retransmettre sur les canaux afin de reproduire leur comportement.

Chapitre 6

Crowd Localization

6.1 Contexte

L'utilisation du *Bluetooth Low Energy (BLE)* est récemment devenu populaire dans les services de *crowd-localization*, c'est à dire la localisation d'objets collaborative par une foule équipée de *smartphones* d'une même communauté d'utilisateurs. Dans ce chapitre, nous mettons en évidence la menace que constituent de tels services et nous proposons une architecture de *crowd-localization* qui garantie le respect de la vie privée des utilisateurs.

Les applications de *crowd-localization* font appel à tous leurs utilisateurs pour effectuer des *scans BLE* périodiquement et de détecter les périphériques *BLE* à portée. L'utilisateur exécute une application sur son *smartphone*, laquelle va périodiquement envoyer la liste des périphériques qu'il a détecté autour de lui vers un serveur central. Les utilisateurs peuvent ensuite se connecter au serveur pour obtenir la localisation de leurs périphériques. Sur la plupart des services, pour obtenir la position de son objet, il faut le déclarer comme perdu. Cela permet à un utilisateur de suivre la mobilité de ses périphériques sans payer de carte *SIM* dédiée ou se soucier d'un quelconque réseau auquel l'objet doit se connecter.

Nous montrons que ces périphériques utilisent une adresse publique ou une adresse aléatoire statique pour identifier un utilisateur. Ces adresses sont envoyées au serveur de la compagnie sans modification, accompagnées de la position *GPS* et de l'identité du *smartphone* qui a collecté l'information.

Un tel comportement fait courir un risque inutile aux utilisateurs car la mobilité de ses objets (« a priori » la sienne) peut être suivie par la compagnie qui fournit le service. Étant donné que le service nécessite que les objets connectés utilisent une adresse publique ou

une adresse aléatoire statique, l'objet connecté (et donc l'utilisateur) devient sensible aux attaques de type BLEB décrites au chapitre précédent.

Nous proposons PPCL (*Privacy Preserving Crowd Localization*) une nouvelle architecture qui effectue le *crowd-localization* tout en préservant la confidentialité des utilisateurs. PPCL rend impossible le suivi des utilisateurs par une compagnie ainsi que tout adversaire qui possède un dispositif *BLE*. C'est uniquement lorsque l'utilisateur déclare son objet comme perdu que le serveur va pouvoir déterminer sa position. Il ne pourra pas récupérer l'historique des positions. Dès que l'objet est retrouvé, les clés sont changées et le serveur sera de nouveau incapable de suivre l'objet. Par rapport aux propositions actuelles, PPCL n'a aucun impact sur les objets connectés, la durée de vie de leurs batteries ou celles du *smartphone*. PPCL n'altère pas les performances du *crowd-localization*. Son seul inconvénient est une légère augmentation du coût de calcul pour les serveurs de la compagnie lorsqu'un utilisateur déclare la perte de son appareil.

Quatre variantes seront proposées, en fonction des risques dont on souhaite se protéger et du nombre de objets connectés suivis.

Pour résumer, les contributions de ce chapitre sont les suivantes :

1. Analyse des applications de *crowd-localization* existantes et mise en exergue des risques liés à leur utilisation ;
2. Analyse des propriétés souhaitées pour une solution idéale de *crowd-localization* respectueuse de la vie privée et proposition de PPCL avec quatre variantes en fonction des contraintes applicatives, matérielles et des attaques que l'on souhaite tolérer.

6.2 Localisation et suivi d'objets

L'augmentation des objets connectés introduit de nouveaux moyens pour localiser un objet. Des solutions existantes utilisaient la plupart du temps le *GPS* couplé au *GPRS* ou à une radio à haute fréquence [131, 132]. De telles solutions n'étaient pas toujours efficaces dû au fort coût des dispositifs et à l'énergie consommé en plus de l'imprécision dans un contexte de géo-localisation à l'intérieur de bâtiments. Pour pallier ce problème, le déploiement d'une infrastructure d'objets connectés fixes a été proposée [114, 115, 133, 134]. Cependant, à grande échelle, cette approche devient trop coûteuse.

Pour éviter l'utilisation d'une architecture, les chercheurs ont proposé des solutions construites autour du *crowd-localization*, un système dont le processus de géo-localisation est effectué selon les informations remontées par les utilisateurs [135–137].

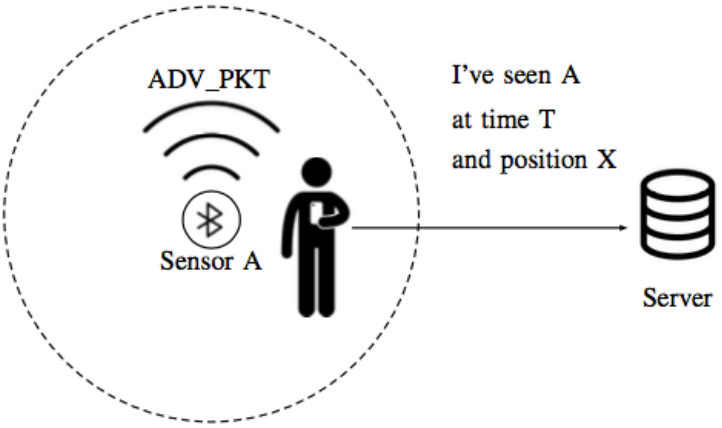


FIGURE 6.1: Entités impliquées dans le *crowd-localization* : l'objet connecté envoie périodiquement des ADV_IND contenant un identifiant noté A reçu par l'application du *smartphone* puis envoyé à un serveur accompagné de l'heure T et de la position X.

Dans un contexte commercial, des compagnies (*TrackR*, *Wistiki*, *iTrackEasy* and *Zizai Tech Nut*) ont déployé une solution de *crowd-localization* qui aide les utilisateurs à retrouver leurs objets perdus grâce aux informations envoyées par la communauté. Alors que d'autres interfaces telles que le *WiFi* auraient pu être utilisées [138], la plupart des implémentations préfèrent *BLE* pour son faible coût et sa faible consommation d'énergie.

Comme le montre la figure 6.1, ces systèmes sont composés d'un objet connecté appairé et connecté à un *smartphone* avec une application qui communique avec un serveur central. Lorsque l'objet n'est pas connecté, il émet périodiquement des paquets d'avertissements(ADV_IND). Les applications des utilisateurs vont effectuer périodiquement des *scans* actifs pour récupérer l'identifiant des objets connectés cibles à portée et envoyer tous les résultats vers le serveur. Lorsqu'un utilisateur perd son objet, le serveur doit être capable d'avoir sa dernière position connue grâce au *crowd-localization*. Cependant, cette méthode soulève un risque pour la vie privée des utilisateurs avec la possibilité de suivre leur mobilité. Si l'identifiant du objet connecté dans les ADV_IND n'est pas brouillé, tout dispositif *BLE* pourra être capable de détecter la présence d'utilisateurs cibles.

6.3 Menaces actuelles

Comme exemple, nous considérons *TrackR* dont le but est d'aider les utilisateurs à retrouver leurs objets perdus en utilisant soit le *crowd-localization* soit l'application de l'utilisateur lorsque son *smartphone* et son objet connecté sont à portée. Comme décrit dans 6.1, cette architecture est composée d'une application, d'un objet connecté en mode *beacon* et d'un serveur. Le processus d'appariement entre l'objet connecté et le

smartphone de l'utilisateur est effectué une seule fois. Le *smartphone* va ensuite périodiquement réaliser des *scans* actifs en *BLE* pour détecter la présence d'autres objets connectés à portée et envoyer leurs adresses MAC *Bluetooth* au serveur. Le serveur va collecter et sauvegarder ces données. L'objet, lorsqu'il n'est pas connecté au *smartphone*, va périodiquement émettre des *ADV_IND* pour informer de sa présence aux autres périphériques *BLE* à proximité. Lorsque le *smartphone* et l'objet sont connectés, ce dernier peut faire sonner le *smartphone* et vice-versa pour retrouver le *smartphone* ou l'objet connecté.

6.3.1 Attaque 1 : Suivi non-autorisé des utilisateurs

6.3.1.1 Utilisation d'une adresse aléatoire statique

Avec l'outil Ubertooth [106], nous avons collecté les *ADV_IND* envoyés par les objets connectés de *TrackR*. Il apparaît que ces objets connectés utilisent une adresse aléatoire statique. Même si la batterie est remplacée, l'adresse MAC reste la même. Comme cet identifiant unique quasi-statique est périodiquement envoyé par l'objet connecté, il est possible pour un adversaire de suivre la mobilité de l'utilisateur de cet objet. La mobilité des utilisateurs est considérée comme une information privée et sa fuite est une menace sérieuse pour la vie privée de l'utilisateur [122, 139]. Notons que de telles attaques peuvent être exploitées à plus large échelle en utilisant un *botnet BLE* 5.

6.3.1.2 Les adresses aléatoires statiques sont envoyées sans modification au serveur

Nous avons analysé les données échangées entre l'application et le serveur. En utilisant un proxy http, nous étions capables de déchiffrer tous les paquets envoyés entre ces deux entités. Nous avons trouvé que l'adresse MAC de l'objet connecté était envoyé sans modification vers le serveur. En effet, l'application envoie un champ *trackR id* qui est construit de l'identifiant du constructeur composé de 4 zéros (0000), suivi de l'adresse MAC de l'objet connecté à l'envers. Cela implique que les serveurs de *trackR* possèdent toutes les informations sur la mobilité d'un utilisateur en fonction de la position de son objet connecté. Cela pourrait être une menace pour sa vie privée car toutes ces informations privées peuvent être utilisées pour une analyse détaillée, de la classification ou de la publicité ciblée. Il est possible également à ce que l'accès aux données soit accordé à des agences gouvernementales ou à tout adversaire ayant réussi à accéder à la base de données.

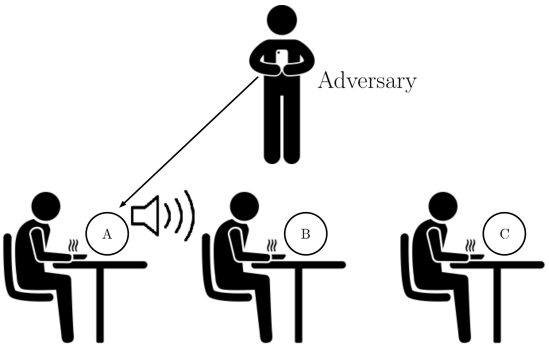


FIGURE 6.2: Scénario où un utilisateur malicieux va faire sonner le objet connecté A pour associer l'adresse MAC de ce dernier à un individu réel.

6.3.2 Attaque 2 : Lier une personne physique à l'identifiant d'un objet connecté

Si un adversaire est dans un restaurant rempli de monde et détecte un ou plusieurs objets connectés, il sait uniquement qu'un individu dans ce restaurant a en possession un objet connecté avec l'adresse MAC A. Si un adversaire peut déterminer qui est le propriétaire de cet objet connecté alors cela ajoute une information précieuse. Nous avons découvert qu'il existe une faille dans la conception de l'objet connecté qui facilite le lien entre l'adresse MAC d'un objet connecté et une personne réelle. Nous supposons seulement que l'attaquant possède un *smartphone* équipé du *BLE* avec une application de type *Light Blue* ou *BLE Scanner* installée. L'adversaire n'a pas besoin d'avoir un *smartphone rooté* et ces applications sont disponibles gratuitement sur le *Google-Play* et l'*App Store*. La faille de sécurité principale est que les objets connectés autorisent un processus d'appariement non-authentifié, ce qui permet à des applications connectées non-authentifiées d'avoir accès à certains attributs de cet objet (informations sur la batterie, nom du périphérique, modèle du constructeur,...). Les objets connectés *trackR* utilisent un service appelé *Immediate Alert Service (IAS)* . Ce dernier permet à un périphérique *BLE* d'envoyer une alerte qui fera sonner l'objet connecté. Dans notre contexte, la permission sur cette valeur est fixée à *écriture sans permission*, ce qui veut dire qu'après s'être connecté à un objet connecté cible, un adversaire peut facilement altérer cette valeur qui fera directement sonner l'objet connecté. Un exemple de scénario est présenté à la figure 6.2.

6.4 *crowd-localization* respectueux de la vie privée

Les services de *crowd-localization* déjà déployés semblent avoir des performances satisfaisantes. Cependant, elles ne protègent pas la vie privée des utilisateurs. Cette sous-section

présente les propriétés qu'un service idéal de *crowd-localization* devrait satisfaire.

6.4.1 Propriétés souhaitées

1. *crowd-localization* : Possibilité de retrouver un objet connecté aussi vite que possible, avec la meilleure précision spatiale et temporelle.
2. *Sécurité sur les informations privées* : Les objets connectés *BLE* ne doivent pouvoir être suivis par personne d'autre que son propriétaire. Cela inclut la compagnie qui fournit le service car celle-ci ne peut pas être considérée comme une entité de confiance.
3. *Contraintes de coût* : Le respect des deux premières contraintes ne doit pas se faire au détriment d'une augmentation excessive du coût dans le temps et dans l'espace.
4. *Facilité de déploiement* : La solution devrait autant que possible s'appuyer sur les standards déjà déployés. Sinon, elle ne sera probablement pas utilisée.

6.4.2 Notre proposition : PPCL

Nous proposons une architecture qui préserve les informations privées des utilisateurs et qui satisfait les propriétés listées en 6.4.1. Les caractéristiques sont implémentées sur l'objet connecté, application installée sur les *smartphones* et sur le serveur. Cette section présente les interactions entre les entités de notre architecture.

L'application du *smartphone* est appairée à un objet connecté et établit une connexion sécurisée pour la communication. Lorsque cet objet n'est pas connecté au *smartphone*, il diffuse périodiquement des ADV_IND contenant un identifiant aléatoire résolvable. L'application du *smartphone* recherche périodiquement d'autres objets connectés à proximité et envoie les résultats au serveur. Si l'objet connecté est perdu, l'utilisateur peut demander la dernière position connue au serveur.

6.4.2.1 Sensor

Le comportement des objets connectés peut être décrit comme suit :

1. **Processus d'appariement (voir la section 3.1.4)** : l'objet connecté et le *smartphone* échangent deux clés secrètes : la LTK et l'IRK. La LTK permettra de chiffrer la session actuelle et les futures sessions tandis que l'IRK sera utilisée pour la génération de d'adresse aléatoire résolvable.

2. **Connecté :** Tant que l'objet reste à portée du *smartphone*, il peut rester connecté. Cela permet au *smartphone* de faire sonner l'objet connecté et vice-versa.

3. **Déconnecté ou perdu :** Lorsqu'il n'est pas à portée du *smartphone* de l'utilisateur, l'objet connecté diffuse périodiquement des *smartphone* en utilisant des adresses privées fréquemment mises à jour. Pour permettre la reconnexion automatique, ces adresses doivent pouvoir être résolues. Comme le montre la figure 6.3, ces ADV_IND contiennent un identifiant privé résolvable noté RPI_i^A. En sus du RPI_i^A (on peut considérer que l'adresse MAC qui apparaît dans l'ADV_IND correspond à RPI_i^A), l'ADV_IND contient un champ *UUID* qui identifie le service de *crowd-localization* et permet de savoir que le message doit être traité par l'application. Nous reviendrons plus tard sur les différentes façons de composer RPI_i^A. On peut simplement faire l'hypothèse qu'il est généré à partir de l'*IRK*.

4. **Mise à jour périodique de l'*IRK* :** L'objet connecté met à jour son *IRK* et l'envoie au *smartphone*. Cela se produit uniquement lorsque l'objet est connecté au *smartphone*.

6.4.2.2 Application

Comme le montre la figure 6.3, l'application analyse périodiquement les canaux *Bluetooth* et collecte les ADV_IND des objets connectés à portée. Si le périphérique *BLE* détecté appartient au service de *crowd-localization*, l'application envoie au serveur un 3-tuple (RPI_i^A, Pos, TS) avec RPI_i^A l'identifiant résolvable du périphérique détecté, Pos la position actuelle du *smartphone* et TS l'heure actuelle. Notez qu'un *smartphone* peut uniquement résoudre l'identifiant de son propre objet connecté. Périodiquement, le *smartphone* obtiendra l'*IRK* mis à jour par l'objet connecté. La mise à jour sera envoyée par ce dernier puisque les interfaces de programmation applicatives officielles proposées par *iOS* et *Android* ne permettent pas aux développeurs d'avoir accès à la valeur des clés secrètes (*LTK*, *IRK*). La communication entre l'application et le serveur utilisera *SSL*. La mise à jour de l'*IRK* ne pourra se faire que lorsque l'objet connecté sera connecté au *smartphone*.

6.4.2.3 Serveur

Le serveur reçoit et sauvegarde tous les 3-uplets (contenant les identifiants privés résolvables, notés *RPI*) envoyés par les applications. Même si un attaquant a accès à la base de données, celui-ci ne pourra pas résoudre les *RPI* puisqu'il ne dispose pas de l'*IRK*.

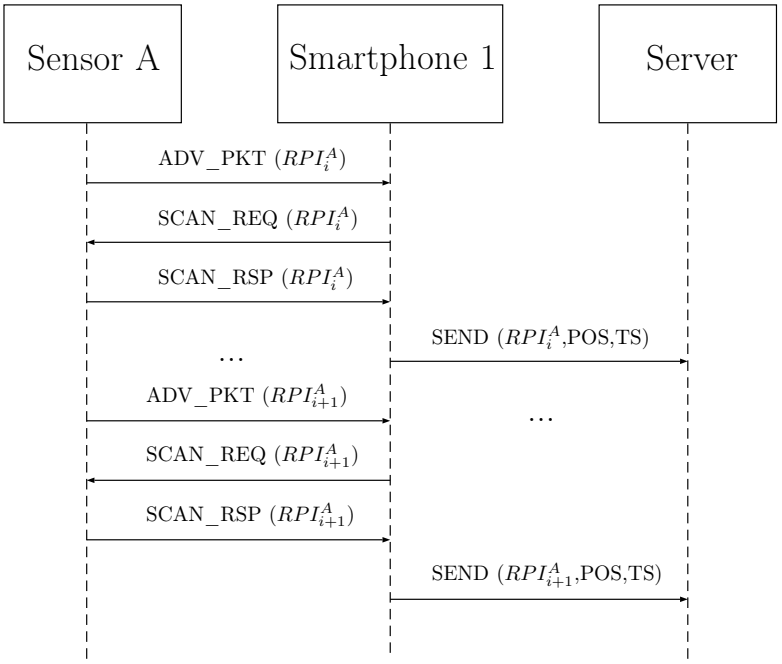


FIGURE 6.3: Interaction entre l'objet connecté, l'application et le serveur où l'application va envoyé au serveur les résultats de son *scan* actif.

Il en va de même pour le propriétaire du service de *crowd-localization*. Lorsque l'utilisateur déclare un objet connecté c comme perdu, l'application du *smartphone* envoie un 3-uplet (IRK_c, Pos_c, TS_l) avec IRK_c l'IRK actuel de l'ojet connecté, Pos_c la position actuelle de l'utilisateur et TS_l le temps de la dernière rencontre entre l' objet connecté et le *smartphone* de l'utilisateur. Le serveur tente de résoudre chaque RPI de la base de données avec IRK_c (nous verrons plus tard comment résoudre un RPI à l'aide d'une IRK). Il essaie d'abord avec le RPI le plus récent qui est le plus proche de la dernière position connue Pos_c. Il s'arrête lorsque les RPI sont plus anciens que TS_l. S'il y a correspondance, le serveur renvoie à l'utilisateur la dernière position connue de l'objet connecté c.

6.5 Analyse des propriétés

Consommation énergétique de l'objet connecté : Une contrainte forte des objets connectés est d'allonger la durée de vie de leur batterie. Dans notre proposition, l'objet connecté n'envoie pas plus d'ADV_IND que les solutions actuellement déployées. Le fait que les objets connectés utilisent une adresse privée résolvable, induit un léger sur-coût par rapport aux adresses statiques. Cependant, le calcul de l'adresse aléatoire résolvable est négligeable par rapport au coût de la transmission d'une trame et est réalisé efficacement par le matériel *BLE*.

Serveur : Lorsqu'un utilisateur demande au serveur de trouver son objet connecté perdu, le serveur tente de résoudre tous les identifiants de la base de données.Dans notre cas, le processus pour trouver l'identifiant correcte est linéaire en fonction de la longueur de la base de données. Ces dernières ont un temps d'exécution logarithmique car il est possible de trier les adresses MAC et le serveur se contente de trouver les entrées correspondant à l'objet connecté perdu d'identifiant MAC_c. Pour améliorer nos performances, nous prendrons en compte la mise à jour de l'*IRK*. Lorsqu'un périphérique envoie un *IRK*, le serveur peut connaître le jour associé à la clé. Par conséquent, il essayera seulement de résoudre les *RPI* dont la date est supérieure ou égal à celle de l'*IRK*. Même avec cette extension, le temps d'exécution est toujours linéaire sur une partie de l'ensemble des données et représente un coût significatif. Il convient alors de classer les entrées par zones géographiques et de résoudre en priorité les *RPI* proches de la zone de Pos_c où l'objet a été vu pour la dernière fois par le *smartphone*. Cette méthode peut rendre le coût de la recherche négligeable s'il existe une entrée pour l'objet connecté dans les zones environnantes. A l'inverse s'il n'y a pas d'entrée, cela nécessite de parcourir toutes les entrées de la base de données dont la date est plus récente que l'*IRK* fourni.

6.6 Différentes versions de notre architecture

Cette sous-section présente les quatre versions de PPCL. Celles-ci se distinguent par leur coût de calcul et de la tolérance aux attaques. Pour apprécier la nécessité de ces différentes versions, nous listons ci-dessous les attaques qui peuvent être faites contre le service de *crowd-localization*, soit pour suivre un objet de manière illégitime, soit pour atteindre l'intégrité du service :

1. Un champ identifiant un utilisateur dans une zone donnée : ADV_PKT ou $SCAN\_RSP$ contiennent souvent des champs identifiant la marque ou le modèle de l'objet connecté. Ces informations sont utiles ou obligatoires pour l'application. Bien que cela puisse sembler inoffensif, ce champ peut être unique parmi les objets connectés présents dans une zone donnée. Par exemple, si un adversaire sait que Bob est le seul à utiliser notre objet connecté à l'université, il pourrait être capable de suivre les mouvements de Bob dans cette zone.

2. Falsification de la position géographique avec une attaque par rejeu : Un adversaire pourrait écouter et stocker un identifiant RPI_i^A valide et recréer des paquets contenant cet identifiant à divers endroits. De même, l'adversaire pourrait modifier l'application du *smartphone* afin qu'il envoie un emplacement falsifié en utilisant un identifiant RPI_i^A légitime. L'adversaire pourrait amplifier ses attaques grâce

à l'utilisation de plusieurs *smartphones* ou même créer de nombreux utilisateurs fantômes fonctionnant sur des machines virtuelles [140].

3. Attaque par déni de service sur le serveur : Un adversaire pourrait surcharger le serveur en envoyant une quantité excessive de requête qui indique la position et l'identifiant des objets connectés. Il pourrait également générer de manière importante de faux emplacements qui rempliraient la capacité de stockage et augmenteraient le coût pour résoudre chaque identifiant des objets connectés.

La protection contre ces attaques a un coût important, que ce soit en complexité de protocole, en calcul ou en espace.

6.6.0.1 Utilisation d'une adresse MAC privée résolvable

La première version utilise l'adresse privée résolvable décrite dans la norme du *Bluetooth* comme identifiant. Pour rappel, une adresse résolvable commence par 01 suivi d'un nombre aléatoire *prand* de 22 bits et du *hash* sur 24 bits de l'IRK et du *prand*.

$$RPI = prand||ah(IRK, prand)$$

Lors de la réception d'un ADV_IND, l'application *smartphone* vérifie si elle utilise une adresse privée résolvable. Si c'est le cas, il enverra l'adresse au serveur ainsi que sa position et le temps actuel. Aucun filtre n'est effectué par l'application du *smartphone*. Les périphériques *BLE* d'autres marques peuvent également être envoyés au serveur. Cela peut nécessiter beaucoup de ressources sur le serveur et augmente le risque de faux positifs. Nous considérons qu'une adresse privée résolvable est un faux positif si elle se situe dans la plage de valeurs valides générées par un objet connecté légitime (voir 3.1.2), c'est à dire si $RPI_{faux-positif} = prand||ah(IRK_c, prand)$ avec IRK_c l'IRK de l'objet que l'on souhaite retrouver. Cela peut être le cas si il existe dans la base de donnée, un RPI ayant été généré par une IRK_{fp} dont le hash avec *prand* produit le même résultat que IRK_c, c'est à dire si il existe dans la base de données IRK_{fp} tel que $ah(IRK_{fp}, prand)||ah(IRK_c, prand)$.

Cette méthode est également sensible à la falsification de positions avec des attaques de rejeu. En effet, rien n'empêche un attaquant de copier de l'ADV_IND et de le rejouer à des endroits ciblés. Pour s'en protéger, il faudrait que lorsque le serveur résout un 3-uplet (RPI_i^A, Pos, TS), celui-ci ne conserve que le premier 3-uplet reçu lorsqu'il existe plusieurs 3-uplets d'entre eux partagent le même *prand*. Pour assurer d'avantage que les répétitions de valeur de *prand* n'apparaissent pas trop souvent pour un même objet,

l'objet connecté peut garder en mémoire les *prand* récemment utilisés et interdire la réutilisation de *prand* tant que la clé *IRK* n'a pas été changée. L'espace de valeur des *prand* peut également être parcouru de manière séquentielle.

6.6.0.2 Utilisation d'une adresse MAC privée résolvable et d'un identifiant représentant la marque ou le modèle de l'objet connecté

L'un des inconvénients majeurs de la version précédente est résulte du fait que tout *ADV_IND* contenant une adresse privée résolvable sera transmis au serveur puis stocké pour une certaine durée. Pour filtrer et conserver uniquement un *ADV_IND* généré par un objet connecté de notre marque, celui doit inclure un champ de données qui est en lien avec la marque et/ou le modèle de l'objet connecté (Ex : « PrivacyPreservingSensorA »). En faisant cela, le serveur ne sera pas surchargé par d'autres adresses MAC résolvables, réduisant ainsi les coûts de calcul, de stockage et de communication. Cela réduira également la probabilité d'avoir des faux positifs.

L'inconvénient est que nos objets connectés seront traçables dans les zones où cet identifiant est unique (*par exemple* « PrivacyPreservingSensorA ») (voir la première attaque dans 6.6). En effet, si Bob est le seul à l'université à utiliser cet objet connecté qui émet périodiquement des *ADV_IND* contenant une valeur unique, un adversaire pourrait suivre sa mobilité en collectant ces données.

Par ailleurs, cette méthode est sensible aux attaques de falsification de position géographique avec des attaques par rejeu. En effet, l'identifiant du service de *crowd-localization* (*e.g. 'PrivacyPreservingSensorA'*) n'est pas protégé et ne change pas au cours du temps. En conséquence, cette version est aussi sensible aux attaques par déni de service en utilisant de manière excessif des attaques de falsification de position géographique.

6.6.0.3 Utilisation d'un *UUID* personnalisé (identifiant unique) sur 128 bits

Cette troisième version ne s'appuie pas sur une adresse privée résolvable pour identifier un objet connecté. Au lieu de cela, un champ *UUID* de 128 bits est ajouté aux *ADV_IND*. La génération de cet identifiant est décrit par la figure 6.4.

Le *smartphone* génère périodiquement une clé appelée $SSKey_i$. Cette clé est envoyée à l'objet connecté. L' objet connecté calcule les 64 premiers bits de l'*UUID* comme suit :

$$UUID[0,63]_i = ah(SSKey_i, prand)||prand$$

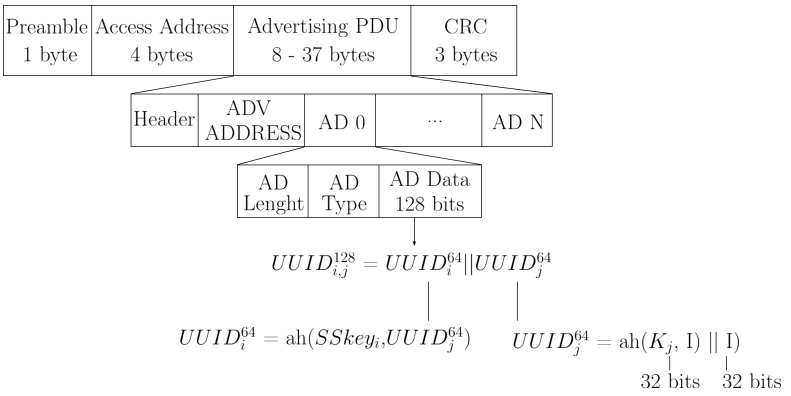


FIGURE 6.4: Schéma d'un *ADV_IND* contenant un *UUID* comme identifiant. $SSKey_i$ est générée par l'application. K_j est générée par le serveur. La valeur I est un nombre aléatoire. *ah* est une fonction de hachage.

où *prand* est un nombre aléatoire et *ah* est une fonction d'adressage aléatoire. Cette première moitié de l'*UUID* permet à quiconque avec la $SSKey$ d'identifier l'objet connecté. En effet, cette clé est nécessaire pour résoudre les différents *UUID* collectés et permettra de savoir quels identifiants parmi un ensemble appartiennent à un même périphérique *BLE*. En comparaison avec la norme officielle *BLE*, cette clé a une fonction quasi semblable à l'*IRK* (voir 3.1.4).

Les 64 bits de la seconde partie de l'*UUID* ont pour but de filtrer les périphériques qui appartiennent à notre marque d' objet connecté sans avoir le risque d'être suivi par un attaquant dans les zones où les objets connectés de notre marque sont rares. Ces 64 bits sont générés de la manière suivante. Le serveur génère périodiquement une clé appelé K_j qui est envoyée aux différents *smartphones* qui à leur tour partagent cette clé aux objets connectés. La formule est décrite comme suit :

$$UUID[64, 127]_j = ah(K_j, prand_s)||prand_s$$

où $prand_s$ est un nombre aléatoire. Enfin, $UUID_{i,j}$ est la concaténation de $UUID[0, 63]_i$ et $UUID[64, 128]_j$.

Lorsque l'application d'un *smartphone* reçoit un *ADV_IND* contenant $UUID_{x,y}$, celui-ci vérifie d'abord si $UUID_{x,y}[64 - 127]$ a été généré en utilisant une des clés récentes K_y générées par le serveur. Si c'est le cas, alors cela veut dire que cet *ADV_IND* a été construit par un objet connecté du service de *crowd-localization* et que l'identifiant de cet objet connecté doit être envoyé vers le serveur.

Lorsqu'un objet connecté est perdu, l'application va envoyer la dernière valeur de $SSKey_i$ et la position géographique actuelle de l'utilisateur Pos_x.

Notons que dans cette version, le risque de faux positif est négligeable.

Si notre objet connecté est entouré par d'autres périphériques émettant des *UUID* de 128 bits, ce dernier ne sera pas traçable, a moins qu'un adversaire possède l'information sur la plus récente clé K_j générée par le serveur. Si dans une zone donnée, un utilisateur du service de *crowd-localization* est le seul à utiliser un *UUID* de 128 bits, l'*UUID* pourrait être obfusqué de sorte que le contenu des *ADV_IND* soit similaire aux objets connectés populaires.

Enfin, le dernier point concerne le développement réel de cette solution. L'utilisation d'*UUID* comme identifiant est compatible avec l'API *BLE* sur *Android* et *iOS*.

Pour résumer, cette troisième version est résistante aux attaques de falsification de position par rejeu et de déni de service car l'adversaire doit avoir connaissance de la clé K. Même si c'était le cas, cette clé n'est pas unique et est mise à jour périodiquement par le serveur, ce qui permettrait à un attaquant d'avoir accès à la mobilité d'un utilisateur uniquement durant la durée de vie d'une clé K. En plus de nécessiter un développement supplémentaire qui s'éloigne de ce que propose le standard *Bluetooth*, l'inconvénient principal de cette version est qu'un attaquant peut suivre la mobilité d'un utilisateur si celui-ci est le seul à avoir un objet connecté de notre marque dans une zone dû à l'unicité de la valeur de l'identifiant. Il est possible d'atténuer ce risque en plaçant l'*UUID* dans un champ de donnée fréquemment utilisé par de nombreux objets connectés populaires pour essayer de « camoufler » cet identifiant.

6.6.0.4 Utilisation d'une adresse privée résolvable personnalisée

Comme vu sur la version précédente, en dépit d'avoir un identifiant apparemment aléatoire pour un adversaire, le fait que notre *ADV_IND* inclut un *UUID* de 128 bits peut être une menace dans les zones où les périphériques émettant des *UUID* de 128 bits sont rares. Cette version propose de masquer à la fois l'identifiant résolvable du périphérique et l'identifiant résolvable du service de *crowd-localization* dans les 48 bits d'une adresse *BLE* privée résolvable. Le calcul de l'adresse est le même que celui décrit dans la documentation officielle à l'exception de la génération de `prand`. Le serveur génère périodiquement une clé appelée K_j envoyée au *smartphone* et aux objets connectés. `prand` sera généré comme suit :

$$prand = ah(K_j, P)||P$$

où P est généré aléatoirement. L'ensemble du processus est décrit dans la figure. 6.5.

Notez que cette version nécessite une expérimentation plus détaillée pour trouver le meilleur compromis entre la période de mise à jour K_j, qui influence le nombre de tests à faire par une application pour décoder l'identifiant après chaque *scan*, et l'espace de P

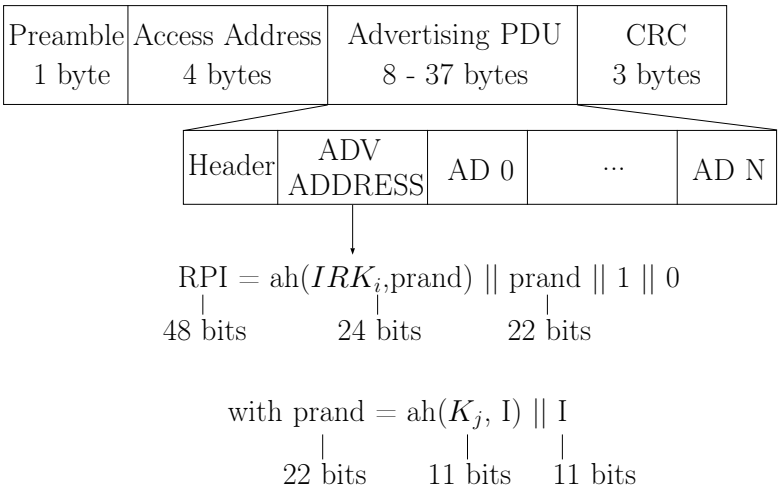


FIGURE 6.5: Schéma d'un *ADV_IND* contenant une adresse privée résolvable personnalisée comme identifiant. K_j est générée par le serveur. La valeur P est un nombre aléatoire. *ah* est une fonction de hachage.

pour ne pas avoir deux fois la même valeur pendant la durée de vie d'une clé générée par le serveur. Cette version est résistante aux attaques de falsification de la position et aux attaques de déni de service grâce à la clé K_j. Même si l'adversaire a connaissance de la valeur de cette clé, celle-ci est changée périodiquement par le serveur et dans le pire des cas, l'adversaire aura accès a la mobilité des utilisateurs uniquement pendant la durée de vie de cette clé. Concernant un développement réel de cette version, cette dernière ne fonctionnera que pour les utilisateurs sous *Android* puisque l'API *BLE* sur *iOS* ne donne pas accès aux adresses des *ADV_IND* collectées. Ainsi, dans ce cas, et comme dans les déploiements actuels qui ne préservent pas la vie privée, seuls les utilisateurs sous *Android* contribueront au *crowd-localization*.

6.7 Conclusion

Ce chapitre a mis en exergue les menaces induites par les services de *crowd-localization*. Les objets connectés distribués par ces derniers utilisent des adresses publiques qui, comme on l'a vu dans le chapitre précédent, permet à des attaquants de suivre facilement la mobilité de l'objet connecté. L'analyse des informations remontées par les sondes des *smartphones* vers le serveur d'un service de *crowd-localization* a montré que les adresses MAC relevées sont envoyées en clair et que le serveur sait donc à tout moment où se trouvent les objets connectés, où se trouvent les *smartphones* des utilisateurs et quels *smartphones* ont vu quels objets connectés. Si les serveurs des services de *crowd-localization* conservent ces informations, cela signifie qu'ils ont accès à la mobilité des

utilisateurs et à leurs liens sociaux, des informations très sensibles qui permettent d'en inférer bien d'autres.

PPCL, l'architecture que nous avons proposée permet de palier ces défauts. Les quatre variantes ont comme propriété commune de ne pas utiliser d'identifiant statique ou unique dans les *ADV_IND* (ce qui permet de ne pas être suivi par une attaque de type BLEB) et de ne pas pouvoir être tracé par le serveur, sauf lorsque l'attaquant déclare son objet comme perdu. La variante 6.6.0.2 utilise une adresse MAC résolvable comme identifiant privé ainsi qu'un *UUID* qui identifie le service de *crowd-localization*. C'est la méthode qui présente le meilleur compromis entre coût et facilité d'implémentation. Cependant, l'*UUID* qui identifie le service permet à un attaquant de suivre l'objet si le service n'est pas populaire ou si l'objet connecté est dans une zone où il est le seul à utiliser ce service. Notons que ce problème est commun à tous les périphériques *BLE* qui envoient des *ADV_IND*. Pour s'en protéger, nous avons néanmoins proposé les variantes 6.6.0.3 et 6.6.0.4 qui remplacent l'*UUID* par un identifiant de service résolvable et contient pas de champs ou valeur spécifique au service de *crowd-localization*.

Ce chapitre met clairement en lumière le fait qu'il est possible de déployer des services de *crowd-localization* respectueux de la vie privée sans impacter les performances du service, en utilisant les standards du *BLE* et sans générer de sur-coût significatif sur les serveurs ou les applications *smartphones*.

Chapitre 7

Conclusion et perspectives

Cette thèse étudie le *Physical Tracking*, l'étude de la mobilité des utilisateurs grâce aux informations émises par les interfaces sans fil de leurs *smartphones* ou objets connectés.

Elle comporte deux parties, la première présente les technologies *PAN* et *WAN*, l'état de l'art des méthodes de suivi et les contre mesures adoptés. La deuxième partie présente les contributions qui visent à proposer de nouvelles méthodes de suivi, analyser les performances de celles-ci face aux méthodes existantes et dans le cas particulier de l'application de *crowd-localization*, à proposer des méthodes de suivi respectueuses de la vie privée.

Il en résulte qu'après avoir analysé les différentes méthodes de suivi de mobilité avec les technologies de communication sans fil (*WiFi*, *Bluetooth*, *BLE*, réseaux cellulaires), le risque pour la vie privée des utilisateurs est bien réel. On peut néanmoins noter une prise de conscience et les problématiques liées à la sécurité et à la vie privée des utilisateurs sont désormais considérées. Ainsi, une meilleure utilisation des normes *Bluetooth*et *WiFi* ont été effectuées par les développeurs au cours de ces dernières années pour assurer la sécurité de ces données privées. Par exemple, en *Bluetooth*, certains périphériques ne restent que peu de temps en mode découvrable. De plus, les *OS* utilisent maintenant des adresses MAC *WiFi* aléatoires qui changent périodiquement dans les *probes requests*.

Cependant, malgré la bonne utilisation et la bonne configuration de ces périphériques, l'état de l'art indique qu'il demeure un risque résiduel lié à l'utilisation du *WiFi* et *classic Bluetooth*. Par exemple, un périphérique *WiFi*, une fois associé à un point d'accès, divulgue en clair son adresse MAC réelle. Par ailleurs, les attaques temporelles (*timing attacks*) peuvent aussi être utilisées en analysant la fréquence d'envoi des *probes requests* des périphériques *WiFi*. Notons néanmoins que ces attaques nécessitent de déployer des

moyens bien plus important qu'auparavant, qu'elles sont détectables ou qu'elles génèrent des empreintes peu exploitables.

La partie contribution de cette thèse s'est attachée à étudier d'autres moyens de suivi que ceux couramment employés dans les *PAN* et *WAN* (*classic Bluetooth* et *WiFi*).

Le chapitre 4, a étudié les informations disponibles lorsque les périphériques du *classic Bluetooth* sont connectés. Il en résulte que l'*Access Code* contenu dans l'entête de chaque trame *Bluetooth* est dérivé à partir du *LAP*, les 24 derniers bits de l'adresse MAC. De même le champs de contrôle d'erreur de l'entête ainsi que la séquence de canaux est dérivée à partir de l'*UAP*, les 8 derniers bits de l'*OUI* de l'adresse MAC. Il en résulte que l'on peut obtenir 32 bits de l'adresse MAC publique de l'utilisateur en écoutant le trafic. Nos mesures ont montrés qu'en écoutant sur un seul canal parmi les 79 que compte le *Bluetooth*, le *LAP* (resp. *UAP*) était détecté après au plus $20ms$ (resp. $800ms$) dans 95% des cas lorsqu'il y a du trafic échangé et au plus $10s$ (resp. $100s$) dans 95% des cas lorsqu'il n'y a pas de trafic échangé avec des variabilités en fonction des profils *Bluetooth* et des périphériques. Notons que ces délais peuvent être significativement réduis si l'on écoute plusieurs canaux *Bluetooth* simultanément au lieu d'un seul. Ces performances sont bien meilleures que celles de méthodes de suivi basées sur le service de découverte actif qui étaient auparavant utilisées, le *WiFi* nécessitant entre 30s et 300s pour détecter une interface et le *Bluetooth* nécessitant au moins $10.92s$ pour compléter un *scan*, sans garantie que le périphérique cible ait bien été détecté. Cette méthode de suivi n'a sans doute pas été exploité plus tôt de par la nature du *Bluetooth* qui utilise 79 canaux et dont les périphériques ne peuvent être configurés pour écouter sur un seul canal, ce qui nécessite d'utiliser des outils d'analyse coûteux et par conséquent incompatibles avec le déploiement d'une architecture de suivi. Le développement et la commercialisation de l'*Ubertooth*, un analyseur *Bluetooth open source* à bas coût ($\approx$ 40€) rend désormais possible ce type d'attaque pour suivre les utilisateurs. Ceux qui utilisent des objets connectés en *Bluetooth* (haut parleur / écouteurs sans fil, kit main libre, dispositifs médicaux ...) doivent donc être conscients qu'ils peuvent être suivis dès lors qu'une connexion est établie et ce même si les applications sont inactives et qu'aucune données ne soient échangées.

Le chapitre 5 a étudié les moyens par lesquels le *Bluetooth Low Energy* pourrait être utilisé pour suivre ses utilisateurs. Il en résulte que bien que la norme prévoit des mécanismes permettant de protéger la vie privée des utilisateurs, ceux-ci sont rarement utilisés. Ainsi, les adresses MAC contenues dans les *ADV_IND* périodiquement envoyées par les objets connectés sont soit publiques, soit aléatoires statiques et constituent des identifiants. Les mesures effectuées ont montré que ces identifiants peuvent être perçus par un attaquant en moins de $1.5s$ dans 95% des cas, ce qui constitue une menace bien plus importante que

les méthodes utilisées auparavant pour le *WiFi* et le *classic Bluetooth*. Ce qui est apparu comme bien plus préoccupant, c'est qu'à l'inverse du *WiFi* et du *classic Bluetooth*, ces informations peuvent être collectées depuis un *smartphone*, sans qu'il ait besoin d'être *rooté* ou modifié. Cela permet à un attaquant de déployer de nombreux points de mesures, simplement en installant une application sur les *smartphones* d'utilisateurs normaux. De plus ces applications, ou tout du moins le code qui effectue les *scans BLE*, peuvent être installées à l'insu des utilisateurs. Les *botnets BLE* (que nous avons appelé BLEB) ainsi formés permettent un suivi des utilisateurs à une échelle, une précision et avec une ubiquité sans précédent. Les utilisateurs qui possèdent des objets connectés doivent être conscients qu'ils sont sensibles à ces attaques. Il faut qu'ils veillent à ce que leurs objets utilisent des adresses privées résolvables ou aléatoires non statiques. A l'heure actuelle, rares sont les périphériques *BLE* qui utilisent ce type d'adresses.

Le dernier chapitre (chap. 6) adresse les services de *crowd-localization*, c'est à dire les services de suivi d'objets connectés qui utilisent les applications déployées sur les *smartphones* d'une communauté d'utilisateurs pour localiser des objets, les suivre et les retrouver lorsqu'ils sont perdus. La stratégie utilisée par ces services est similaire à celle des BLEB décrits dans le chapitre 5, a l'exception qu'ils ne sont pas déployés à des fins malicieuses et qu'ils ciblent un sous ensemble de périphériques bien identifiés. Nous avons montré que les objets vendus pour la *crowd-localization* utilisent des adresses publiques ou privées mais statiques et soumettent donc les utilisateurs au risque de suivi par les BLEB. Nous avons également montré que les adresses MAC relevées par les applications sont envoyées en clair et que le serveur sait donc à tout moment où se trouvent les objets connectés, où se trouvent les *smartphones* des utilisateurs et quels *smartphones* ont vu quels objets connectés. A travers PPCL, notre proposition de *crowd-localization* respectueuse de la vie privée, nous avons montré que ces services font subir un risque inutile à leur utilisateur. En effet notre proposition n'impacte pas les performances du service rendu, peut être déployé sans modification des standard du *BLE* et n'augmente pas significativement les coûts de développement, de traitement ou de stockage.

7.1 Perspectives

Plusieurs technologies sans fil pouvant être utilisées pour le suivi des utilisateurs n'ont pas été abordées dans cette thèse et pourront faire l'objet de travaux futurs.

En effet, les réseaux dédiés à l'*IoT* tels que *LoRa, SigFox, Nb-IoT, LTE-M* sont actuellement en cours de déploiement. La portée de ces radios laissent présager la possibilité de suivre des objets sur une zone importante avec peu de capteurs. Le faible coût et la faible capacité des objets amènera probablement les développeurs à négliger les problématiques

de sécurité et vie privée. Les normes et implémentations de ces technologies ne seront sans doute pas non plus dépourvues de failles exploitables.

Similaire au *Bluetooth*, la technologie *ANT/ANT+* est beaucoup utilisée dans les objets connectés pour la santé et le sport. Elle n'a pas fait l'objet d'étude approfondie concernant la possibilité de suivre ses utilisateurs.

Le *Bluetooth Low Energy* mérite également une attention particulière en ce qui concerne les informations embarquées dans les *ADV_IND* ou *SCAN_RSP*. En effet, cette thèse s'est focalisé sur l'identifiant que constitue l'adresse MAC mais il apparaît clairement que de nombreuses informations potentiellement identifiantes sont accessibles et exploitables par des BLEB. La composition des adresses résolvables mérite également d'être surveillée par la communauté, que ce soit pour la possibilité d'effectuer une attaque de force brute pour recouvrir l'*IRK* ou la fréquence avec laquelle le *prand* est renouvelé.

De manière général, il reste une grande quantité d'études à mener afin de couvrir toutes les technologies sans fil liées au *PT*. De plus, c'est un effort à fournir de façon périodique pour mettre à jour l'évaluation des menaces sur la vie privée des utilisateurs avec les technologies contemporaines.

Troisième partie

Annexes

Outils pour le WiFi

Ce chapitre présente les différents outils permettant de collecter des informations dans un contexte de suivi de mobilité des utilisateurs. Ces informations sont présentées ici afin de rendre compte de la facilité avec lesquelles ces données peuvent être collectées. Elles ne sont que rarement présentées dans la littérature, en tout cas à des fin de *Physical Tracking*, en particulier dans le cas du *classic Bluetooth* et du *Bluetooth Low Energy*.

Pour chaque outil présenté, un cas d'utilisation sera décrit accompagné d'une commande de base et d'une explication.

7.2 Carte réseau *WiFi*

Une carte réseau *WiFi* dispose de plusieurs modes :

Managed **:** Le périphérique devient client du point d'accès auquel il est connecté

Monitor **:** Permet de collecter toutes les trames présentes sur le canal y compris les trames de *management*

Ad-hoc **:** Permet à un périphérique de se connecter et de communiquer sans l'aide d'un point d'accès

Access Point **:** L'interface devient point d'accès

Dans notre cas, nous nous intéresserons uniquement au mode *monitoring*.

7.2.1 Configuration

Pour utiliser le mode *monitoring* d'une carte *WiFi*, nous pouvons soit utiliser la commande *iwconfig/ifconfig*, soit passer par l'utilitaire *airmon-ng*.

Il faut tout d'abord définir l'interface *WiFi* à utiliser. Pour obtenir la liste des interfaces *WiFi* ainsi que les mode supportées :

```
iwconfig
```

Ensuite, il faut activer cette interface :

```
ifconfig nom_interface up
```

Enfin, il faut activer le mode *monitoring/promiscious* de l'interface choisie :

```
iwconfig nom_interface mode Monitor
```

Il est aussi possible d'effectuer ces opérations avec *airmon-ng*. La commande suivante permettra à l'interface *WiFi* de collecter tous les paquets *WiFi* sur le canal 6 :

```
airmon-ng start nom_interface 6
```

7.2.2 Capture des *probes requests*

Pour rappel, les *probes requests* sont des trames émises périodiquement par les périphériques *WiFi* lorsqu'ils exécutent un service de découverte de points d'accès à portée (voir 1.1.3.2).

Lorsque l'interface *WiFi* est active en mode *monitoring*, ce dernier collecte toutes les trames qu'il voit sur un canal. En effet, une carte réseau ne peut écouter que sur un seul canal. Pour une écoute multi-canaux, il faudrait avoir plusieurs cartes *WiFi* associées à différents canaux parmi les 11 disponibles.

Pour exécuter la capture, il est possible d'utiliser *tshark*, un logiciel en ligne de commande permettant d'analyser les paquets capturés depuis une interface *WiFi*. Une commande *tshark* permettant de filtrer les paquets est décrite ci-dessous :

```
sudo /usr/bin/tshark -i nom_interface -R "wlan.fc.type ==
0x0" -T fields -e wlan.sa -e wlan.da -e wlan_mgt.ssid -e
wlan.fc.type -e wlan.fc.type_subtype -e radiotap.flags.fcs  -e
radiotap.dbm_antsignal -e radiotap.channel.freq -E
separator=";"
```

où :

- L'option *-i* va permettre de sélectionner l'interface nom_interface pour la capture.

- L'option -*R* est utilisé comme filtre. Ici on ne collecte que les paquets de type "0x0", c'est-à-dire les trames de managements (*beacons*, *probes requests*,...)
- L'option -*T* indique le format de sortie des paquets décodés.
- L'option -*e* indique les champs qu'on veut garder pour chaque trame.
    1. *wlan.sa* fait référence à l'adresse source.
    2. *wlan.da* fait référence à l'adresse de destination.
    3. *wlan_ mgt.ssid* fait référence au nom du réseau des trames de managements :
    4. *wlan.fc.type* et wlan.fc.subtype font référence au type et au sous-type de la trame collectée.
    5. *radiotap.flags.fcs* indique que la somme de contrôle d'erreur a detecté une erreur.
    6. *radiotap.dbm_ antsignal* spécifie si le puissance du signal de l'antenne en dBm est présente.
    7. *radiotap.channel.freq* spécifie la fréquence du canal d'où provient la trame collectée.
- L'option -*E* indique le séparateur entre chaque champ (ici le " ;").

Cas d'utilisation : Cette méthode est efficace pour suivre passivement la mobilité des utilisateurs grâce aux informations émises par leur *smartphone*. L'utilisation de ces utilitaires sont disponibles sur l'ensemble des périphériques Linux et permettent très simplement de détecter la présence de périphériques *WiFi*. De ce fait, si l'on dissémine plusieurs points d'écoute à l'échelle d'un campus, il devient possible de retracer une partie de la mobilité des périphériques détectés. Cependant, Cunche et al. [91, 141] ont démontré qu'il est possible d'associer une empreinte unique à un utilisateur malgré l'utilisation d'adresses MAC aléatoires grâce aux autres informations contenues dans les paquets de managements ou au timing d'envoi des *probes requests*. De ce fait, l'écoute passive via les cartes *WiFi* peut rester viable dans un contexte de suivi des utilisateurs.

7.2.3 Création de faux points d'accès

L'utilisation d'adresse MAC aléatoires pose un frein au suivi de la mobilité des utilisateurs. Il est possible de contourner les adresses MAC aléatoire en incitant les utilisateurs à se connecter à un faux point d'accès et ainsi révéler leur adresse MAC réelle. Par ailleurs, cette méthode est aussi applicable pour la falsification de la position géographique lorsque celle-ci est obtenue via *WPS* (*WiFi Positionning System*). Un *WPS* calcule la position

d'un utilisateur par trilateration sur les points d'accès *WiFi* qu'il observe. Dans [68], des chercheurs ont générés de faux AP afin de fausser la position obtenue par les WPS. Cette position est utilisée par les applications de réseaux sociaux et se retrouve publiquement sur internet. Par effet de bord, le *WPS spoofing* permet d'associer l'adresse adresse MAC observée par le faux AP à une position géographique d'un post de réseau social et donc un profil et son propriétaire.

La création d'un faux point d'accès *WiFi* se fait grâce à l'utilitaire *airbase-ng*. Il faut au préalable configurer une carte *WiFi* en mode *monitoring*. La commande pour créer un point d'accès est la suivante :

```
airbase-ng -e MyFakeAP -c 6 nom_interface
```

où :

- L'option *-e* spécifie le nom du point d'accès crée (MyFakeAP)
- L'option *-c* spécifie le canal *WiFi* à utiliser (le canal 6)
- *nom_interface* le nom de l'interface *WiFi* en mode *monitoring*

Il faut également configurer un serveur DHCP pour fournir une adresse IP dynamique aux utilisateurs qui se connectent à notre faux points d'accès. Pour se faire, nous utilisons le serveur `dhcp` par défaut d'Ubuntu :

```
dhcpd -d -f -cf dhcpd_ap.conf nom_interface &
```

où :

- L'option *-d* pour afficher la description des erreurs standards
- L'option *-f* pour lancer le processus en tâche de fond.
- L'option *-cf* spécifie le fichier de configuration à utiliser (dhcpd_ap.conf)
- *nom_interface* le nom de l'interface *WiFi* en mode *monitoring*

Le fichier de configuration (dhcpd_ap.conf) est constitué des éléments suivants : Le faux point d'accès est maintenant créé et les utilisateurs peuvent s'y connecter.

La génération de faux points d'accès peut également se faire comme un clone d'un point d'accès légitime. La première étape est de sonder les canaux *WiFi* pour avoir les informations sur les points d'accès à portée. Ces dernières contiennent l'adresse MAC du point de d'accès et son nom. Il suffit par la suite, de lancer la commande :

```
airbase-ng -a XX:XX:XX:XX:XX:XX --essid "MyAPToCopy" -c 11 nom_interface
```

où :

- L'option *-a* spécifie l'adresse MAC à copier pour le faux point d'accès
- L'option *-essid* spécifie le nom du point d'accès à copier
- L'option *-c* spécifie le numéro du canal du point d'accès qu'on veut cloner
- *nom_interface* le nom de l'interface *WiFi* en mode *monitoring*

La création de faux point d'accès *WiFi* permet de récupérer l'adresse MAC *WiFi* réelle de tous les périphériques qui s'y connectent. Par ce biais, cette méthode permet d'outrepasser le fait que les périphériques utilisent des adresses MAC aléatoires contenues dans les *probes requests*.

7.3 Interfaces de programmation applicatives sous *iOS* / *Android*

Nous nous intéressons aux API liées au *WiFi* et au *GPS*. Nous étudierons les permissions requises pour qu'une application mobile puisse utiliser ces interfaces de programmation avant de lister quelques exemples de méthodes permettant de collecter l'identifiant du *smartphone* par le biais de son adresse MAC, la configuration du réseau auquel il est connecté, la liste des réseaux à portée ou encore sa position géographique actuelle.

7.3.1 Permissions

Les permissions accordent des privilèges aux applications mobiles qui en ont l'accès. Ces dernières sont contrôlées par l'utilisateur qui doit donner son accord si une application veut interagir avec certains capteurs de son *smartphone*. Par exemple, l'utilisateur peut autoriser une application de réseau social à utiliser sa caméra, ce qui permet de prendre et de télécharger des photos. De manière générale, deux types de permissions existent : les permissions normales et celles qui sont considérées dangereuses. La première regroupe toutes les interfaces de programmation applicatives qui ne présentent aucun danger pour la vie privée de l'utilisateur. La seconde donne accès aux informations confidentielles d'un utilisateur (sa liste de contacts, ses messages, ses photos ...). Cette classification est réalisée par *iOS* et *Android*. Certaines permissions seront affichés à l'installation de l'application et d'autres au lancement de celle-ci. Il faudra obligatoirement une interaction avec l'utilisateur pour autoriser les applications à interagir avec les différentes interfaces de programmations.

***Android* :** Pour ce système d'exploitation, chaque application mobile possède un fichier spécial appelé *manifest.xml*. Ce dernier regroupe la liste des permissions que l'application va demander à l'utilisation, lors de son installation ou au lancement de celui-ci.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
```

La première permission permet de créer une *socket* réseau. La seconde donne accès aux informations sur les réseaux/points d'accès à portée. Enfin, la troisième permission donne le droit de changer l'état de l'interface du *WiFi*.

***iOS* :** Pour le système d'exploitation d'Apple, Un fichier d'informations appelé *info.plist* est utilisé pour avertir l'utilisateur que l'application a besoin d'accéder à une interface sans fil du *smartphone*. L'exemple ci-dessus indique l'utilisation du *WiFi* pour permettre un bon fonctionnement de l'application.

```
<key>UIRequiresPersistentWiFi</key>
<string>true</string>
```

Cette permission permet à l'application de maintenir l'interface *WiFi* activé même si le *smartphone* est inactif. Dans notre contexte, il peut être intéressant de l'utiliser pour remonter périodiquement des informations sur l'AP auquel est connecté le *smartphone*. *iOS* ne possède pas de permissions spécifiques pour interagir avec les interfaces de programmation du *WiFi*. Cependant, il restreint la possibilité aux développeurs d'utiliser certaines méthodes des ces interfaces. Par exemple, il n'est pas possible de lister les points d'accès *WiFi* à porté de l'utilisateur.

7.4 Interfaces de programmation applicatives

Les interfaces de programmation applicatives du *WiFi* permettent d'accéder à certaines données comme par exemple l'adresse MAC de l'interface *WiFi* qui représente dans notre contexte un identifiant unique. Cette sous-section étudiera trois cas : la faisabilité de la collecte de cet identifiant au travers de l'utilisation des interfaces de programmation proposées, la possibilité ou non de sonder les canaux *WiFi* pour détecter la présence de périphériques à portée et de collecter leur identifiant et les méthodes à utiliser pour récupérer la position géographique du *smartphone*. Chaque cas sera analysé selon le système d'exploitation d'*Android* et d'*iOS*.

7.4.1 *Android*

L'utilisation de la classe *WifiManager* fournit par l'interface de programmation permet de gérer différents aspects d'une connectivité avec le *WiFi*.

7.4.1.1 Obtention de l'adresse MAC *WiFi* du *smartphone*

Premièrement, il est possible de récupérer l'adresse MAC *WiFi* comme suit :

```
WifiManager manager = (WifiManager) getSystemService(Context.WIFI_SERVICE);
WifiInfo info = manager.getConnectionInfo();
String address = info.getMacAddress();
```

Cette version du code fonctionne avec toutes les versions d'*Android* antérieur à la version 6.0. A partir de cette dernière, la méthode d'accès à l'adresse MAC a été retirée par *Android* pour apporter plus de sécurité sur ces informations liées à la vie privée des utilisateurs. La valeur de l'adresse MAC par défaut renvoyée est 02 :00 :00 :00 :00 :00.

Cependant, une alternative existe pour accéder à cette valeur en utilisant des combinaisons d'autres interfaces de programmation publiques proposées. L'idée derrière ce code est de lister toutes les interfaces réseaux liées au *smartphone* et de trouver celle qui correspond au *WiFi* (dans cet exemple, *wlan0* a été utilisé) :

```
public String getMacAddr() {
       try {
        // Lister toutes les interfaces reseaux
           List<NetworkInterface> all =
           Collections.list(NetworkInterface.getNetworkInterfaces());
           for (NetworkInterface nif : all) {
            // Si le WiFi exite,
               if (!nif.getName().equalsIgnoreCase("wlan0")) continue;

               // On recupere la valeur de l'adresse MAC reelle en bytes
               byte[] macBytes = nif.getHardwareAddress();

               // Si elle n'existe pas, on quitte la methode
               if (macBytes == null) {
                   return "";
               }
```

```
                // Sinon, on convertit cette valeur en chaine de caracteres
                StringBuilder res = new StringBuilder();
                for (byte b : macBytes) {
                    res.append(Integer.toHexString(b & 0xFF) + ":");
                }

                if (res.length() > 0) {
                    res.deleteCharAt(res1.length() - 1);
                }

                // Et, on retourne le resultat
                return res.toString();
            }
        } catch (Exception ex) {
        }
        //Sinon, on renvoit la valeur par defaut
        return "02:00:00:00:00:00";
    }
```

7.4.1.2 Obtention de l'adress MAC de point d'accès à portée

Une fonctionnalité intéressante serait de pouvoir sonder les canaux *WiFi* et de lister les AP à portée identifiés par leur adresse MAC. Il est possible de le faire grâce au code suivant :

```
 WifiManager manager = (WifiManager) getSystemService(Context.WIFI_SERVICE);

// On lance le scan
 manager.startScan();

// On recupere les resultats
 results = wifi.getScanResults();

// Pour chaque point d'accès a portée, on affiche l'adresse MAC et le nom du reseau
 for (ScanResult result : results) {
Log.v("Test", "Address : "+result.BSSID+" and SSID : "+result.SSID);
}
```

Les récentes mise à jour des versions d'*Android* requièrent désormais deux permissions supplémentaires pour accéder à ces valeurs : les permissions *ACCESS_ COARSE_ LOCATION* et/ou *ACCESS_ FINE_ LOCATION*. En effet, le résultat des *scan WiFi* peut être exploité par des services *WPS* (*WiFi Positionning System*, qui par trilateration des points d'accès, permettent d'avoir de localiser le *smartphone* de manière plus précise que le *GPS* dans un environnement urbain. Ainsi toute application ayant accès aux *scans WiFi* peut avoir accès à la position de l'utilisateur. Il était donc devenu nécessaire de contrôler l'accès à ces informations et d'avertir l'utilisateur sur les conséquences de l'autorisation *ACCESS_ WIFI_ STATE*.

Google fourni un service de *WPS*. L'exemple ci-dessous permet d'obtenir la position du *smartphone* via *WPS*, sans avoir recours au capteur *GPS*. Les seules interfaces utilisées seront le *WiFi* et le réseau cellulaire.

```
 // Recuperer une instance d'un objet permettant d'interagir avec la position et les mo
LocationManager locationManager = (LocationManager) this.getSystemService(Context.LOCAT

// Lancement du processus de mise a jour periodique de la position
 String locationProvider = LocationManager.NETWORK_PROVIDER;
locationManager.requestLocationUpdates(locationProvider, 0, 0, locationListener);

// Recuperer la derniere position connue
 Location lastKnownLocation = locationManager.getLastKnownLocation(locationProvider);
```

7.4.2 *iOS*

Depuis la version d'*iOS* 7.0, les interfaces de programmation du *WiFi* ne possèdent plus de méthode permettant de récupérer l'adresse MAC *WiFi* réelle du *smartphone* pour des raisons évidentes de sécurité. Même si les développeurs utilisent les méthodes existantes, une valeur par défaut (02 :00 :00 :00 :00 :00) sera toujours renvoyée.

Pour pallier ce problème, Apple a mis en place l'utilisation d'un *UUID (Universally Unique Identifier*, RFC4122), un identifiant codé sur une séquence de 128 bits qui peut garantir l'unicité selon l'espace et le temps. Cependant, pour une sécurité toujours plus forte, Apple rejette toute application de son store qui utilise cette valeur. Le système a ainsi définie deux types d'*UUID* mise à disposition des développeurs : l'identifiant vendeur (*a.k.a. vendor*) et l'identifiant publicitaire.

7.4.2.1 Identifiant vendeur - *Vendor specific UUID*

La valeur de l'identifiant vendeur est la même pour les applications provenant du même fournisseur et qui s'exécute sur un même *smartphone*. Une valeur différente sera renvoyée pour toutes les applications sur le même *smartphone* provenant de différents vendeurs et pour les applications sur différents smartphones quel que soit le fournisseur. Enfin, cette valeur change lorsque l'utilisateur supprime toutes les applications de ce fournisseur de l'appareil et réinstalle ensuite une ou plusieurs d'entre elles.

Les développeurs d'applications ont donc la possibilité d'identifier les *smartphone* de manière unique, même à travers d'autres applications réalisées par un même développeur. Des applications issues de vendeurs différents percevront cependant des *UUID* différents pour un même *smartphone iOS*.

Le code pour récupérer cet identifiant est présenté comme suit :

```
// Recuperer la valeur de l'identifiant vendeur
 UIDevice.currentDevice().identifierForVendor!.UUIDString
```

De plus, il est aussi possible de récupérer d'autres informations qui peuvent être inférer afin d'avoir des données plus précises sur un utilisateur dans un contexte de suivi de mobilité, comme par exemple le nom associé au *smartphone* (Ex : *iPhone de Taher*), le modèle ou encore le nom et la version du système d'exploitation utilisé :

```
// Nom associe au smartphone
 UIDevice.currentDevice().name

// Modele du smartphone
 UIDevice.currentDevice().model

// Nom du systeme d'exploitation
 UIDevice.currentDevice().systemName

// Version du systeme d'exploitation
 UIDevice.currentDevice().systemVersion
```

7.4.2.2 Identifiant publicitaire

Cependant, pour les réseaux publicitaires, qui nécessitent un identifiant cohérent et qui ne change pas entre les applications de différents développeurs, une approche différente est requise : l'utilisation d'un identifiant publicitaire.

Depuis *iOS* 6.0, Apple a introduit l'identifiant publicitaire, un identifiant du *smartphone* non personnel et non permanent, que les réseaux publicitaires utilisent pour donner à l'utilisateur plus de contrôle sur la capacité des annonceurs à fournir des publicités ciblés grâce à leur méthode de suivi de mobilité. L'utilisateur peut donc choisir à tous moment de change la valeur de l'identifiant ou de limiter les informations envoyées (sa position par exemple).

Le code suivant montre comment une application peut obtenir cet identifiant :

```
Si un identifier publicitaire est utilise, alors on l'affiche
 if ASIdentifierManager.sharedManager().advertisingTrackingEnabled{
myAdvId=ASIdentifierManager.sharedManager().advertisingIdentifier.UUIDString
    Log.d("TEST", "Adversiting identifier : "+myAdvId);
}
```

7.4.2.3 *Scans WiFi* avec *iOS*

Nous allons maintenant nous intéresser à la possibilité de sonder les canaux *WiFi* avec les interfaces de programmation d'*iOS*. Le système d'Apple bloque les méthodes permettant de scanner les périphériques à portée pour des raisons de sécurité. Il est néanmoins possibilite d'utiliser des interfaces de programmations privées, c'est-à-dire des libraires de codes qui ne sont disponibles qu'en ayant les privilèges super-utilisateur du *smartphone*. Pour rappel, les applications utilisant des librairies privées ne peuvent pas être déployer sur l'App store et necessitent que les périphériques soient rootés.

De plus, depuis la version 7.0 du système d'exploitation, Apple propose une application utilitaire officielle déployée sur l'App store nommé *AirPort Utility* permettant aux utilisateurs qui la téléchargent d'exécuter des *scans* de périphériques *WiFi* à portée. Le temps des *scans* peut être ajusté et les informations des points d'accès observés que l'on peut obtenir sont l'adresse MAC (BSSID) et le nom du réseau (SSID). Pour le moment, le code source de cet utilitaire n'est pas dévoilé.

Enfin, pour accéder aux positions géographique du *smartphone*, l'interface de programmation à utiliser est *CoreLocation*. Ce dernier va utiliser tout le matériel embarqué disponible, y compris le *WiFi*, le *GPS*, le *Bluetooth*, le magnétomètre, le baromètre et le réseau cellulaire pour recueillir des données.

Pour utiliser cette interface, il faudra indiquer la permission dans le fichier *info.plist* en ajoutant *NSLocationAlwaysUsageDescription* et un message d'alerte qui sera affiché à l'utilisateur pour lui demander son autorisation afin d'accéder aux données relatives à la position géographique.

Le code suivant présente les étapes à suivre :

```
// Initialiser l'objet LocationManager
 locationManager = CLLocationManager()
 locationManager.delegate = self;
// Precision accrue
 locationManager.desiredAccuracy = kCLLocationAccuracyBest
// Demande de l'autorisation
 locationManager.requestAlwaysAuthorization()
// Mise a jour de la position geographique periodiquement
 locationManager.startUpdatingLocation()
```

Pour afficher la position actuelle (longitude et latitude) de l'utilisateur, la méthode suivante est utilisée :

```
func locationManager(manager: CLLocationManager, didUpdateLocations locations: [CLLocat
    let locValue:CLLocationCoordinate2D = manager.location!.coordinate
    print("locations = \(locValue.latitude) \(locValue.longitude)")
}
```

Outils pour le Bluetooth Classic

Ce chapitre présente les deux principaux outils utilisés dans le cadre de cette thèse afin d'analyser les paquets émis par les interfaces *classic Bluetooth* de différents périphériques. D'une part, nous présenterons l'analyseur *Ubertooth* et d'autre part, nous étudierons l'utilisation des interfaces de programmation applicatives sous iOs et Android en lien avec le *Bluetooth*.

L'outil Ubertooth est un analyseur de paquets Bluetooth *open source* destinée aux expérimentations sur le protocole Bluetooth Classic/BLE. L'outil permet d'envoyer et de recevoir des paquets sur la bande ISM des 2,4 GHz. Sa puissance maximale théorique est de 20 dBM et à une portée théorique de 100 mètres sans obstacles. La partie logicielle d'Ubertooth est composée de plusieurs librairies à installer. Elles sont compatibles avec un système d'exploitation Linux ou MacOSX et l'Ubertooth peut donc être utilisé sur les nano-ordinateurs tels que le *raspberry-pi*.

7.5 Collecte active et passive

Les paquets *Bluetooth* sont composés d'un *Access Code* calculé à l'aide du *Lower Address Part ou LAP* de l'adresse MAC du *master* durant une communication. De plus, cette *Access Code* identifie un piconet *i.e.* une connexion active entre plusieurs périphériques *Bluetooth* qui s'échangent des paquets. Pour rappel, l'adresse MAC Bluetooth est composée sur 48 bits et le *LAP* représente les 24 derniers bits de cette adresse. La section 4.2 décrit le procédé pour recouvrir la valeur du *LAP* et de l'*UAP* du *master* d'une communication.

7.5.1 Passif

Ubertooth perment de collecter passivement les paquets de type *classic Bluetooth*. Pour ce faire, la commande suivant doit être utilisée :

```
ubertooth-rx
```

Pour pouvoir détecter les 8 bits supplémentaires de l'adresse MAC du *master* de la communication, il faut spécifier en paramètre de la commande ci-dessous, la valeur du *LAP* du *master* ciblé :

```
ubertooth-rx -l [LAP_master_target]
```

Pour résumer, ces deux dernières commandes sont recommandées pour être utiliser s'il l'on veut détecter passivement la présence des périphériques *master* qui communiquent. Dans le meilleur cas, il sera possible d'identifier 32 bits sur 48 de l'adresse MAC réelle du *master* de la communication.

Il est possible de lancer cette commande en continu sans interruption. Par défaut, *Ubertooth* va changer de canal parmi les 79 périodiquement de manière aléatoire. Il est possible de spécifier à l'outil de rester sur un canal précis (par exemple, le premier canal à une fréquence de 2402 MHz) via la commande ci-dessous :

```
ubertooth-util -c2402
```

7.5.2 Actif

Lorsque les périphériques *classic Bluetooth* ne communiquent pas, il est possible de sonder les canaux activement via l'utilisation de *SCAN_REQ* pour détecter leur présence s'ils sont en mode découvrable.

La libraire *BlueZ*, sous *Linux*, intégrée à *Ubertooth* permet d'effectuer un scan *actif* durant une période de temps donné en paramètre (ex : 30s), décrit comme suit :

```
ubertooth-scan -b -hci1 -t 30
```

Cependant, cette dernière n'est pas adéquate pour pouvoir effectuer un *scan* actif des périphériques à portée de manière continu.

Une solution alternative serait d'utiliser l'outil *BTScanner* sous *Linux* afin d'exécuter un *scan* actif sans interruption :

```
btscanner
```

Les informations récoltées sur un périphériques sont le temps où celui-ci a été détecté, son adresse MAC Bluetooth, la valeur de son horloge ou encore le nom associé au périphérique.

7.6 Interfaces de programmation applicatives

Nous nous intéressons aux principales méthodes permettant d'interagir avec le Bluetooth Classic via les systèmes d'exploitations d'android et d'iOS. Les principales fonctions mise à disposition aux développeurs sont : le scan des périphériques Bluetooth à portée, l'établissement d'une communication, l'échange de données, la liste des informations Bluetooth sur le périphérique local ou encore la gestion de plusieurs connexions en parallèle.

7.6.1 Permissions

Similairement au WiFi, pour pouvoir interagir avec la carte Bluetooth du smartphone, il faut renseigner à l'application la permission d'utilisation des interfaces de programmation de cette technologie.

7.6.1.1 Android

Afin d'utiliser toutes les fonctionnalités proposées par les interfaces de programmation du Bluetooth sous android, trois principales permissions peuvent être utilisées. La première est le BLUETOOTH. Cette autorisation est nécessaire pour une demande une connexion, accepter une connexion, établir une communication et transférer des données. La seconde permission à déclarer est ACCESS_COARSE_LOCATION ou ACCESS_FINE_LOCATION qui sont en lien avec la position géographique du périphérique. Cette autorisation de localisation est requise car les scan Bluetooth peuvent être utilisés pour collecter des informations sur l'emplacement de l'utilisateur (il en va de même pour le WiFi). Enfin, Si l'application active la découverte de périphériques à portée ou manipule les paramètres Bluetooth, l'autorisation BLUETOOTH_ADMIN doit être déclarée en plus de l'autorisation BLUETOOTH.

Le code suivant écrit dans le fichier *manifest.xml* permet de déclarer les permissions décrites ci-dessus :

```
# <uses-permission android:name="android.permission.BLUETOOTH"/>
# <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
# <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

7.6.1.2 iOS

Depuis la mise à jour vers *iOS* 10, Apple étend la portée du contrôle des informations sur la confidentialité des utilisateurs. Une nouvelle permission appelée permission privée (*privacy permission*) regroupe l'ensemble des interfaces de programmation qui traite avec des données privées de l'utilisateur. Le *Bluetooth* en fait partie.

Par conséquent, si une application souhaite utiliser cette interface, le développeur devra obligatoirement renseigner un descriptif sur l'utilisation que va faire l'application avec cette interface en le déclarant dans le fichier *info.plist* comme suit :

```
# <key>Bluetooth Peripheral Usage Description</key>
# <value>Scan periodique des peripheriques a portee</value>
```

L'utilisateur peut à tout moment modifier les permissions des applications qui utilisent des interfaces qui requierent une permissions privée.

7.6.2 Interfaces de programmation sous Android

Dans cette partie, nous discuterons uniquement des API qui ont un lien avec un contexte de suivi de mobilité, c'est-à-dire l'activation de l'interface Bluetooth, la collecte des identifiants ou encore la découverte des périphériques à portée.

7.6.2.1 Manipulation de l'interface Bluetooth :

La première étape est de pouvoir interagir avec le Bluetooth et voir si cet interface est active. Pour ce faire, l'objet *BluetoothAdapter* de l'interface de programmation est utilisé comme le démontre le code suivant :

```
# BluetoothAdapter mBluetoothAdapter BluetoothAdapter.getDefaultAdapter();
# if (mBluetoothAdapter == null) {
    // Le smartphone ne prend pas en charge le Bluetooth
}
```

De plus, si l'interface Bluetooth existe, l'objet *BluetoothAdapter* peut demander l'activation ou la désactivation de cette interface. Deux types de procédures existent. La

première conciste à alerter l'utilisateur. Le code suivant demande à l'utilisateur l'autorisation d'activer l'interface Bluetooth par le biais d'une fenêtre de dialogue où il devra explicitement donner son accord en cliquant sur un bouton :

```
Si l'interface Bluetoot n'est pas active
# if (!mBluetoothAdapter.isEnabled()) {
// Une autorisation sera envoyée
    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    // En créant une fenêtre de dialogue à l'utilisateur
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

La seconde procédure s'exécute sans l'interaction de l'utilisateur et l'activation ou la désactivation du Bluetooth se fait directement au travers l'exécution du code suivant :

```
public void disableBT(View view){
    BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
    // S'il est actif,
    if (mBluetoothAdapter.isEnabled()){
    // On le désactive.
        mBluetoothAdapter.disable();
    }
}
```

La différence entre l'utilisation de ces deux méthodes est basée sur leurs permissions associées. La première requiert uniquement la permission BLUETOOTH et la seconde exige les permissions BLUETOOTH et BLUETOOTH_ADMIN.

7.6.2.2 Découverte des périphérique Bluetooth à portée

Pour exécuter le service de découverte active, l'interface de programmation propose une méthode simple appelé *startDiscovery()*. Ce processus est asynchrone et retourne une valeur booléenne (oui ou non) lorsque celui-ci aura démarré. Cette méthode essayera de détecter la présence des périphériques en utilisant la procédure d'investigation (*inquiry scan process*), durant environ douze secondes. S'en suit une procédure de pagination (*paging process*) pour récupérer le nom et l'adresse MAC associés aux périphériques détectés.

Le code suivant va déclencher un service de découverte de périphérique et collecter les identifiants de ces appareils détectés :

```
protected void onCreate(Bundle savedInstanceState) {
    ...

    // Cette méthode sera appelé à chaque fois qu'un périphérique sera détecté
    IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
    registerReceiver(mReceiver, filter);

     BluetoothAdapter btAdapter = BluetoothAdapter.getDefaultAdapter();

     // On lance le service de découverte
     btAdapter.startDiscovery();
}


// Affichage les résultats
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            // Un périphérique a été détecté ...
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DE
            // On récupère son nom ...
            String deviceName = device.getName();
            // et son adresse MAC.
            String deviceHardwareAddress = device.getAddress();
        }
    }
};
```

7.6.2.3 Mode découvrable

Un dernier point est la possibilité de rendre *découvrable* le périphérique sur lequel s'exécute l'application *i.e.* détectable par les autres périphériques à portée, en utilisant directement du code sans passer par les paramètres système du smartphone. Par défaut, le durée où il est détectable est de deux minutes mais celui peut être modifié.

Le code suivant va permettre de passer le smartphone en mode *découvrable* durant dix minutes (600 secondes). Il est aussi possible de rendre l'appareil détectable tout le temps en spécifiant la valeur 0.

```
Intent discoverableIntent =
        new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 600);
startActivity(discoverableIntent);
```

Dans ce cas présent, les récentes mises à jour d'android imposent d'avertir l'utilisateur et de lui demander son autorisation avant d'exécuter cette tâche. Il est évident que la possibilité de rendre un périphérique détectable à l'insu de l'utilisateur peut l'exposer à des risques sur ces informations privées (collecte de son identifiant, suivi de sa mobilité). De ce fait, lorsque le code ci-dessus sera exécuté, une fenêtre de dialogue sera présentée à l'utilisateur où il devra clicuer sur un bouton pour autoriser le processus.

7.6.2.4 Informations sur le périphérique local :

Certaines informations en lien avec le Bluetooth revêtent un caractère sensible et sont accessibles via les interfaces de programmation. En effet, il est possible de récupérer les identifiants de tous les périphériques qui se sont appairés sur un smartphone sur lequel est déployée une application comme l'illustre le code suivant :

```
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();

if (pairedDevices.size() > 0) {
    // Il y a des périphériques appairés
    for (BluetoothDevice device : pairedDevices) {
    // Pour chacun d'eux, on récupère le nom et l'adresse MAC
        String deviceName = device.getName();
        String deviceHardwareAddress = device.getAddress();
    }
}
```

Le nom d'un périphérique peut apporter plus de connaissances sur un utilisateur. Par exemple, si le nom d'un périphérique appairé à son smartphone a comme nom "CasqueAudioBeats", on peut aisément en déduire qu'il possède un casque audio de la marque *Beats by Dre*. L'adresse MAC est une information beaucoup plus sensible car elle donne accès à un attaquant de pouvoir identifier et suivre la mobilité du smartphone mais aussi

de tous les périphériques qui se sont appairés à celui-ci. L'utilisateur expose plusieurs identifiants et augmente le risque sur ces informations privées. On peut imaginer un scénario où un attaquant déploie une application qu'il fait passer pour légitime qui va être installée par plusieurs utilisateurs et qui collectera les résultats des scans Bluetooth et la liste des périphériques appairés à chaque smartphone. Ces dernières seront remontées vers une entité centrale où l'utilisateur pourra suivre la mobilité des utilisateurs de smartphones infectés.

7.6.3 Interfaces de programmation iOS

//

Cette partie s'intéresse à l'ensemble des interfaces de programmation sous iOS qui utilisent le Bluetooth. Nous verrons qu'il est beaucoup plus difficile d'exploiter les API d'Apple dans un contexte de PT.

7.6.3.1 Les mécanismes du *classic Bluetooth* gérés par l'*OS*

Il est possible de faire intéragir différents dispositifs en classic Bluetooth avec les périphériques iOS. Cependant, la découverte de périphérique à portée, le processus d'appariement et la communication sont gérés automatiquement par l'*OS* du smartphone. Apple ne permet donc pas d'exécuter ces processus par le biais d'interface de programmation applicatives.

7.6.3.2 *Bluetooth Manager*

Une autre solution plus adéquate serait l'utilisation de l'interface de programmation *Bluetooth Manager*. Cette libraire permet d'utiliser toutes les méthodes liées au service de découverte de périphérique, aux informations sur le périphérique local où est installé l'application, les différentes étapes du processus d'appariement et l'échange de données durant une communication.

L'activation/désactivation de l'interface Bluetooth du smartphone s'effectue comme suit :

```
BluetoothManager *manager = [BluetoothManager sharedInstance];
// ON
[btManager setEnabled:YES];
[btManager setPowered:YES];
```

```
// OFF
[btManager setEnabled:NO];
[btManager setPowered:NO];
```

L'exemple de code ci-dessous décrit comment exécuter un scan actif :

```
BluetoothManager *manager = [BluetoothManager sharedInstance];
if ([btManager enabled])
{
    // start scan
    [btManager  setDeviceScanningEnabled:YES];
}
```

Enfin, la fonction ci-dessous présente la méthode de connexion à un périphérique :

```
/* Bluetooth connectivity */
- (void)deviceConnect:(NSInteger)index {
    // On recupere l'objet qui correspond au
    // peripherique auquel on veut s'appairer
    BTListDevItem *item = (BTListDevItem *)[btDevItems objectAtIndex:index];

    // On se connecte
    [item.btdev connect];
}
}
```

Toutefois, il faut noter que*Bluetooth Manager* est une interface de programmation privée mais ne requiert pas de *rooter* le smartphone pour être utiliser. De ce fait, le déploiement d'une application utilisant cette librairie sera interdite sur le *store* d'application officiel d'Apple. L'intérêt de cette libraire pourrait s'orienter sur l'utilisation d'un smartphone iOS comme station d'écoute qui sonderait les canaux Bluetooth périodiquement pour détecter la présence de périphériques à portée et collecter leur identifiant.

Outils pour le BLE

La technologie du Bluetooth Low Energy, bien que proche de sa version classique, présente des différences fondamentales notamment sur la phase de découverte de périphérique à portée, la composition des paquets de données, certaines informations échangées pendant la phase d'appariement ou le type d'adresse MAC utilisée (voir 3). De ce fait, la plate-forme Ubertooth propose en conséquence des commandes dédiées aux interactions avec des périphériques BLE. En premier lieu, ce chapitre présentera les fonctionnalités BLE supportées par Ubertooth. En seconde partie, les interfaces de programmation applicatives sous android et iOS seront détaillées pour une utilisation dans un contexte de PT.

7.7 Ubertooth

La configuration de cet outil reste la même pour les fonctionnalités BLE et est déjà présentée dans la partie du Bluetooth Classic 7.4.2.3. Le module Ubertooth pour le Bluetooth Low Energy permet :

1. de récolter de manière passive des ADV_PKT et de suivre une connexion active entre des périphériques BLE
2. d'interférer dans des connexions
3. d'envoyer des ADV_PKT

Le premier point est la fonctionnalité la plus robuste supportée par le module BLE d'Ubertooth. Il possède deux modes principaux : le mode *follow* et le mode *promiscuous*. Ubertooth ne peut utiliser qu'un seul de ces deux modes à la fois.

7.7.1 Suivi et écoute de nouvelles connexions

Le mode *follow* consiste à écouter sur l'un des trois canaux (37,38 et 39) dédiés à l'envoi périodique de ADV_IND. Lorsqu'une connexion est établie durant ce mode, Ubertooth va suivre cette connexion en changeant de canal périodiquement selon la séquence de canaux à visiter échangée entre les périphériques BLE de cette dite-connexion. Il va en plus collecter tous les paquets de données échangés. Pendant ce temps, il ne sera plus en mesure de collecter les trames ADV_IND. Une fois que la connexion est terminée, Ubertooth va de nouveau basculer sur les canaux 37,38 et 39 pour collecter passivement les ADV_IND tout en attendant un nouvel établissement de connexion pour réitérer le processus.

La commande suivante permet d'utiliser le mode *follow* avec l'argument *-f* :

```
# ubertooth-btle -f
```

7.7.2 Suivi et écoute de connexions existante (mode promisuous)

Le mode *promiscuous* permet de suivre et d'écouter des connexions déjà établies. Ce mode peut être utilisé pour écouter des connexions de longue durée au cours du temps. Ubertooth va sonder les 37 canaux de données afin de détecter la présence d'une connexion entre plusieurs périphériques BLE. Il va ensuite essayer de retrouver la séquence de saut utilisée afin de collecter les paquets de données. La commande suivante permet d'utiliser le mode *promiscuous* avec l'argument *-p* :

```
# ubertooth-btle -p
```

Par défaut, Ubertooth va suivre la première connexion disponible qu'il voit selon les paquets collectés. Il est possible de spécifier une adresse MAC avec l'argument *-t* pour suivre uniquement la connexion d'un périphérique cible. Par exemple :

```
# ubertooth-btle -f -tXX:XX:XX:XX:XX:XX
```

7.7.3 Interferer avec des connexions existantes

Par ailleurs, Ubertooth offre la possibilité d'interférer avec les connexions nouvellement établies ou de longue durée. Lorsque cette attaque réussit, la connexion BLE entre les périphériques BLE sera rompue. Par exemple, la commande pour interférer des connexions récupérées avec le mode *promiscuous* s'utilise avec l'argument *-I* comme suit :

```
# ubertooth-btle -p -I
```

7.7.4 Envoi d'ADV_INT

Une dernière fonctionnalité que propose ce module mais qui est encore à l'état expérimental est l'envoi d'*Advertising Packets* avec une adresse MAC spécifié en argument. La commande suivante va envoyer périodiquement l'ADV_IND d'adresse XX :XX :XX :XX :XX :XX :

```
# ubertooth-btle -sXX:XX:XX:XX:XX:XX
```

7.7.5 Ecoute du BLE sur un canal

Durant nos expérimentations, nous avons collecté plusieurs ADV_IND émis par différents capteurs BLE (montre connecté, bracelet connecté, ou cardio-fréquence mètre) en utilisant Ubertooth en mode *follow*. Cependant, cette commande ne permet pas un scan continu car Ubertooth va suivre une connexion dès que possible et ne remontera plus les informations observées sur le canal d'écoute initial. Pour contourner ce problème, nous avons spécifié en argument de la commande, une adresse MAC d'un périphérique à suivre qui n'existe pas (ou qui a une très faible probabilité d'exister), par exemple :

```
# ubertooth-btle -f -t00:00:00:00:00:00
```

7.7.6 Attaque de la phase d'appariement

Ryan et al. ont développé un outil nommé *Crackle*[124] qui exploite une faille dans le processus d'appariement BLE pour récupérer les valeurs des clés secrètes échangées, à savoir l'IRK ou la LTK et par la suite de déchiffrer les paquets de données collectés (voir 3.1.4). Ubertooth peut être couplé à cet outil en sauvegardant tous les paquets collectés dans un fichier *.pcap* avec l'argument *-c* :

```
# ubertooth-btle -f -c sample.pcap
```

Il faudra utiliser ce fichier résultat pour retrouver la valeur de la LTK qui va permettre de déchiffrer tous les paquets de données collectés avec l'argument *-o* :

```
# crackle -i sample.pcap -o decrypted_sample.pcap
```

Dans un contexte de suivi de la mobilité des utilisateurs, les paquets de données BLE ne sont pas construits à partir de l'adresse MAC Bluetooth réelle d'un périphérique contrairement au Bluetooth Classic. De ce fait, il n'est pas possible d'extraire des parties de l'adresse MAC réelle à partir de ces paquets. Cependant, il est probable que d'autres informations contenues dans ces paquets aient une entropie assez forte pour identifier un ou plusieurs périphériques durant la communication. Pour ce faire, il faut absolument récupérer la clé LTK qui va permettre de déchiffrer le contenu de ces paquets.

Un second intérêt de Crackle est qu'il permet d'obtenir l'IRK. Celui-ci pourra être utilisé pour résoudre des adresses privées résolvables.

7.8 Interfaces de programmation applicatives

Bien que similaire par leur nom, la technologie du BLE présente de grandes différences dans sa conception par rapport à sa version classique comme nous l'avons décrit dans les sections précédentes. De ce fait, les concepteurs d'interfaces de programmation se sont adaptés et ont mis à disposition des développeurs une nouvelle interface dédiée au Bluetooth Low Energy. Nous allons exposer dans cette sous-section les méthodes et les contraintes liées aux interfaces de programmation du BLE que peut utiliser un développeur dans le but d'élaborer une application liée au PT.

7.8.1 Permissions

Dans le but d'utiliser toutes les fonctionnalités du BLE lors du développement d'une application, la déclaration des permissions est obligatoire. Le BLE n'échappe pas à cette règle d'autant plus qu'elle traite des informations privées (adresse MAC Bluetooth par exemple) sur l'interface Bluetooth du smartphone auquel l'application sera déployée.

Android Les interfaces de programmation du BLE ont été introduites depuis la version 4.3 d'android. Similairement aux permissions décrites pour sa version classique 7.6.1, la permission BLUETOOTH est nécessaire pour l'utilisation des différentes fonctions essentielles du BLE, à savoir l'activation de l'interface Bluetooth, une demande de connexion, l'acceptation d'une connexion entrante mais surtout l'échange de données. De plus, la permission BLUETOOTH_ADMIN est utilisée dans la cadre de manipulation des options Bluetooth du smartphone et pour le service de découverte de périphériques à portée.

Par suite, certaines fonctionnalités de ces interfaces permettent au smartphone d'envoyer périodiquement des trames de *beacon* qui utilisent la position géographique actuelle du smartphone. De ce fait, la permission ACCESS_COARSE_LOCATION ou ACCESS_FINE_LOCATION doit être déclarée afin d'en avertir l'utilisateur.

Finalement, le contenu du fichier *manifest.xml* est quasi-similaire à celui de la version classique du Bluetooth excepté l'ajout d'une ligne spécifiant que l'application utilisera la technologie du BLE :

```
# <uses-permission android:name="android.permission.BLUETOOTH"/>
# <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
# <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
// Spécifie que l'application utilisera le BLE
#<uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>
```

iOS L'interface de programmation principale du BLE sous iOS se nomme *Core Bluetooth*. Similairement à version classique, la technologie BLE est considérée comme sensible car elle possède des informations privées du smartphone de l'utilisateur. Elle possède donc une permission privée à déclarer dans le fichier *info.plist*.

De plus, pour renseigner que l'utilisateur que l'application va utiliser les fonctionnalités BLE, la ligne de code suivante doit être spécifier dans le fichier de configuration de l'application (*info.plist*) comme suit :

```
#<key>UIRequiredDeviceCapabilities</key>
#<array>
#    <string>bluetooth-le</string>
#</array>
```

Par suite, certaines fonctionnalités liées au BLE utilisent la position géographique du smartphone qui est considérée comme une information sensible. De ce fait, pour manipuler ce type de donnée, une permission doit être obtenue de la part de l'utilisateur. Celle ci doit être décrite suivant le format suivant :

```
<key>NSLocationWhenInUseUsageDescription</key>
<string>...Detailed reason...</string>
<key>NSLocationAlwaysUsageDescription</key>
<string>...Detailed reason...</string>
```

Depuis *iOS8*, la description détaillé dans le fichier *.plist* est obligatoire sinon l'application ne se lancera pas. La clé *NSLocationWhenInUseUsageDescription* spécifique que l'accès aux données géographiques sera fera uniquement lorsque l'application sera exécuté en activité principale. La seconde (*NSLocationAlwaysUsageDescription*) permet d'accéder a ce type de donné même lorsque l'application s'exécute en arrière-plan. Pour apporter un contrôle plus fun à l'utilisateur, ce dernier pourra refuser l'accès aux données géographiques via le BLE à l'application à tout moment en se rendant dans les paramètres général du smartphone.

7.8.2 Interface de programamtion sous Android

Cette sous-section présentera les différentes fonctionnalités BLE qu'a en sa possession un développeur sur la plate-forme Android dans un contexte de PT. Elle reprendra les différents mécanismes du BLE décrits dans les paragraphes précédents, à savoir, le service de découverte de périphérique, la processus d'appariement et la phase de communication.

7.8.2.1 Informations locales du smartphone

L'activation, la désactivation du Bluetooth ainsi que la collecte de certaines informations sensibles (adresse MAC réelle, liste des périphériques appairés, nom) liées à l'interface Bluetooth du smartphone sont accessibles via l'interface de programmation Android. Les méthodes d'accès sont similaires à la classique présentée dans la section 7.6.2.

7.8.2.2 Service de découverte de périphérique à portée

La seconde étape est la possibilité de sonder les canaux Bluetooth à la recherche de périphériques BLE à portée. Il est possible d'exécuter ce processus de manière continue ou durant une période spécifiée par la méthode suivante :

```
public void scanLeDevice(boolean enable) {

BluetoothManager bluetoothManager = (BluetoothManager)
getActivity().getSystemService(Context.BLUETOOTH_SERVICE);
mBluetoothAdapter = bluetoothManager.getAdapter();
BluetoothLeScanner bluetoothLeScanner = mBluetoothAdapter.getBluetoothLeScanner();

if (enable) {
    // Stop scan apres un delai
```

```
        mHandler.postDelayed(new Runnable() {
            @Override
            public void run() {
                mScanning = false;

                bluetoothLeScanner.stopScan(mLeScanCallback);
            }
        }, SCAN_PERIOD);

        // Start scan
        mScanning = true;
        bluetoothLeScanner.startScan(mLeScanCallback);
    } else {
        // Stop scan
        mScanning = false;
        bluetoothLeScanner.stopScan(mLeScanCallback);
    }
}
```

La méthode de *callback* affiche les résutats des périphériques détectés. Dans l'exemple suivant, nous récupérons le nom associé à l'interface *Bluetooth* et son adresse MAC :

```
        public ScanCallback mLeScanCallback = new ScanCallback() {
    @Override
    public void onScanResult(int callbackType, ScanResult result) {
        super.onScanResult(callbackType, result);
        // On recupere l'adresse MAC
        // et le nom du peripherique detecte
        String addr = result.getDevice().getAddress();
        String name = result.getDevice().getName();
    }
};
```

Ce service de découverte de périphérique est un processus actif *i.e.* que le périphérique va collecter tous les ADV_IND, émettre une SCAN_REQ pour chaque ADV_IND reçu et attendre une réponse pour détecter la présence d'un périphérique BLE à portée. L'utilisation d'un scan passif n'est pas autorisé par les interfaces de programmation.

7.8.2.3 Mode *advertiser* - envoie d'*Advertising Packets*

Dans la plupart des cas, le smartphone BLE jouera le rôle de *scanner i.e.* qu'il va sonder les canaux Bluetooth à la recherche de périphériques BLE à proximité. Cependant, l'interface de programmation du BLE permet également de mettre le smartphone en mode *advertiser*. Celuici va envoyer périodiquement des ADV_IND pour alerter sa présence dans son voisinage. Il est possible de cloner des ADV_IND précédemment aperçus lorsque l'application était en mode *scanner* ou de forger ses propres paquets à émettre. Pour la seconde option, il faut générer les paramètres d'envoi des paquets et les données à envoyer. La génération des paramètres se fait au travers de l'objet *AdvertiseSettings* où l'on peut indiquer la fréquence et la portée d'envoi des paquets mais aussi la possibilité de se connecter à ce périphérique :

```
AdvertiseSettings settings = new AdvertiseSettings.Builder()
        .setAdvertiseMode( AdvertiseSettings.ADVERTISE_MODE_LOW_LATENCY )
        .setTxPowerLevel( AdvertiseSettings.ADVERTISE_TX_POWER_HIGH )
        .setConnectable( false )
        .build();
```

Le code précédent montre par exemple que l'on va paramétrer le smartphone pour émettre peu fréquemment (dans le but d'économiser de la batterie) avec une forte portée et qu'il est impossible de se connecter à cet appareil.

Par suite, le paramétrage des données à émettre se déclare de la façon suivante :

```
AdvertiseData data = new AdvertiseData.Builder()
        .setIncludeDeviceName(true)
        .addServiceUuid(pUuid)
        .addServiceData(pUuid, "Data".getBytes(Charset.forName("UTF-8")))
        .build();
```

Ce code montre que l'on va inclure le nom associé à l'interface Bluetooth dans nos paquets, un *service* identifié par un UUID préalablement généré (variable *pUuid*) et les données de ce service à envoyer, dans notre cas, la chaîne de caractère "Data" convertie en un flux binaire.

Enfin, la dernière étape est l'appel à la méthode *advertiser.startAdvertising(settings, data, callback) ;* qui lancera le processus avec une méthode de *callback* pour gérer le bon déroulement des actions d'envoi de paquets.

Nous constatons évidemment dans un contexte de suivi de mobilité qu'il y a un potentiel danger pour les utilisateurs. Un attaquant peut périodiquement transformer les smartphones sur lesquels une application non-légitime est installé en *advertiser* pour plus facilement détecter la présence des utilisateurs par des sondes qui seraient installées à des points stratégiques dans une zone donnée ou par d'autres smartphones en mode *scanner*. Les interfaces de programmation sous android n'empêchent pas de lancer ce processus en tâche de fond. Il suffit juste que l'attaquant ne déclenche pas ce processus trop souvent car ce dernier augmente la consommation énergétique de l'appareil ce qui pourrait éveiller des soupçons de la part des utilisateurs affectés.

7.8.2.4 Processus d'appariement et communication

Avant d'entrer dans le vif de ce sujet, nous allons définir et détailler quelques mots clés liés à la phase d'échange de données en BLE.

Définitions (GATT, services et caractéristiques : Les attributs génériques ou *GATT* pour *Generic Attribute*, représente une structure de données exposée aux périphériques BLE connectés.

Ces données sont identifiées par un *UUID* (*Universal Unique Identifier*) unique sur 128 bits. Ils sont composés de caractéristiques et de services. Une caractéristique est représentée par une valeur. Un service représente un ensemble de caractéristiques.

Connexion à un périphérique avec le profil GATT : Lors d'une communication, l'un des périphérique jouera le rôle de serveur *GATT* et l'autre de client *GATT*. Dans la plupart des cas, le smartphone sera client et l'objet connecté le serveur. Le code suivant met en avant la demande d'une connexion par l'application installé sur le smartphone :

```
# private BluetoothGatt mBluetoothGatt;
# mBluetoothGatt = device.connectGatt(this, false, mGattCallback);
// Le premier paramètre représente le contexte,
// le second indique une reconnexion automatique ou non
// le troisième est la fonction qui remonte les résultats
```

Une fois la connexion établie, l'application déployée sur le smartphone aura accès à l'ensemble des données qui sont disponibles en lecture et/ou en écriture.

Certains périphériques qui jouent le rôle de serveur *GATT* peuvent autoriser une connexion sans authentification *i.e.* toute application peut se connecter librement au périphérique pour pouvoir avoir accès à ces données.

D'autre périphériques peuvent nécessiter une authentification *i.e.* que l'application doit d'abord s'appairer au périphérique cible. Dans ce cas, la méthode *connectGatt(...)* ne va pas aboutir et va automatiquement entamer un processus d'appariement avec le périphérique client. Cela est géré par l'OS du smartphone et non par les interfaces de programmation. Il est également possible d'entamer une appariement à l'aide de la méthode *createBond()*.

La figure 7.1 résume l'interaction entre un client *GATT* et un serveur *GATT*. Le nom et l'UUID du service, des caractéristiques sont définies par la norme du Bluetooth. L'accès aux données requiert une authentification. Dans notre exemple, la valeur de la caractéristique est la fréquence cardiaque calculée.

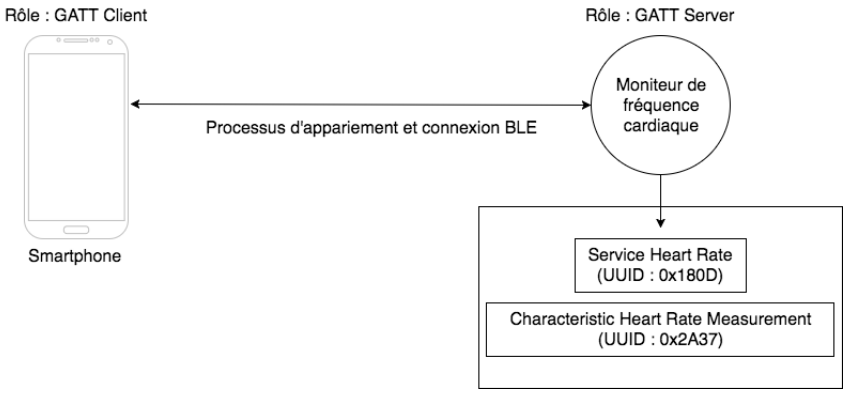


FIGURE 7.1: Schéma montrant l'interaction entre un smartphone et un moniteur de fréquence cardiaque connecté.

7.8.3 iOS

Apple, contrairement au cas du Bluetooth Classic, offre d'avantage de possibilités aux développeurs d'application BLE tout en conservant un certain contrôle.

7.8.3.1 Informations sur le smartphone

L'activation et la désactivation de l'interface Bluetooth via les interfaces de programmation proposées par iOS est impossible. La seule information que peut avoir le développeur via le code est l'état actuel du Bluetooth (actif ou non). Il peut, cependant, alerter l'utilisateur en affichant une fenêtre de dialogue à l'écran et en lui demandant explicitement d'activer l'interface Bluetooth tel que décrit par le code suivant :

```
- (void)centralManagerDidUpdateState:(CBCentralManager *)central {

    if (central.state == CBCentralManagerStatePoweredOff) {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle: @"Error" message: @"Pl
       [alert show];
    }
 }
```

L'adresse MAC Bluetooth du smartphone sur lequel s'exécute une application n'est pas accessible.

7.8.3.2 Service de découverte de périphériques à portée

La découverte de périphériques BLE à portée est accessible via la méthode *scanForPeripherals(...)*. Le terme *peripherals* fait référence aux périphériques qui envoient des ADV_IND par opposition aux *centrals* qui sont ceux qui effectuent une scan des canaux BLE. Il est possible d'effectuer ce processus selon une période donnée ou continuellement. C'est au développeur d'appeler la méthode *stopScan()* pour arrêter l'exécution de ce processus. De plus, il n'est possible de lancer un scan BLE en tâche de fond uniquement en spécifiant un UUID d'un *service* passé en paramètre de la méthode *scanForPeripherals(...)*. De ce fait, seuls les périphériques BLE à portée utilisant ce service seront détectables par l'application. Pour détecter la présence de tous les périphériques BLE à portée sans utiliser de filtre, l'application ne doit pas s'exécuter en tâche de fond. Chaque fois qu'un périphérique BLE est détecté, l'interface de programmation permet de récupérer son nom, la liste des *services*, un UUID identifiant le périphérique ou encore le rapport signal sur bruit du périphérique distant.

L'accès à l'adresse MAC Bluetooth du périphérique détecté est bloqué et il est impossible de récupérer sa valeur. Pour protéger cette information sensible, Apple a développé un système d'identifiant local au smartphone pour pouvoir identifier de façon unique les différents équipements BLE. En effet, pour chaque périphérique détecté, le smartphone va générer localement un identifiant unique (UUID sur 128 bits) qui sera associé à ce dernier. De ce fait, pour un même périphérique BLE P qui émet des ADV_IND, deux smartphones *S1* et *S2* sous iOS lui associeront un identifiant différent. *S1* et *S2* détecteront la présence de P sans pour autant savoir qu'il s'agit du même périphérique.

D'un point de vue du PT, cette nouvelle gestion des périphériques BLE apporte une sécurité évidente sur la donnée privée que représente l'adresse MAC Bluetooth car il est plus difficile de mettre en oeuvre des attaques de type BLEB. Cependant, le nom associé

et la liste des *services* constituent tout de même, une source d'informations qui selon leur contenu pourraient identifier un utilisateur.

7.8.3.3 Processus d'appariement et communication

Contrairement au système d'exploitation android, l'interface de programmation ne permet pas de déclencher un processus d'appariement. Pour la communication,la procédure de connexion à un périphérique BLE se fait à l'aide la méthode *connect(...)* de l'objet *CentralManager*. Une fois connecté, l'application sera en mesure d'accéder à la liste des *services*, des *characteristics* et des *descriptors* disponibles et d'effectuer des actions de lecture et/ou d'écriture. Si ces données sont chiffrés, elle requiert une authentification. Dans ce cas, le système d'exploitation d'Apple va automatiquement déclencher un processus d'appariement si la procédure de connexion échoue :

```
// Moniteur de frequence cardiaque cible
var heartRatePeripheral: CBPeripheral!

// Initialisation
var centralManager = CBCentralManager(delegate: self, queue: nil);

// Connexion
centralManager.connect(heartRatePeripheral);

// Connexion réussie,
// lister les services disponibles
heartRatePeripheral.discoverServices(nil);
```

7.8.3.4 iBeacon

Comme décrit dans l'état de l'art sur le BLE, les applications déployées tirent profit de cette technologie dans un contexte de localisation à l'intérieur de bâtiment. En effet, cette méthode consiste à placer à des endroits stratégiques des capteurs qui émettent périodiquement des ADV_IND, autrement dit des paquets de type *beacon* dans le but de proposer de la publicité ciblée ou des promotions sur un produit spécifique. C'est dans ce contexte qu'Apple a développé son propre standard appelé *iBeacon*. Les beacons sont composés d'un *UUID*, d'un *minor*, d'un *major* et de la puissance d'émission *Tx-power* :

- UUID : Codé sur 16 octets, il représente une valeur associée à un groupe de capteurs iBeacon qui partagent une information similaire.

Lieu des magasins		**Paris**	**Londres**	**San Francisco**
UUID		DECF18B9-2539-43D1-81A9-2A49B4C1AD5C		
Major		1	2	3
Minor	Informatique	10	10	10
	Électronique	20	20	20
	Robotique	30	30	30

Tous les capteurs de type *beacon* relatifs à un même déploiement partagent un même UUID.

- Major : Cette valeur codée sur 2 octets permet de distinguer un sous-ensemble de capteurs *beacon* parmi tout l'ensemble déployé.
- Minor : Codée sur 2 octets, cette valeur permet d'identifier un capteur *beacon* de façon individuel.
- Tx Power : Cette valeur est utilisée pour déterminer la proximité (distance) de l'application de l'utilisateur. La puissance TX est définie comme la force du signal à exactement 1 mètre de l'appareil. Elle doit être calibré et codée en avance. Les appareils peuvent ensuite l'utiliser comme référence pour obtenir une estimation approximative de la distance.

Ce standard est aussi compatible avec Android, qui de son côté, n'a pas créé son propre protocole mais assure le bon fonctionnement de leurs interfaces de programmation avec des capteurs de type *beacon*. À noter qu'iOS et android ne sont pas les seuls à proposer du contenu pour cette technologie, mais d'autres compagnies (Estimote,Radius Network ...) se sont spécialisés dans ce domaine en proposant leurs propres capteurs ainsi que leur propre kit de développement composés d'interfaces de programmation spécialisées.

Dans un contexte de PT, l'objectif est de pouvoir collecter un identifiant unique de l'utilisateur et de pouvoir suivre sa mobilité. Les informations envoyés par les capteurs *beacons* ne sont pas chiffrés et peuvent être collectées très facilement pour un attaquant. Celui-ci peut cloner ces informations et l'insérer dans un capteur qui lui appartient ou même transformer son propre smartphone en capteur *beacons* en utilisant les interfaces de programmation officielles, pour pouvoir interagir avec l'application dédiée installée sur les smartphones des utilisateurs. Ensuite, l'attaque dépend du comportement des capteurs *beacons* installés. Si ces derniers sont utilisés pour autoriser les connexions, alors l'application cible peut se connecter au capteur cloné de l'attaquant pour y divulguer sa position et des informations personnelles, sinon ces capteurs n'autorisent aucune connexion entrante et ne permet à l'attaquant d'aboutir à la collecte des données sensibles des utilisateurs.

7.8.3.5 Mode *advertiser* :

Similairement à la plate-forme android, il est possible d'utiliser le smartphone comme un *advertiser* qui va émettre périodiquement des ADV_IND. Pour reprendre l'exemple précédent, nous allons montrer au travers de quelques lignes de code la démarche à suivre pour transformer un smartphone iOS en émetteur de *beacons*. La première étape est de renseigner les données que va envoyer l'appareil.

```
func createBeaconRegion() -> CLBeaconRegion? {
    let proximityUUID = UUID(uuidString:
                "40EF23AF-0020-5153-90A3-C3421FF1881D")
    let major : CLBeaconMajorValue = 100
    let minor : CLBeaconMinorValue = 1
    let beaconID = "com.example.myDeviceRegion"

    return CLBeaconRegion(proximityUUID: proximityUUID!,
                major: major, minor: minor, identifier: beaconID)
}
```

Le code précédent va générer les valeurs de l'UUID, du *Minor* et du *Major*. La valeur beaconID est un identifiant local pour différencier les régions/zones prises en charge par l'application et ne sera pas contenu dans les paquets émis. Le code suivant déclare que l'application se comportera en tant que *peripheral* et va emettre des ADV_IND :

```
func advertiseDevice(region : CLBeaconRegion) {
    let peripheral = CBPeripheralManager(delegate: self, queue: nil)
    let peripheralData = region.peripheralData(withMeasuredPower: nil)

    peripheral.startAdvertising(((peripheralData as NSDictionary) as! [String : Any]))
}
```

L'appel à la méthode *startAdvertising(...)* présentée ci-dessus va permettre au smartphone d'émettre périodiquement des ADV_IND qui vont contenir comme donnée l'UUID, le *Minor* et le *Major*. Une information importante à prendre en compte est qu'une application iOS qui joue le rôle d'un *advertiser* ne peut pas fonctionner si elle est exécutée en tâche de fond. Si l'utilisateur quitte l'application, le smartphone n'enverra plus de paquets BLE. De plus, les informations sur la période d'envoie et la puissance de transmission des paquets n'est pas paramétrable avec les interfaces de programmation et est

automatiquement géré par le système d'exploitation iOS. Enfin, dans un contexte de PT, ce contrôle imposé par Apple permet de contrer les attaques qui consistent à exécuter ce code en tâche de fond à l'insu d'un utilisateur et donc ne pas l'exposer à de potentiels risques pour sa vie privée.

Bibliographie

[1] Ieee standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements-part 11 : Wireless lan medium access control (mac) and physical layer (phy) specifications : Amendment 6 : Medium access control (mac) security enhancements. IEEE Std 802.11i-2004, pages 1–190, July 2004. doi : 10.1109/IEEESTD.2004.94585.

[2] Mathieu Cunche. I know your MAC address : targeted tracking of individual using Wi-Fi. Journal of Computer Virology and Hacking Techniques, December 2013. doi : 10.1007/s11416-013-0196-1. URL http://hal.inria.fr/hal-00923467.

[3] Mathieu Cunche, Mohamed A. Kaafar, and Roksana Boreli. Linking wireless devices using information contained in Wi-Fi probe requests. Pervasive and Mobile Computing, 2013. doi : 10.1016/j.pmcj.2013.04.001. URL http://hal.inria.fr/hal-00816374.

[4] Andreea Picu and Thrasyvoulos Spyropoulos. Dtn-meteo : Forecasting the performance of dtn protocols under heterogeneous mobility. IEEE/ACM Trans. Netw., 23(2) :587–602, April 2015. ISSN 1063-6692. doi : 10.1109/TNET.2014.2301376. URL http://dx.doi.org/10.1109/TNET.2014.2301376.

[5] G. S. Thakur, U. Kumar, A. Helmy, and W. J. Hsu. On the efficacy of mobility modeling for dtn evaluation : Analysis of encounter statistics and spatio-temporal preferences. In 2011 7th International Wireless Communications and Mobile Computing Conference, pages 510–515, July 2011. doi : 10.1109/IWCMC.2011.5982586.

[6] P-U Tournoux, Jérémie Leguay, Farid Benbadis, Vania Conan, M Dias De Amorim, and John Whitbeck. The accordion phenomenon : Analysis, characterization, and impact on dtn routing. In INFOCOM 2009, IEEE, pages 1116–1124. IEEE, 2009.

[7] Augustin Chaintreau, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass, and James Scott. Impact of human mobility on opportunistic forwarding algorithms.

IEEE Transactions on Mobile Computing, 6(6) :606–620, June 2007. ISSN 1536-1233. doi : 10.1109/TMC.2007.1060. URL http://dx.doi.org/10.1109/TMC.2007.1060.

[8] Xiaolan Zhang, Jim Kurose, Brian Neil Levine, Don Towsley, and Honggang Zhang. Study of a bus-based disruption-tolerant network : Mobility modeling and impact on routing. In Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking, MobiCom '07, pages 195–206, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-681-3. doi : 10.1145/1287853.1287876. URL http://doi.acm.org/10.1145/1287853.1287876.

[9] Injong Rhee, Minsu Shin, Seongik Hong, Kyunghan Lee, Seong Joon Kim, and Song Chong. On the levy-walk nature of human mobility. IEEE/ACM Trans. Netw., 19(3) :630–643, June 2011. ISSN 1063-6692. doi : 10.1109/TNET.2011.2120618. URL http://dx.doi.org/10.1109/TNET.2011.2120618.

[10] Minkyong Kim and David Kotz. Extracting a mobility model from real user traces. In In Proceedings of IEEE INFOCOM, 2006.

[11] Long Vu, Klara Nahrstedt, Samuel Retika, and Indranil Gupta. Joint Bluetooth/-Wifi Scanning Framework for Characterizing and Leveraging People Movement in University Campus. In Proceedings of the 13th ACM International Conference on Modeling, Analysis, and Simulation of Wireless and Mobile Systems, MSWIM '10, pages 257–265, New York, NY, USA, October 2010. ACM. ISBN 978-1-4503-0274-6. doi : 10.1145/1868521.1868563. URL http://dx.doi.org/10.1145/1868521.1868563.

[12] Fosca Giannotti and Dino Pedreschi. Mobility, data mining and privacy : Geographic knowledge discovery. Springer Science & Business Media, 2008.

[13] David Kotz, Tristan Henderson, Ilya Abyzov, and Jihwang Yeo. CRAWDAD dataset dartmouth/campus (v. 2009-09-09). Downloaded from http ://crawdad.org/dartmouth/campus/20090909, September 2009.

[14] Teemo. URL https://teemo.co/fr/.

[15] Aislelabs. URL https://www.aislelabs.com/.

[16] Euclid. URL https://geteuclid.com.

[17] Onyxbeacon. URL https://www.onyxbeacon.com/solutions/.

[18] Swirl. URL https://www.swirl.com.

[19] Fidzup. URL https://www.fidzup.com.

[20] Analytics in-store : panorama des technologies d'analyse du parcours shopper et de mesure du trafic points de vente. URL http://www.blog-to-store.com/data-in-store-panorama-des-technologies-danalyse-du-parcours-shopper-et-de-mesure

[21] Iinside. URL https://iinside.com/.

[22] The icon of modern art puts estimote beacons on display, 2015. URL https://blog.estimote.com/post/157200820650/the-icon-of-modern-art-puts-estimote-beacons-on.

[23] A. Thangavelu, K. Bhuvaneswari, K. Kumar, K. SenthilKumar, and S.N. Sivanandam. Location identification and vehicle tracking using vanet (vetrac).

[24] The use of beacons at the levi's stadium, 2017. URL http://www.ibeacontrends.com/beacons-levis-stadium/.

[25] Trackr's official website. URL https://www.thetrackr.com.

[26] Tile mate trackers. URL https://www.thetileapp.com/en-us/store/tiles/mate.

[27] Wistiki. URL https://www.wistiki.com.

[28] Your apps know where you were last night, and they're not keeping it secret, . URL https://www.nytimes.com/interactive/2018/12/10/business/location-data-privacy-apps.html.

[29] How to stop apps from tracking your location, . URL https://www.nytimes.com/2018/12/10/technology/prevent-location-data-sharing.html.

[30] Levent Demir, Mathieu Cunche, and Cédric Lauradoux. Analysing the privacy policies of Wi-Fi trackers. In Workshop on Physical Analytics, Bretton Woods, United States, June 2014. ACM. doi : 10.1145/2611264.2611266. URL http://hal.inria.fr/hal-00983363.

[31] Commission nationale de l'informatique et des libertés, 2016. URL http://www.cnil.fr.

[32] Les start-up fidzup et teemo épinglées par la cnil pour non-respect des données clients. URL https://www.lsa-conso.fr/les-start-up-fidzup-et-teemo-epinglees-par-la-cnil-pour-non-respect-des-donnees-c 293671.

[33] Peter Eckersley. How unique is your web browser ? In Proceedings of the 10th International Conference on Privacy Enhancing Technologies, PETS'10, pages 1–18, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-14526-4, 978-3-642-14526-1. URL http://dl.acm.org/citation.cfm?id=1881151.1881152.

[34] Bin Zan, Marco Gruteser, and Fei Hu. Key agreement algorithms for vehicular communication networks based on reciprocity and diversity theorems. IEEE Trans. Vehicular Technology, 62(8) :4020–4027, 2013. doi : 10.1109/TVT.2013.2254507. URL https://doi.org/10.1109/TVT.2013.2254507.

[35] Bin Zan, Zhanbo Sun, Macro Gruteser, and Xuegang Ban. Linking anonymous location traces through driving characteristics. In Proceedings of the Third ACM Conference on Data and Application Security and Privacy, CODASPY '13, pages 293–300, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1890-7. doi : 10.1145/2435349.2435391. URL http://doi.acm.org/10.1145/2435349.2435391.

[36] J. Thramann. Systems and methods for tracking mobile devices, mar 2013. URL https://www.google.com/patents/US20130072226. US Patent App. 13/624,428.

[37] Sandra Siby, Rajib Ranjan Maiti, and Nils Ole Tippenhauer. Iotscanner : Detecting privacy threats in iot neighborhoods. In Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security, IoTPTS '17, pages 23–30, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4969-7. doi : 10.1145/3055245.3055253. URL http://doi.acm.org/10.1145/3055245.3055253.

[38] D. A. Knox and T. Kunz. Wireless fingerprints inside a wireless sensor network. ACM Trans. Sen. Netw., 11(2) :37 :1–37 :30, March 2015. ISSN 1550-4859. doi : 10.1145/2658999. URL http://doi.acm.org/10.1145/2658999.

[39] Ryan M. Gerdes, Thomas E. Daniels, Mani Mina, and Steve F. Russell. Device identification via analog signal fingerprinting : A matched filter approach. In In 144 Proceedings of the Network and Distributed System Security Symposium (NDSS, page 78, 2006.

[40] Boris Danev, Davide Zanetti, and Srdjan Capkun. On physical-layer identification of wireless devices. ACM Comput. Surv., 45(1) :6 :1–6 :29, December 2012. ISSN 0360-0300. doi : 10.1145/2379776.2379782. URL http://doi.acm.org/10.1145/2379776.2379782.

[41] Bloessl Bastian, Sommer Christoph, Dressier Falko, and Eckhoff David. The scrambler attack : A robust physical layer attack on location privacy in vehicular networks. 2015 International Conference on Computing, Networking and

Communications (ICNC), 00 :395–400, 2015. doi : doi.ieeecomputersociety.org/10.1109/ICCNC.2015.7069376.

[42] Yan Li and Ting Zhu. Gait-based wi-fi signatures for privacy-preserving. In Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, ASIA CCS '16, pages 571–582, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4233-9. doi : 10.1145/2897845.2897909. URL http://doi.acm.org/10.1145/2897845.2897909.

[43] Jin Zhang, Bo Wei, Wen Hu, and Salil S. Kanhere. Wifi-id : Human identification using wifi signal. In DCOSS, pages 75–82. IEEE Computer Society, 2016. ISBN 978-1-5090-1460-6.

[44] T. Kohno, A. Broido, and k. claffy. Remote physical device fingerprinting. IEEE Transactions on Dependable and Secure Computing, 2(2) :93–108, May 2005.

[45] S. B. Moon, P. Skelly, and D. Towsley. Estimation and removal of clock skew from network delay measurements. In IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320), volume 1, pages 227–234 vol.1, March 1999. doi : 10.1109/INFCOM.1999.749287.

[46] Internet Control Message Protocol. RFC 792, September 1981. URL https://rfc-editor.org/rfc/rfc792.txt.

[47] David A. Borman, Robert T. Braden, and Van Jacobson. TCP Extensions for High Performance. RFC 1323, May 1992. URL https://rfc-editor.org/rfc/rfc1323.txt.

[48] Network Time Protocol (Version 3) Specification, Implementation and Analysis. RFC 1305, March 1992. URL https://rfc-editor.org/rfc/rfc1305.txt.

[49] Ieee 802.11 working group and other, 2012.

[50] Marius Cristea and Bogdan Groza. Fingerprinting smartphones remotely via icmp timestamps. IEEE Communications Letters, 17(6) :1081–1083, 2013.

[51] Li Lu, Runzhe Wang, Jing Ding, Wubin Mao, Wei Chen, and Hongzi Zhu. Fastid : An undeceived router for real-time identification of wifi terminals. In Proceedings of the 14th IFIP Networking Conference, Networking 2015, Toulouse, France, 20-22 May, 2015, pages 1–9, 2015. doi : 10.1109/IFIPNetworking.2015.7145328. URL https://doi.org/10.1109/IFIPNetworking.2015.7145328.

[52] Chrisil Arackaparambil, Sergey Bratus, Anna Shubina, and David Kotz. On the reliability of wireless fingerprinting using clock skews. In Proceedings of the Third ACM Conference on Wireless Network Security, WiSec '10, pages 169–174, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-923-7. doi : 10.1145/1741866.1741894. URL http://doi.acm.org/10.1145/1741866.1741894.

[53] D. Huang, W. Teng, C. Wang, H. Huang, and J. M. Hellerstein. Clock skew based node identification in wireless sensor networks. In IEEE GLOBECOM 2008 - 2008 IEEE Global Telecommunications Conference, pages 1–5, Nov 2008. doi : 10.1109/GLOCOM.2008.ECP.363.

[54] Fabian Lanze, Andriy Panchenko, Benjamin Braatz, and Andreas Zinnen. Clock skew based remote device fingerprinting demystified. In 2012 IEEE Global Communications Conference, GLOBECOM 2012, Anaheim, CA, USA, December 3-7, 2012, pages 813–819, 2012. doi : 10.1109/GLOCOM.2012.6503213. URL https://doi.org/10.1109/GLOCOM.2012.6503213.

[55] Jeffrey Pang, Ben Greenstein, Ramakrishna Gummadi, Srinivasan Seshan, and David Wetherall. 802.11 user fingerprinting. In Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking, MobiCom07, pages 99–110, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-681-3. doi : 10.1145/1287853.1287866. URL http://doi.acm.org/10.1145/1287853.1287866.

[56] K. N. Gopinath, Pravin Bhagwat, and K. Gopinath. An empirical analysis of heterogeneity in ieee 802.11 mac protocol implementations and its implications. In Proceedings of the 1st International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization, WiNTECH '06, pages 80–87, New York, NY, USA, 2006. ACM. ISBN 1-59593-539-8. doi : 10.1145/1160987.1161002. URL http://doi.acm.org/10.1145/1160987.1161002.

[57] Christoph Neumann, Olivier Heen, and Stéphane Onno. An empirical study of passive 802.11 device fingerprinting. 2012 32nd International Conference on Distributed Computing Systems Workshops, pages 593–602, 2012.

[58] Jason Franklin, Damon McCoy, Parisa Tabriz, Vicentiu Neagoe, Jamie Van Randwyk, and Douglas Sicker. Passive data link layer 802.11 wireless device driver fingerprinting. In Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1267336.1267348.

[59] Piers O'hanlon, Ravishankar Borgaonkar, and Lucca Hirshi. Mobile subscriber wifi privacy. In IEEE Symposium on Security and Privacy 2017 workshop on Mobile Security Technologies, 2017. (Best Paper Award).

[60] Pieter Robyns, Bram Bonné, Peter Quax, and Wim Lamotte. Non-cooperative 802.11 MAC layer fingerprinting and tracking of mobile devices, April 2017. URL https://doi.org/10.5281/zenodo.545970. This work was funded by a FWO SBO grant.

[61] Suranga Seneviratne, Fangzhou Jiang, Mathieu Cunche, and Aruna Seneviratne. SSIDs in the Wild : Extracting Semantic Information from WiFi SSIDs. In The 40th IEEE Conference on Local Computer Networks (LCN), Clearwater Beach, Florida, United States, October 2015. URL https://hal.inria.fr/hal-01181254.

[62] Alessandro Mei Adriano Di Luzio and Julinda Stefa. Mind your probes : De-anonymization of large crowds through smartphone wifi probe requests, 2016.

[63] 2016. URL https://wigle.net.

[64] Mathieu Cunche, Mohamed Ali Kaafar, and Roksana Boreli. I know who you will meet this evening ! Linking wireless devices using Wi-Fi probe requests. In WoWMoM - 13th IEEE International Symposium on a World of Wireless, Mobile and Multimedi San Francisco, United States, June 2012. URL https://hal.inria.fr/hal-00747825.

[65] Shuja Jamil, Sohaib Khan, Anas Basalamah, and Ahmed Lbath. Classifying smartphone screen on/off state based on wifi probe patterns. In Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing : Adjunct, UbiComp '16, pages 301–304, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4462-3. doi : 10.1145/2968219.2971377. URL http://doi.acm.org/10.1145/2968219.2971377.

[66] Davide Sanvito Alessandro E. C. Redondi and Matteo Cesana. Passive classification of wi-fi enabled devices, 2016.

[67] Shin Sungjin, Han Donghyuk, Cho Hyoungjun, and Chung Jong-Moon. Improved association and disassociation scheme for enhanced wlan handover and vho mobile information systems, 2016.

[68] Célestin Matte, Jagdish P. Achara, and Mathieu Cunche. Short : Device-to-Identity Linking Attack Using Targeted Wi-Fi Geolocation Spoofing. In ACM WiSec 2015, New York, United States, June 2015. doi : 10.1145/2766498.2766521. URL https://hal.inria.fr/hal-01176842.

[69] Kai Tao, Jing Li, and Srinivas Sampalli. Detection of Spoofed MAC Addresses in 802.11 Wireless Networks, pages 201–213. Springer Berlin Heidelberg, Berlin,

Heidelberg, 2009. ISBN 978-3-540-88653-2. doi : 10.1007/978-3-540-88653-2-15. URL https://doi.org/10.1007/978-3-540-88653-2-15.

[70] 2016. URL https://play.google.com/store/apps/details?id=sh.geofence.tracker.tasker.

[71] A. B. M. Musa and Jakob Eriksson. Tracking unmodified smartphones using wi-fi monitors. In Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems, SenSys '12, pages 281–294, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1169-4. doi : 10.1145/2426656.2426685. URL http://doi.acm.org/10.1145/2426656.2426685.

[72] D.Fung. How stores use your phone's wifi to track your shopping habits ?, 2013. URL https://www.washingtonpost.com/news/the-switch/wp/2013/10/19/how-stores-use-your-phones-wifi-to-track-your-shopping-habits/?utm_term=.146152c56b14.

[73] Khuong An Nguyen and Zhiyuan Luo. Selective mixture of gaussians clustering for location fingerprinting. In Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems : Computing, Networking and Services, MOBIQUITOUS '14, pages 328–337, ICST, Brussels, Belgium, Belgium, 2014. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-1-63190-039-6. doi : 10.4108/icst.mobiquitous.2014.257983. URL http://dx.doi.org/10.4108/icst.mobiquitous.2014.257983.

[74] Julien Freudiger. How talkative is your mobile device ? : An experimental study of wi-fi probe requests. In Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks, WiSec '15, pages 8 :1–8 :6, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3623-9. doi : 10.1145/2766498.2766517. URL http://doi.acm.org/10.1145/2766498.2766517.

[75] Levent Demir, Mathieu Cunche, and Cédric Lauradoux. Analysing the privacy policies of wi-fi trackers. In Proceedings of the 2014 workshop on physical analytics, pages 39–44. ACM, 2014.

[76] Ashkan Soltani. Privacy trade-offs in retail tracking, 2015. URL https://www.ftc.gov/news-events/blogs/techftc/2015/04/privacy-trade-offs-retail-tracking.

[77] Higgins and Tien. Mobile tracking code of conduct falls short of protecting consumers, 2013.

[78] Paolo Palmieri, Luca Calderoni, and Dario Maio. Spatial bloom filters : Enabling privacy in location-aware applications. In Information Security and

Cryptology - 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers, pages 16–36, 2014. doi : 10.1007/978-3-319-16745-9_2. URL http://dx.doi.org/10.1007/978-3-319-16745-9_2.

[79] Luca Calderoni, Paolo Palmieri, and Dario Maio. Location privacy without mutual trust : The spatial bloom filter. Computer Communications, 68 :4–16, 2015. doi : 10.1016/j.comcom.2015.06.011. URL http://dx.doi.org/10.1016/j.comcom.2015.06.011.

[80] A. R. Beresford and F. Stajano. Mix zones : user privacy in location-aware services. In IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second, pages 127–131, March 2004. doi : 10.1109/PERCOMW.2004.1276918.

[81] Balaji Palanisamy and Ling Liu. Mobimix : Protecting location privacy with mix-zones over road networks. In Proceedings of the 2011 IEEE 27th International Conference on Data Engineering, ICDE '11, pages 494–505, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-1-4244-8959-6. doi : 10.1109/ICDE.2011.5767898. URL http://dx.doi.org/10.1109/ICDE.2011.5767898.

[82] Marco Gruteser and Dirk Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In Proceedings of the 1st International Conference on Mobile Systems, Applications and Services, MobiSys '03, pages 31–42, New York, NY, USA, 2003. ACM. doi : 10.1145/1066116.1189037. URL http://doi.acm.org/10.1145/1066116.1189037.

[83] Janne Lindqvist, Tuomas Aura, George Danezis, Teemu Koponen, Annu Myllyniemi, Jussi Mäki, and Michael Roe. Privacy-preserving 802.11 access-point discovery. In Proceedings of the Second ACM Conference on Wireless Network Security, WiSec '09, pages 123–130, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-460-7. doi : 10.1145/1514274.1514293. URL http://doi.acm.org/10.1145/1514274.1514293.

[84] Jeffrey Pang and Srinivasan Seshan. Tryst : The case for confidential service discovery. In In HotNets VI : The Sixth Workshop on Hot Topics in Networks, 2007.

[85] L. Demir, A. Kumar, M. Cunche, and C. Lauradoux. The pitfalls of hashing for privacy. IEEE Communications Surveys Tutorials, PP(99) :1–1, 2017. doi : 10.1109/COMST.2017.2747598.

[86] J. Novak K. Skinner. Privacy and your app. in apple worldwide dev. conf. (wwdc), 2015.

[87] W. Wang. Wireless networking in windows 10. in windows hardware engineering community conference (winhec), 2015.

[88] Android 6.0 changes, 2015. URL https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html.

[89] E. Grumbach. iwlwifi : mvm : support random mac address for scanning. linux commit effd05ac479b, 2016.

[90] Jeremy Martin, Travis Mayberry, Collin Donahue, Lucas Foppe, Lamont Brown, Chadwick Riggins, Erik C. Rye, and Dane Brown. A study of mac address randomization in mobile devices and when it fails. CoRR, abs/1703.02874, 2017. URL http://arxiv.org/abs/1703.02874.

[91] Célestin Matte, Mathieu Cunche, Franck Rousseau, and Mathy Vanhoef. Defeating MAC Address Randomization Through Timing Attacks. In ACM WiSec 2016, Darmstadt, Germany, July 2016. doi : 10.1145/2939918.2939930. URL https://hal.inria.fr/hal-01330476.

[92] Mathy Vanhoef, Célestin Matte, Mathieu Cunche, Leonardo S. Cardoso, and Frank Piessens. Why mac address randomization is not enough : An analysis of wi-fi network discovery mechanisms. In Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, ASIA CCS 16, pages 413–424, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4233-9. doi : 10.1145/2897845.2897883. URL http://doi.acm.org/10.1145/2897845.2897883.

[93] Wahhab Albazrqaoe. A study of bluetooth frequency hopping sequence : modeling and a practical attack, 2011.

[94] Simon Hay and Robert Harle. Bluetooth Tracking without Discoverability, pages 120–137. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-01721-6. doi : 10.1007/978-3-642-01721-6_8. URL https://doi.org/10.1007/978-3-642-01721-6_8.

[95] J. Hallberg, M. Nilsson, and K. Synnes. Positioning with bluetooth. In 10th International Conference on Telecommunications, 2003. ICT 2003., volume 2, pages 954–958 vol.2, Feb 2003. doi : 10.1109/ICTEL.2003.1191568.

[96] R. Schroer. Position, location, and navigation symposium (plans). IEEE Aerospace and Electronic Systems Magazine, 19(2) :33–34, Feb 2004. ISSN 0885-8985. doi : 10.1109/MAES.2004.1269690.

[97] Hui Liu, H. Darabi, P. Banerjee, and Jing Liu. Survey of wireless indoor positioning techniques and systems. Trans. Sys. Man Cyber Part C, 37(6) :1067–1080, November 2007. ISSN 1094-6977. doi : 10.1109/TSMCC.2007.905750. URL http://dx.doi.org/10.1109/TSMCC.2007.905750.

[98] María Rodríguez-Damián, Xosé Antón Vila Sobrino, and Leandro Rodríguez-Liñares. Indoor Tracking Persons Using Bluetooth : A Real Experiment with Different Fingerprinting-Based Algorithms, pages 25–32. Springer International Publishing, Heidelberg, 2013. ISBN 978-3-319-00566-9. doi : 10.1007/978-3-319-00566-9_4. URL https://doi.org/10.1007/978-3-319-00566-9_4.

[99] Markus Jakobsson and Susanne Wetzel. Security Weaknesses in Bluetooth. In Proceedings of the 2001 Conference on Topics in Cryptology : The Cryptographer's Track at RSA, CT-RSA 2001, pages 176–191, London, UK, UK, 2001. Springer-Verlag. URL http://dl.acm.org/citation.cfm?id=646139.680779.

[100] B. S. Peterson, R. O. Baldwin, and J. P. Kharoufeh. Bluetooth Inquiry Time Characterization and Selection. Mobile Computing, IEEE Transactions on, 5(9) : 1173–1187, September 2006. doi : 10.1109/TMC.2006.125. URL http://dx.doi.org/10.1109/TMC.2006.125.

[101] Ford-Long Wong and Frank Stajano. Location Privacy in Bluetooth. In Proceedings of the Second European Conference on Security and Privacy in Ad-Hoc and Sensor Networks, ESAS'05, pages 176–188, Berlin, Heidelberg, 2005. Springer-Verlag.

[102] Ollie Whitehouse. War nibbling : Bluetooth insecurity, 2003.

[103] Daniel Cross, Justin Hoeckle, Michael Lavine, Jason Rubin, and Kevin Snow. Detecting Non-Discoverable Bluetooth Devices, pages 281–293. Springer US, Boston, MA, 2008. ISBN 978-0-387-75462-8. doi : 10.1007/978-0-387-75462-8_20. URL https://doi.org/10.1007/978-0-387-75462-8_20.

[104] Joshua Wright and Johnny Cache. Hacking Exposed Wireless : Wireless Security Secrets & Solutions. McGraw-Hill Education Group, 3rd edition, 2015. ISBN 0071827633, 9780071827638.

[105] Martin Herfurt and Collin Mulliner. Remote device identification based on bluetooth fingerprinting techniques, 2004.

[106] Dominic Spill. Ubertooth one, 2015. URL http://www.shmoocon.org/schedule#ubertooth.

[107] Dominic Spill. Bluetooth Packet Sniffing Using Project Ubertooth, 2012.

[108] Dominic Spill. Discovering the Bluetooth UAP, 2014.

[109] Jun Huang, Wahhab Albazrqaoe, and Guoliang Xing. Blueid : A practical system for bluetooth device identification. In 2014 IEEE Conference on Computer Communications, INFOCOM 2014, Toronto, Canada, April 27 - May 2, 2014, pages 2849–2857, 2014. doi : 10.1109/INFOCOM.2014.6848235. URL https://doi.org/10.1109/INFOCOM.2014.6848235.

[110] Marc Haase, Matthias Handy, and Matthias H. Bluetrack - imperceptible tracking of bluetooth devices. In In Ubicomp Poster Proceedings, 2004.

[111] Mathias Versichele, Tijs Neutens, Matthias Delafontaine, and Nico V. de Weghe. The use of Bluetooth for analysing spatiotemporal dynamics of human movement at mass events : A case study of the Ghent Festivities. Applied Geography, 32 (2) :208–220, 2012. doi : http://dx.doi.org/10.1016/j.apgeog.2011.05.011. URL http://www.sciencedirect.com/science/article/pii/S0143622811000944.

[112] M Pels, J Barhorst, M Michels, R Hobo, and Jeffrey Barendse. Tracking people using bluetooth : Implications of enabling bluetooth discoverable mode. Technical report, University of Amsterdam, 09 2005.

[113] Matthias Delafontaine, Mathias Versichele, Tijs Neutens, and Nico V. de Weghe. Analysing spatiotemporal sequences in Bluetooth tracking data. Applied Geography, 34 :659–668, 2012. doi : http://dx.doi.org/10.1016/j.apgeog.2012.04.003. URL http://www.sciencedirect.com/science/article/pii/S014362281200029X.

[114] Samuel K. Opoku. An Indoor Tracking System Based on Bluetooth Technology. CoRR, abs/1209.3053, 2012. URL http://arxiv.org/abs/1209.3053.

[115] Mathias Versichele, Tijs Neutens, and Nico Van de Weghe. Person monitoring with Bluetooth tracking. In Chiara Renso, Stefano Spaccapietra, and Esteban Zimányi, editors, Mobility data : modeling, management, and understanding, pages 277–294. Cambridge University Press, 2013.

[116] Simon Hay and Robert Harle. Bluetooth tracking without discoverability. Location and Context Awareness, pages 120–137, 2009. URL http://dx.doi.org/10.1007/978-3-642-01721-6_8.

[117] S. Lujan, J. Suardiaz-Muro, A. Cabrera-Lozoya, and F. Cerdan. Deeploc : Discreet Indoor People Location Application. In New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on, pages 1–5, February 2011. doi : 10.1109/NTMS.2011.5720615. URL http://dx.doi.org/10.1109/NTMS.2011.5720615.

[118] Simon Hay and Robert Harle. Bluetooth Tracking Without Discoverability. In Proceedings of the 4th International Symposium on Location and Context Awareness, LoCA '09, pages 120–137, Berlin, Heidelberg, 2009. Springer-Verlag. doi : 10.1007/978-3-642-01721-6_8. URL http://dx.doi.org/10.1007/978-3-642-01721-6_8.

[119] Yaniv Shaked and Avishai Wool. Cracking the bluetooth pin. In Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services, MobiSys '05, pages 39–50, New York, NY, USA, 2005. ACM. ISBN 1-931971-31-5. doi : 10.1145/1067170.1067176. URL http://doi.acm.org/10.1145/1067170.1067176.

[120] Keijo Haataja and Pekka Toivanen. Practical man-in-the-middle attacks against bluetooth secure simple pairing. In 2008 4th International Conference on Wireless Communications, Networking and Mobile Computing, pages 1–5. IEEE, 2008.

[121] Keijo MJ Haataja and Konstantin Hypponen. Man-in-the-middle attacks on bluetooth : a comparative analysis, a novel attack, and countermeasures. In Communications, Control and Signal Processing, 2008. ISCCSP 2008. 3rd International Symposium on, pages 1096–1102. IEEE, 2008.

[122] Aveek K. Das, Parth H. Pathak, Chen-Nee Chuah, and Prasant Mohapatra. Uncovering privacy leakage in ble network traffic of wearable fitness trackers. In Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications, HotMobile '16, pages 99–104, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4145-5. doi : 10.1145/2873587.2873594. URL http://doi.acm.org/10.1145/2873587.2873594.

[123] Mike Ryan. Bluetooth : With low energy comes low security. In WOOT, 2013.

[124] GIT Crackle.
urlhttps ://github.com/mikeryan/crackle/, 2014.

[125] Aiguo Zhang, Ying Yuan, Qunyong Wu, Shunzhi Zhu, and Jian Deng. Wireless localization based on rssi fingerprint feature vector. International Journal of Distributed Sensor Networks, 11(11) :528747, 2015. doi : 10.1155/2015/528747. URL https://doi.org/10.1155/2015/528747.

[126] Bluetooth Specification v4.2, December 2014.

[127] John M. Sandidge. Bluetooth frequency hop prediction. Technical report, Texas Tech University, 2002.

[128] Pierre Rouveyrol, Patrice Raveneau, and Mathieu Cunche. Large Scale Wi-Fi tracking using a Botnet of Wireless Routers. In Workshop on Surveillance & Technology, Philadelphia, United States, June 2015. URL https://hal.inria.fr/hal-01151446.

[129] Official linux bluetooth protocol stack, 2005. URL http://www.bluez.org.

[130] Keuchul Cho, Woojin Park, Moonki Hong, Gisu Park, Wooseong Cho, Jihoon Seo, and Kijun Han. Analysis of latency performance of bluetooth low energy (ble) networks. Sensors, 15(1) :59–78, 2014.

[131] Varun Kumar Digumber Raj Kishen Moloo. Low-cost mobile gps tracking solution. In 2011 International Conference on Business Computing and Global Informatization. IEEE, 2011.

[132] M. Behzad, A. Sana, M.A. Khan, Z. Walayat, U. Qasim, Z.A. Khan, and N. Javaid. Design and development of a low cost ubiquitous tracking system. Procedia Computer Science, 34 :220 – 227, 2014. ISSN 1877-0509. doi : https://doi.org/10.1016/j.procs.2014.07.014. URL http://www.sciencedirect.com/science/article/pii/S1877050914008692. The 9th International Conference on Future Networks and Communications (FNC'14)/The 11th International Conference on Mobile Systems and Pervasive Computing (MobiSPC'14)/Affiliated Workshops.

[133] Ramsey Faragher and Robert Harle. An analysis of the accuracy of bluetooth low energy for indoor positioning applications. In Proceedings of the 27th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS+'14), pages 201–210, 2014.

[134] P. C. Deepesh, Rashmita Rath, Akshay Tiwary, Vikram N. Rao, and N. Kanakalata. Experiences with using ibeacons for indoor positioning. In Proceedings of the 9th India Software Engineering Conference, ISEC '16, pages 184–189, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4018-2. doi : 10.1145/2856636.2856654. URL http://doi.acm.org/10.1145/2856636.2856654.

[135] L. Vincent. Crowd-sourced information for interior localization and navigation, nov 2012. URL https://www.google.com/patents/US8320939. US Patent 8,320,939.

[136] Dejun Yang, Guoliang Xue, Xi Fang, and Jian Tang. Crowdsourcing to smartphones : Incentive mechanism design for mobile phone sensing. In Proceedings of the 18th Annual International Conference on Mobile Computing and Networking,

Mobicom '12, pages 173–184, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1159-5. doi : 10.1145/2348543.2348567. URL http://doi.acm.org/10.1145/2348543.2348567.

[137] G. Chatzimilioudis, A. Konstantinidis, C. Laoudias, and D. Zeinalipour-Yazti. Crowdsourcing with smartphones. IEEE Internet Computing, 16(5) :36–44, Sept 2012. ISSN 1089-7801. doi : 10.1109/MIC.2012.70.

[138] J. Zhu, K. Zeng, K. Kim, and P. Mohapatra. Improving crowd-sourced wi-fi localization systems using bluetooth beacons. In 2012 9th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), pages 290–298, June 2012. doi : 10.1109/SECON.2012.6275790.

[139] Kassem Fawaz, Kyu-Han Kim, and Kang G. Shin. Protecting privacy of ble device users. In 25th USENIX Security Symposium (USENIX Security 16), pages 1205–1221, Austin, TX, 2016. USENIX Association. ISBN 978-1-931971-32-4. URL https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/fawaz.

[140] G. Wang, B. Wang, T. Wang, A. Nika, H. Zheng, and B. Y. Zhao. Ghost riders : Sybil attacks on crowdsourced mobile mapping services. IEEE/ACM Transactions on Networking, 26(3) :1123–1136, June 2018. ISSN 1063-6692. doi : 10.1109/TNET.2018.2818073.

[141] Mathy Vanhoef, Célestin Matte, Mathieu Cunche, Leonardo S Cardoso, and Frank Piessens. Why mac address randomization is not enough : An analysis of wi-fi network discovery mechanisms. In Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, pages 413–424. ACM, 2016.

UNIVERSITÉ DE LA RÉUNION

Pôle Recherche
Ecoles doctorales

LETTRE D'ENGAGEMENT DE NON-PLAGIAT

Je, soussigné(e) ISSOUFALY Taher en ma qualité de doctorant(e) de l'Université de La Réunion, déclare être conscient(e) que le plagiat est un acte délictueux passible de sanctions disciplinaires. Aussi, dans le respect de la propriété intellectuelle et du droit d'auteur, je m'engage à systématiquement citer mes sources, quelle qu'en soit la forme (textes, images, audiovisuel, internet), dans le cadre de la rédaction de ma thèse et de toute autre production scientifique, sachant que l'établissement est susceptible de soumettre le texte de ma thèse à un logiciel anti-plagiat.

Fait à Saint-Denis le : 22/06/2019

Signature :

Extrait du Règlement intérieur de l'Université de La Réunion
(validé par le Conseil d'Administration en date du 11 décembre 2014)

Article 9. Protection de la propriété intellectuelle – Faux et usage de faux, contrefaçon, plagiat

L'utilisation des ressources informatiques de l'Université implique le respect de ses droits de propriété intellectuelle ainsi que ceux de ses partenaires et plus généralement, de tous tiers titulaires de tes droits.
En conséquence, chaque utilisateur doit :
- utiliser les logiciels dans les conditions de licences souscrites ;
- ne pas reproduire, copier, diffuser, modifier ou utiliser des logiciels, bases de données, pages Web, textes, images, photographies ou autres créations protégées par le droit d'auteur ou un droit privatif, sans avoir obtenu préalablement l'autorisation des titulaires de ces droits.

La contrefaçon et le faux
Conformément aux dispositions du code de la propriété intellectuelle, toute représentation ou reproduction intégrale ou partielle d'une œuvre de l'esprit faite ans le consentement de son auteur est illicite et constitue un délit pénal.
L'article 444-1 du code pénal dispose : « Constitue un faux toute altération frauduleuse de la vérité, de nature à cause un préjudice et accomplie par quelque moyen que ce soit, dans un écrit ou tout autre support d'expression de la pensée qui a pour objet ou qui peut avoir pour effet d'établir la preuve d'un droit ou d'un fait ayant des conséquences juridiques ».
L'article L335_3 du code de la propriété intellectuelle précise que : « Est également un délit de contrefaçon toute reproduction, représentation ou diffusion, par quelque moyen que ce soit, d'une œuvre de l'esprit en violation des droits de l'auteur, tels qu'ils sont définis et réglementés par la loi. Est également un délit de contrefaçon la violation de l'un des droits de l'auteur d'un logiciel (…) ».

Le plagiat est constitué par la copie, totale ou partielle d'un travail réalisé par autrui, lorsque la source empruntée n'est pas citée, quel que soit le moyen utilisé. Le plagiat constitue une violation du droit d'auteur (au sens des articles L 335-2 et L 335-3 du code de la propriété intellectuelle). Il peut être assimilé à un délit de contrefaçon. C'est aussi une faute disciplinaire, susceptible d'entraîner une sanction.
Les sources et les références utilisées dans le cadre des travaux (préparations, devoirs, mémoires, thèses, rapports de stage…) doivent être clairement citées. Des citations intégrales peuvent figurer dans les documents rendus, si elles sont assorties de leur référence (nom d'auteur, publication, date, éditeur…) et identifiées comme telles par des guillemets ou des italiques.

Les délits de contrefaçon, de plagiat et d'usage de faux peuvent donner lieu à une sanction disciplinaire indépendante de la mise en œuvre de poursuites pénales.