Étude expérimentale de la sécurité du protocole WirelessHART

Kais Sellami[1], Romain Cayre[1,2], Elies Tali[2], Pierre Ayoub[2], Vincent Nicomette[1,2] et Guillaume Auriol[1,2]
prenom.nom@insa-toulouse.fr[1]
prenom.nom@laas.fr[2]

[1] Univ. Toulouse, INSA, France
[2] LAAS-CNRS, Toulouse, France

Résumé. Les réseaux de capteurs industriels s'appuient sur des mécanismes de communication déterministes et sécurisés afin de répondre à des contraintes strictes de fiabilité et de temps réel. WirelessHART est un protocole sans fil basé sur la norme IEEE 802.15.4 qui répond à ces exigences. Il constitue une extension sans fil du protocole HART, offrant une communication bidirectionnelle entre instruments industriels dits intelligents. Malgré son utilisation au sein d'environnements industriels potentiellement sensibles, la sécurité de ce protocole a été peu étudiée et essentiellement de façon théorique. En effet, peu d'outils sont aujourd'hui disponibles pour analyser et surveiller ce protocole pourtant critique, limitant significativement l'étude des mécanismes de sécurité déployés. Dans cet article, nous décrivons un protocole expérimental qui nous a permis d'évaluer en pratique la sécurité du protocole WirelessHART. Nous avons notamment déployé un environnement réaliste sous la forme d'un réseau de capteurs industriels Dust (Analog Devices), et étendu le framework WHAD en développant un sniffer nous permettant de capturer et d'injecter du trafic au sein du réseau. Sur cette base, nous avons pu évaluer plusieurs attaques de l'état de l'art reposant sur l'injection de paquets malveillants, ainsi qu'identifier une attaque de désynchronisation lors de nos expériences. Nous présentons le modèle de menace considéré et la conception des primitives nécessaires pour la mise en œuvre pratique de ces attaques.

1 Introduction

Les réseaux de capteurs industriels reposent sur des mécanismes de communication déterministes et robustes afin de satisfaire des contraintes strictes de fiabilité et de temps réel. Dans ce contexte, WirelessHART s'est imposé comme un standard industriel largement déployé pour la communication sans fil entre instruments de mesure et systèmes de contrôle, notamment dans les secteurs de l'énergie, de la chimie et du raffinage. Des déploiements concrets ont par exemple été réalisés pour la surveillance de

bacs de stockage dans des raffineries, le suivi de la corrosion, ou encore l'instrumentation des cuves de mélange dans le domaine pharmaceutique, comme documenté dans une étude de cas du *FieldComm Group* [12]. WirelessHART constitue une extension sans fil du protocole HART (Highway Addressable Remote Transducer), un protocole de communication industriel permettant la transmission simultanée de données analogiques et numériques. WirelessHART s'appuie sur la norme IEEE 802.15.4, un standard de communication sans fil à faible consommation. Le protocole utilise un multiplexage temporel avec saut de fréquence (TSCH) et une gestion centralisée du réseau afin d'assurer la robustesse des échanges et la compatibilité avec les infrastructures industrielles existantes. Ces mécanismes, bien qu'efficaces d'un point de vue opérationnel, compliquent fortement l'observation et l'analyse du trafic radio. En effet, malgré son déploiement répandu [18] dans les environnements industriels, la sécurité du WirelessHART n'a été que peu étudiée par la communauté scientifique. Cette situation s'explique notamment par la complexité du protocole, fondé sur des mécanismes de communication déterministes et de saut de fréquence, ainsi que le manque d'outils ouverts dédiés à l'écoute et à l'injection de trames. A notre connaissance, les attaques documentées dans la littérature se concentrent principalement sur des scénarios de déni de service par brouillage radio [1, 7, 9, 19], sur des travaux de rétro-conception matérielle [15], ou encore sur des attaques décrites de manière théorique à partir de l'étude de la spécification du protocole [2, 3, 10, 20]. Aussi, des vérifications formelles du protocole ont été menées [13, 16], menant par exemple à la découverte d'une attaque exploitant le protocole de changement de clés du WirelessHART. En ce qui concerne les expérimentations existantes, elles s'appuient principalement sur des approches basées sur des radios logicielles (SDR) [6, 10], plus coûteuses, ou encore sur des simulateurs [2, 3].

Dans cet article, nous proposons une analyse expérimentale de la sécurité de WirelessHART fondée sur l'observation réelle du trafic radio. Nous avons notamment conçu un sniffer dédié reposant sur le framework open-source WHAD et un dongle radio embarquant un System-on-Chip (SoC) nRF52840 de Nordic Semiconductor (nRF52840-PCA10059). Afin de reproduire un environnement industriel réaliste, nous avons également déployé un réseau de capteurs industriels « Dust » d'Analog Devices, reposant sur le protocole WirelessHART. Nous avons ainsi pu réaliser une analyse détaillée du trafic, nous permettant d'étudier le fonctionnement de fonctionnalités bas niveau telles que les échanges de contrôle, la channel

map et l'allocation des liens de communication. Cette approche nous a également permis de reproduire et d'étendre des attaques existantes.

Organisation La section 2 propose un aperçu des fonctionnalités principales du protocole WirelessHART. La section 3 décrit le modèle d'attaque ainsi que les attaques reproduites et découvertes. La section 4 décrit les expérimentations réalisées ainsi que leurs résultats. La section 5 propose quelques contre-mesures à nos attaques et la section 6 conclut cet article.

2 Fondamentaux du protocole WirelessHART

WirelessHART opère dans la bande ISM des 2,4 GHz et repose sur la couche physique IEEE 802.15.4 [11]. Le protocole s'appuie sur un mécanisme de communication à créneaux temporels (slots) (*Time Division Multiple Access*, TDMA) et de saut de fréquence (*Frequency Hopping Spread Spectrum*, FHSS) améliorant la résilience face aux interférences radio et aux pertes de paquets. Les échanges sont organisés en slots d'une durée fixe de *10ms* et répartis dynamiquement sur différents canaux radio. L'ensemble du réseau est géré de manière centralisée par un gestionnaire de réseau (dit *network manager*) et un gestionnaire de sécurité (dit *security manager*), généralement implémentés sous la forme de nœuds logiques intégrés directement dans la passerelle (*gateway*) [8, 17]. Ces composants sont notamment responsables de l'authentification des équipements, de la distribution des clés de sécurité, du routage et de l'allocation des slots de communication au sein d'une ou plusieurs *superframes* (cycles de communication répétitifs).

Les réseaux WirelessHART adoptent une topologie maillée comme le montre la figure 1 dans laquelle les équipements de terrain (*motes/field devices/nœuds*) peuvent relayer les messages d'autres dispositifs. Plusieurs routes redondantes peuvent être établies entre un équipement et la *gateway* afin d'augmenter la fiabilité des communications dans des environnements radio contraints. Si ces mécanismes renforcent la robustesse du réseau, ils introduisent également des contraintes temporelles et structurelles fortes, qui ont un impact direct sur l'observation du trafic et l'analyse de la sécurité du protocole. WirelessHART supporte différents modes de communication, incluant la publication périodique de données telles que des mesures de capteurs, les notifications déclenchées par des événements, des données de contrôle ainsi que des échanges de type requête/réponse. Ces modes influencent les schémas de trafic et les décisions d'ordonnancement, et constituent des éléments clés pour l'analyse des messages de contrôle et l'identification de vecteurs d'attaque potentiels.

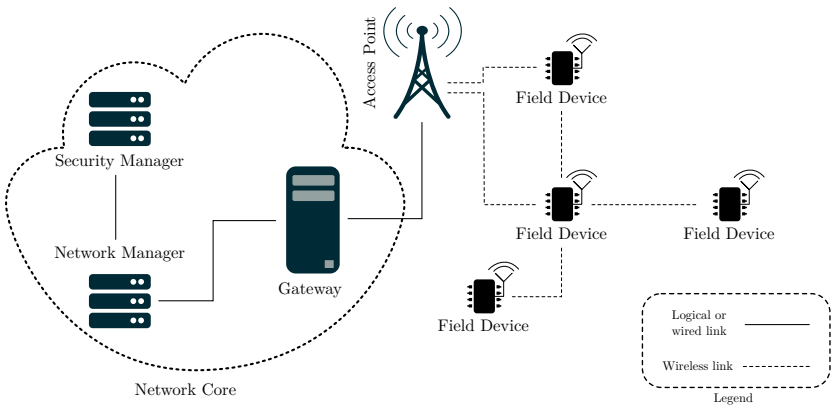

Fig. 1. Architecture d'un réseau maillé WirelessHART.

2.1 Architecture des réseaux WirelessHART

L'initialisation du réseau repose sur la configuration préalable des points d'accès (*Access Points*). Le *network manager* établit une communication sécurisée avec la *gateway*, puis distribue aux points d'accès les paramètres nécessaires au fonctionnement du réseau, incluant la liste des *superframes*, les graphes de communication ainsi que les liens dédiés aux phases d'association d'un mote au sein du réseau (*joining*) et de communication partagée.

Le processus de mise en réseau d'un mote se déroule en trois phases principales : la diffusion de données d'annonce (*advertising*), l'association du mote au sein du réseau (*joining*) et la négociation des paramètres. Lors de la phase d'annonce, les points d'accès — et certains motes autorisés — diffusent périodiquement des messages contenant les identifiants du réseau et des informations temporelles permettant aux nouveaux équipements de se synchroniser. Pour rejoindre le réseau, un mote doit préalablement être configuré avec l'identifiant du réseau ainsi qu'une clé dédiée à l'association des nouveaux motes (*join key*). Cette clé peut être provisionnée au préalable sur l'équipement ou distribuée via la liaison sans fil en utilisant des mécanismes dédiés, spécifiés par le protocole WirelessHART. Le mote utilise ensuite les messages d'annonce pour synchroniser son horloge et transmettre une requête d'association chiffrée sur un lien dédié. Une fois l'association effectuée, les clés de communication de session et celle du réseau sont distribuées au mote via la *gateway*, permettant l'établissement de communications sécurisées avec le reste du réseau [21].

2.2 Communication et saut de fréquence

WirelessHART repose sur une combinaison de multiplexage temporel (TDMA) et de saut de fréquence (FHSS) afin d'assurer des communications déterministes et robustes. Le temps est découpé en slots de 10 ms, regroupés en *superframes*. Un réseau peut définir plusieurs *superframes* de longueurs différentes, en fonction des exigences applicatives et des flux de communication à supporter. Une communication potentielle est planifiée selon un lien (*link*), qui correspond à un échange programmé entre deux équipements. Chaque lien est caractérisé par un identifiant de *superframe*, un indice de slot, un décalage (*offset*), un type de lien et un ensemble d'options. L'identifiant de *superframe* détermine la périodicité du lien, tandis que l'indice de slot définit sa position temporelle précise par rapport au début de la *superframe*. Le décalage est utilisé par l'algorithme de saut de fréquence pour sélectionner le canal radio actif. Les types de liens incluent notamment les liens d'association, de diffusion, de découverte et de communication normale, tandis que les options précisent le sens de communication (émission, réception ou partagé). L'allocation, la mise à jour et la révocation des liens et des *superframes* sont exclusivement assurées par le *network manager*. La figure 2 illustre un exemple d'ordonnancement dans lequel plusieurs *superframes* coexistent et se superposent. Chaque *superframe* possède sa propre périodicité et comporte différents liens.

Le saut de fréquence est réalisé selon un algorithme déterministe. WirelessHART s'appuie sur la couche physique IEEE 802.15.4 et utilise les canaux 11 à 25 de la bande ISM des 2,4 GHz. Un réseau peut toutefois restreindre cet ensemble de canaux via une *channel map*, donnée aux équipements lors du processus d'association ou des paquets spécifiques de mise à jour de cette dernière. Cette *channel map* est représentée par un champ de deux octets, dans lequel chaque bit indique l'activation d'un canal compris dans l'intervalle [11 ; 26[, avec le dernier bit du canal 26 mis à 0, ce dernier n'étant jamais utilisé par le WirelessHART.

Pour chaque slot et chaque lien, le canal actif est calculé à partir du numéro absolu du slot (*Absolute Slot Number* (ASN)) et du décalage associé au lien (*offset*), selon les équations 1 et 2 :

$$\text{channelIndex} = (\text{ASN} + \text{offset}) \bmod N_{\text{actifs}} \tag{1}$$

$$\text{canal} = \text{activeChannelArray}[\text{channelIndex}] \tag{2}$$

où N_{actifs} désigne le nombre de canaux actifs définis dans la *channel map* et l'*activeChannelArray* correspond à la liste ordonnée des canaux

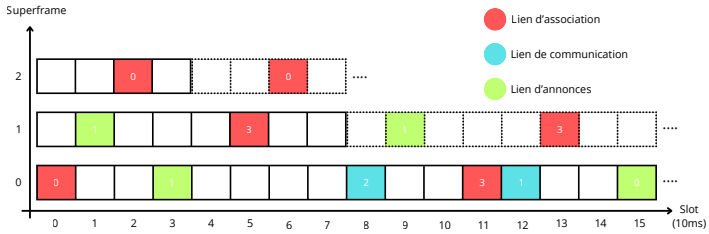

Fig. 2. Planification temporelle des liens WirelessHART et exemple d'offsets (indiqués par des nombres entiers dans les slots utilisés par des liens).

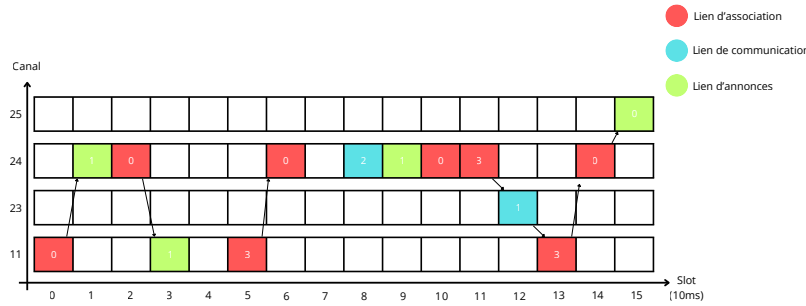

Fig. 3. Exemple de saut de fréquence WirelessHART, calculé selon les équations 1 et 2.

autorisés [14]. Ce mécanisme assure une diversité fréquentielle permettant de réduire l'impact des interférences et du multi-trajet. Toutefois, il impose également une synchronisation temporelle fine entre les équipements.

En complément, WirelessHART met en œuvre un mécanisme de retransmission basé sur un algorithme de temporisation aléatoire (*backoff*) afin d'améliorer la fiabilité en cas de collisions. L'exemple de planification des liens est présenté dans la figure 2, tandis que le changement de canal, résultant de l'ordonnancement des différents liens et de leurs *offsets* respectifs, est illustré par la figure 3.

2.3 Mécanismes de sécurité

WirelessHART intègre des mécanismes de sécurité basés sur des clés cryptographiques. Ces mécanismes sont essentiels pour assurer l'authentification mutuelle des équipements, la confidentialité et l'intégrité des communications.

Hiérarchie des clés cryptographiques WirelessHART utilise trois types de clés (voir "*Network Management Specification*" (*HCF_SPEC-85*) Rev. 4.0 [11, p. 75-79, section 9.4]) :

1. Clé d'association ($\mathcal{K}_{join}$) La clé d'association est utilisée comme premier secret partagé entre un nouveau mote et le *network manager* (préprovisionnée). Son rôle principal est le chiffrement de la requête d'association émise par le mote et des secrets retournés par le *network manager* (la clé $\mathcal{K}_{\text{network}}$ ainsi que la clé $\mathcal{K}_{\text{session\_unicast}}$ entre le nouveau mote et le *network manager*). Elle peut être configurée indépendemment pour chaque nœud ou commune à tout le réseau, cette dernière configuration étant fréquemment observée dans les déploiements par défaut.

2. Clé de réseau ($\mathcal{K}_{network}$) La clé de réseau est une clé maître au niveau de la couche liaison de données (*Data Link Layer*, DLL) contrôlée exclusivement par le *network manager* et partagée par tous les équipements du réseau. Elle est utilisée pour authentifier tous les échanges DLL comme l'indique la figure 4, à l'exception des messages d'annonce et les requêtes d'association (voir "*TDMA Data-Link Layer*" (*HCF_SPEC-75*) Rev. 1.2 [11, p. 36, section 8.4]) et qui peut être modifiée au cours du temps par le *network manager*.

3. Clé de session Les clés de session se divisent en deux catégories et sont utilisées pour l'authentification et le chiffrement au niveau des couches réseau/transport comme l'indique la figure 4 :

- **Clé de session *unicast* ($\mathcal{K}_{\text{session\_unicast}}$)** : Utilisée pour les communications directes (*unicast*) entre deux équipements. Un mote peut disposer de plusieurs clés *unicast*, une pour chaque communicant. Ces clés garantissent la confidentialité des communications en *unicast*.
- **Clé de session *broadcast* ($\mathcal{K}_{\text{session\_broadcast}}$)** : Utilisées pour les communications de groupe (*broadcast*) émises par le *network manager* ou le *security manager* vers l'ensemble des équipements. Ces clés sont communes à tous les récepteurs et transmises par le

network manager chiffrées avec la clé $\mathcal{K}_{\text{join}}$ ou une clé de session lors d'une mise à jour avec la commande 963 *Write/Modify Session* (voir "*Wireless Command Specification*" (*HCF_SPEC-155*) Rev2.0 - 12 juin 2012 [11, p. 121, section 7.99]).

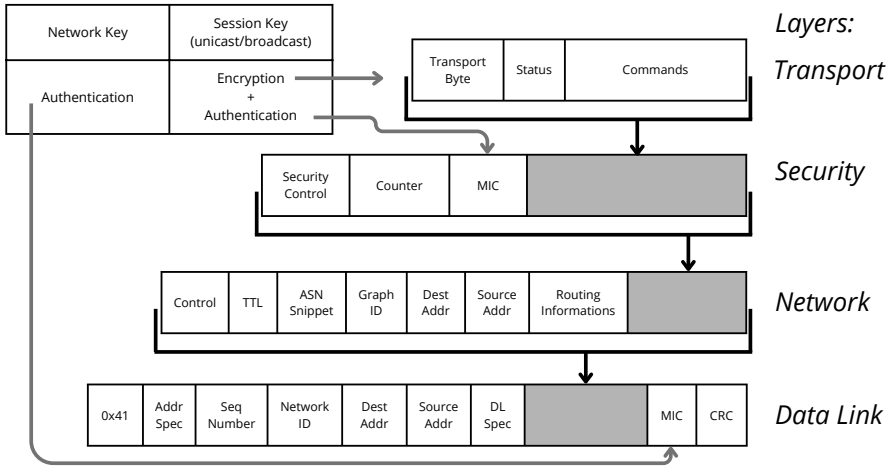

Fig. 4. Intégration des mécanismes de sécurité dans les couches WirelessHART.

Propriétés de sécurité WirelessHART s'appuie sur le chiffrement AES-128 en mode CCM (Counter with CBC-MAC) pour assurer les propriétés suivantes :

- **Confidentialité** : les données de la couche transport sont chiffrées avec les clés de sessions ($\mathcal{K}_{\text{session\_unicast}}$ et $\mathcal{K}_{\text{session\_broadcast}}$).
- **Intégrité (NWK-MIC)** : Un tag d'authentification (NWK-MIC) basé sur les clés de sessions assure l'intégrité des données chiffrées.
- **Anti-rejeu** : Un nonce (dérivé de l'ASN) empêche les attaques par rejeu.
- **Intégrité (DL-MIC)** : Un tag d'authentification au niveau de la DLL basé sur la clé de réseau ($\mathcal{K}_{\text{network}}$) assure que le PDU entier est intègre et provient d'une source authentifiée par le *network manager*.

3 Attaques

3.1 Modèle de menace

Les attaques présentées dans cette section s'inscrivent dans un modèle de menace où l'adversaire dispose d'un accès passif et actif au médium radio WirelessHART, sans nécessiter d'accès privilégié au *network manager*. Le modèle repose sur les hypothèses et capacités suivantes.

Hypothèses de sécurité On considère un attaquant ayant préalablement compromis la clé d'association du réseau. Plusieurs scénarios réalistes peuvent mener à une telle compromission :

- récupération de la clé par extraction de la mémoire d'un mote légitime (par exemple suite à un accès physique ponctuel au mote, dit *trash-can attack*).[3]
- utilisation de clés par défaut connues (`www.hartcomm.org` ou `DUSTNETWORKSROCK` pour les produits Dust), de clés faibles (attaque par force brute) ou prédictibles (attaque par dictionnaire).

L'écoute passive d'une association permet donc à un attaquant connaissant la clé d'association de récupérer l'ensemble des secrets ($\mathcal{K}_{\text{network}}$, ainsi que les clés $\mathcal{K}_{\text{session\_unicast}}$ et $\mathcal{K}_{\text{session\_broadcast}}$) nécessaires à l'injection de trames légitimes.

Objectifs d'un attaquant L'attaquant cherche à intercepter les communications du réseau (impact sur la confidentialité) pour :

- Accéder aux données de mesure transmises par les capteurs (température, pression, débit, etc.).
- Identifier les processus industriels en cours et les paramètres d'exploitation.
- Collecter des informations sur l'infrastructure du réseau.

et compromettre le réseau WirelessHART en :

- Suspendant un ou plusieurs motes (impact sur la disponibilité)
- Usurpant l'identité d'un nœud légitime pour injecter des données malveillantes dans le réseau (impact sur l'intégrité et l'authenticité).

3.2 Déni de service

Les attaques présentées dans cette section exploitent des mécanismes légitimes du protocole WirelessHART afin de provoquer un déni de service partiel ou total du réseau.

[3] Une attaque d'extraction mémoire d'un mote WirelessHART DC9003A-C (Analog Devices) par l'intermédiaire du bus SPI a par exemple été démontrée par Lorente *et al.* [15]

Tableau 1. Résumé de la commande `972 - Suspend Device(s)` [11, p. 133].

Élément	**Description**
Fonction	Suspendre le fonctionnement d'un ou plusieurs motes sur un intervalle d'ASN
Paramètres	ASN de suspension et ASN de reprise (entiers non signés sur 40 bits)
Adressage	*Unicast* ou *broadcast* (commande relayable par les motes)
Restriction d'accès	Commande valide uniquement si émise par le *network manager*

De-authentification massive La de-authentification massive est une attaque de déni de service, dont le principe appliqué au WirelessHART a été théorisé en 2015 dans la thèse de Master de Duijsens [10]. Malgré son impact significatif, cette attaque n'a, à notre connaissance, jamais été évaluée en pratique du fait de contraintes expérimentales (notamment l'absence d'équipements offensifs adaptés). Cette attaque exploite la commande `Suspend` (opcode `0x3CC`) définie dans la spécification du protocole (voir "*Wireless Command Specification*" (*HCF_SPEC-155*) Rev2.0 - 12 juin 2012, section 7.99 [11, p. 133]), dont la structure est reproduite dans le tableau 1.

La requête de la commande `Suspend` ordonne à un mote d'interrompre ses opérations à partir d'un ASN spécifié en paramètre, et ce jusqu'à un ASN de reprise. Lorsqu'elle est acceptée par un mote, cette commande peut être relayée à d'autres équipements légitimes lorsque l'adresse de destination est de type *broadcast*, ce qui peut conduire à la mise en suspension simultanée d'un grand nombre de motes. Un attaquant en connaissance de la clé du *broadcast*, de la clé du réseau et ayant inféré les liens de communications est capable d'engendrer une suspension à grande échelle des motes et un déni de service partiel ou total du réseau sans fil. En cas de réussite, les conséquences d'une telle attaque sur un réseau de capteurs industriels sont potentiellement critiques :

— déni de service des fonctions de mesure et d'actionnement sur de larges portions du réseau ;
— interruption des boucles de contrôle ou des processus industriels dépendant de mises à jour temporelles strictes ;

Cela peut engendrer la nécessité d'une intervention humaine pour rétablir un fonctionnement normal, notamment lorsque l'ASN de suspension est

Tableau 2. Flags de voisinage [11].

Code	Description des flags de voisinage
0x01	Source de temps (modifiable via la commande 971)
0x80	(Lecture seule) Indique l'absence de lien actif vers ce voisin. Ce flag est informatif et est réinitialisé dans la réponse.

très éloigné de l'ASN courant ou lorsque les motes sont en mode *slave* (ils requièrent alors une intervention manuelle pour rejoindre un réseau).

Une attaque similaire avait été théorisée et simulée par Bayou et al. dans [3], en utilisant cette fois-ci des commandes de `Disconnect` mais menant au même résultat. Les mêmes auteurs ont finalement généralisé ce type d'attaques dans [2].

Désynchronisation temporelle Nous avons également identifié une vulnérabilité de déni de service au niveau de la DLL nécessitant uniquement la connaissance de la clé de réseau. Dans le protocole WirelessHART, chaque mote reçoit les flags correspondant à chacun de ses voisins (liaison point à point), présenté dans le tableau 2 (voir "*Wireless Command Specification*" *HCF_SPEC-155* Rev. 2.0, section 7.104 [11, p. 132])

Le mote dépendant ajuste alors son horloge locale à partir du champ *Time Adjustment* présent dans les accusés de réception ou le temps de réception des messages envoyés par le voisin désigné comme source de temps (flag 0x01). Un attaquant capable d'usurper l'adresse du voisin présentant la source de temps peut induire une désynchronisation progressive du mote en injectant des trames légèrement en avance par rapport à l'instant de réception théorique attendu par le mote ciblé. Une attaque similaire a été identifiée par Raza et al. dans [20]. Néanmoins leurs travaux ne mentionnent pas la possibilité de complètement déconnecter un nœud du réseau et ne détaillent pas un schéma d'attaque précis.

Lors d'expérimentations pratiques préliminaires, l'envoi de requêtes *Ping* usurpées, avancées d'environ la moitié de la fenêtre de réception (≈ 1 ms), a conduit le mote à ajuster temporellement son horloge de façon erronée. L'accumulation progressive de ces décalages permet à terme de dépasser la durée de la fenêtre de réception (`TsRxWait`) montré dans la figure 5, empêchant le mécanisme de réception planifiée et entraînant une perte de synchronisation avec le *network manager*. Cette désynchronisation entraîne une déauthentification du mote, nécessitant une reconnexion automatique ou une intervention humaine selon son mode de fonctionnement.

3.3 Usurpation d'identité d'un mote

Les motes d'un réseau utilisant l'implémentation de Dust peuvent fonctionner selon deux modes distincts. En mode *master*, le mote possède la capacité de rejoindre automatiquement le réseau après une déconnexion, sans intervention externe. Il exécute de manière autonome la procédure de reconnexion. En mode *slave*, le mote ne peut pas se reconnecter seul : après une déconnexion, l'API du mote est activée et nécessite une intervention humaine pour relancer la procédure d'association au réseau. Cette attaque cible spécifiquement un mote en mode master, car la reconnexion automatique est essentielle au succès de l'exploitation. En exploitant le comportement de la commande `Suspend`, il est possible de mettre en place une attaque d'usurpation d'identité complète d'un mote légitime du réseau. Cette attaque nécessite la possession des clés $\mathcal{K}_{\text{network}}$ et $\mathcal{K}_{\text{session\_unicast}}$. Une variante de cette attaque est aussi possible en ayant connaissance des clés $\mathcal{K}_{\text{network}}$ et $\mathcal{K}_{\text{session\_broadcast}}$, néanmoins l'attaque ciblera plusieurs nœuds à la fois.

Comportement attendu du protocole Lors d'une suspension planifiée, le mote cible reçoit la commande `Suspend` du *network manager*, envoie un accusé de réception contenant un Response Code (généralement 0 pour succès) confirmant la bonne réception du paquet niveau de la DLL à son voisin qui lui a transmis le paquet et renvoie une réponse de la commande `Suspend` au *network manager* confirmant la prise en compte de la commande, puis se met en sommeil à partir de l'ASN spécifié et jusqu'à l'ASN de reprise. Cette réponse permet au *network manager* de vérifier que la commande a bien été reçue et que le mote va se suspendre comme prévu.

Déroulement de l'attaque L'attaque se déroule en deux étapes. Dans un premier temps, l'attaquant envoie une requête de la commande `Suspend` forgée vers le mote victime, avec un ASN de reprise relativement proche (par exemple, quelques slots après l'envoi du paquet) en usurpant l'identité du *network manager*. Cette suspension courte force le mote à se déconnecter puis à se reconnecter rapidement au réseau. Durant cette phase de reconnexion, l'attaquant écoute passivement le trafic pour capturer les informations suivantes :

— Les clés de session *unicast*.
— La table de liens de communication du mote (ses voisins, les slots alloués, les offsets utilisés).
— Les paramètres de routage et la topologie locale.

Dans un second temps, une fois la reconnexion terminée et après que les secrets soient capturés, l'attaquant injecte immédiatement une seconde commande `Suspend` en se faisant passer pour le *network manager* au niveau de la couche transport et un voisin du mote attaqué au niveau de la DLL avec un ASN de suspension immédiat (égal ou très proche de l'ASN courant) et un ASN de reprise éloigné dans le temps. Cette configuration particulière exploite une subtilité du comportement protocolaire : le mote légitime reçoit la commande et se déconnecte si rapidement qu'il n'a pas le temps d'envoyer son message de réponse au *network manager*. L'absence de cette réponse n'est pas identifiée par le *network manager*, celui-ci n'ayant pas transmis lui-même le paquet injecté. Néanmoins un acquittement sera transmis vers le voisin usurpé par l'attaquant, celui-ci étant transmis dans le même slot que le paquet.

Le mote victime étant maintenant hors ligne pour une très longue durée, l'attaquant peut usurper son identité en utilisant les secrets capturés lors de la première phase de l'attaque. Il forge des trames authentifiées avec l'adresse du mote victime, utilise ses clés de sessions *unicast* pour communiquer avec le *network manager* et ses voisins, et s'intègre dans la topologie en respectant les liens de communication établis. Du point de vue du réseau, l'attaquant apparaît comme le mote légitime ayant simplement repris ses communications après une suspension.

Le *network manager* ne détecte pas l'anomalie pour plusieurs raisons :

— il ne reçoit jamais de réponse à la commande `Suspend` (l'ASN du début du suspend étant proche, le mote n'envoie pas de réponse et se déconnecte malgré ça),
— le mote s'est effectivement déconnecté comme attendu,
— lorsque l'attaquant usurpe l'identité de la victime, les trames sont correctement authentifiées avec les clés de session valides.

Le *network manager* considère donc qu'il s'agit du mote légitime s'étant déconnecté et ayant repris ses activités.

4 Expérimentations

4.1 WHAD

WHAD (Wireless HAcking Devices) [4] est un framework open-source dédié à l'analyse et à l'expérimentation de la sécurité des protocoles sans fil, tels que BLE, ZigBee ou LoRaWAN, présenté en 2025 lors de la conférence SSTIC. Il fournit une interface unifiée entre l'hôte et le matériel radio.

WHAD repose sur un protocole générique permettant aux dispositifs d'annoncer leurs capacités et d'exposer des opérations telles que l'écoute

de paquets ou l'initiation de connexions. Le framework s'articule autour de trois composants : *Whad-client* (logique côté hôte), *ButteRFly* [5] (firmware radio) et *Whad-protocol* (interface de communication standardisée).

Dans le cadre de ces travaux, nous nous sommes basés sur ce framework, que nous avons étendu en y intégrant des outils dédiés à l'analyse du WirelessHart.

4.2 Sniffer WirelessHART

Afin d'évaluer la sécurité de WirelessHART, nous avons notamment développé un sniffer dédié capable de suivre dynamiquement les communications du réseau. Cet outil open source, implémenté sur la base du framework WHAD, réalise un saut de canal synchronisé afin de capturer et d'analyser les échanges en cours.

Architecture et répartition des fonctionnalités Le fonctionnement du sniffer repose sur une répartition entre le firmware embarqué (ButteRFly) et le client WHAD s'exécutant sur l'hôte. Cette séparation est dictée par les contraintes temporelles critiques imposées par le mécanisme TSCH de WirelessHART.

ButteRFly Le firmware prend en charge l'ensemble des opérations nécessitant une réactivité temporelle fine :

- Calcul du canal actif pour chaque slot à partir de l'ASN et de la *channel map*.
- Commutation radio entre canaux (fréquence calculée selon $f_{\text{canal}} = 2405 + 5 \times (\text{canal} - 11)$ MHz).
- Gestion d'un timer matériel pour la synchronisation fine des slots.
- Ajustement temporel à partir des champs *Time Adjustment* contenus dans les accusés de réception.
- Capture des paquets et transmission vers *Whad-client* via *Whad-protocol*.

Cette implémentation bas niveau dans le firmware est indispensable : les slots de 10 ms et les fenêtres de réception de l'ordre de 2 ms ne permettent pas un traitement déporté sur l'hôte, notamment compte tenu de la latence de communication USB et du surcoût d'exécution d'un client Python.

Whad-client Le client, implémenté en Python, se concentre sur les tâches d'analyse de haut niveau :

- Déchiffrement des paquets capturés.

— Génération des paquets à injecter et leur chiffrement.
— Inférence des liens de communication non observés à l'aide d'un algorithme d'exploration.
— Analyse et dissection des commandes WirelessHART (via Scapy).
— Journalisation et export des captures.

Un dissecteur Scapy WirelessHART dédié a été implémenté dans le cadre de cette étude.

Mécanisme de synchronisation La synchronisation du sniffer s'effectue en plusieurs phases :

Phase d'initialisation Le dispositif radio écoute successivement chaque canal $[11; 25]$ pendant une seconde jusqu'à la réception de trois paquets d'annonce distincts provenant potentiellement de différentes sources. Ces annonces permettent de :
— Extraire la *channel map*.
— Vérifier la durée effective du slot à partir des temps de réception et des numéros d'ASN associés.

À la réception du troisième paquet d'annonce, un timer matériel est démarré. Le début du slot suivant est calculé selon :

$$t_{\text{slot\_start}} = t_{\text{adv\_rx}} - \frac{\text{Len}_{\text{pkt}}}{R_{\text{data}}} - T_{s\text{TxOffset}} + T_{\text{slot}}$$

où $t_{\text{adv\_rx}}$ est l'instant de réception du paquet, Len_{pkt} sa longueur, R_{data} le débit de la couche physique, $T_{s\text{TxOffset}}$ le délai de transmission et T_{slot} la durée d'un slot (10 ms).

Maintien de la synchronisation À chaque début de slot, le firmware calcule le canal actif en utilisant l'algorithme de saut de fréquence du WirelessHART, introduit dans la section 2.2. Sans réception de paquet durant un slot, le prochain slot commencera après T_{slot}, dans le cas contraire, le firmware ajuste l'instant prévu du prochain slot selon :

$$t_{\text{next\_slot}} = t_{\text{rx}} + T_{\text{slot}} - \frac{\text{Len}_{\text{pkt}}}{R_{\text{data}}} - T_{s\text{TxOffset}}$$

où t_{rx} est l'instant de réception du paquet de non acquittement.

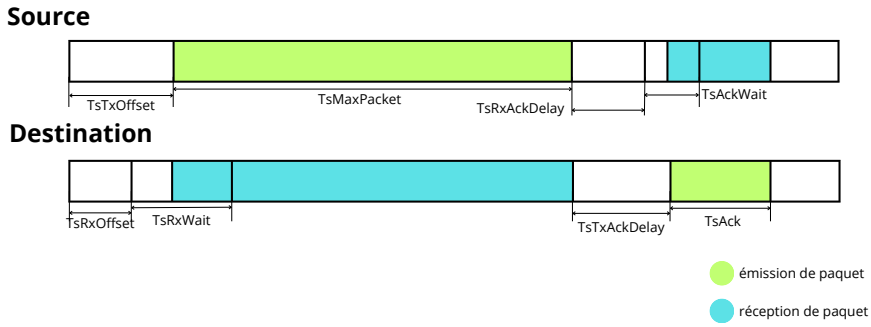

Fig. 5. Découpage temporel d'un slot WirelessHART. [11]

Déchiffrement et analyse des communications Une fois capturés par le firmware, les paquets sont transmis à *Whad-client* qui effectue leur déchiffrement en fonction du champ *Security Type* présent dans l'en-tête de la couche réseau.[4]

Le déchiffrement complet du trafic nécessite :

— La **clé d'association** ($\mathcal{K}_{\text{join}}$) : indispensable pour déchiffrer les échanges d'association et récupérer la clé du réseau et les clés de session distribuées au mote rejoignant le réseau.

— La **clé de réseau** ($\mathcal{K}_{\text{network}}$) : nécessaire pour vérifier l'intégrité au niveau de la DLL (DL-MIC) et pour le calcul d'intégrité lors d'une injection de trafic.

— Les **clés de session** ($\mathcal{K}_{\text{session\_unicast}}$, $\mathcal{K}_{\text{session\_broadcast}}$) : requises pour déchiffrer les données applicatives de la couche transport.

Méthodologie d'utilisation Deux scénarios d'utilisation principaux se distinguent selon l'instant de déploiement du sniffer.

Observation depuis la création du réseau Lorsque le sniffer est déployé avant ou pendant l'association des motes, il peut observer l'intégralité des échanges de clés. Dans ce cas :

1. Le sniffer se synchronise sur les messages d'annonce.
2. Il capture les handshakes d'association et extrait les clés distribuées.

[4] Le champ *Security Type* [11] peut prendre les valeurs suivantes : 0 pour une clé de session, 1 pour une clé d'association, 2 réservé pour usage futur.

3. Il construit progressivement la table des liens de communication de chaque mote.

Ce scénario offre une visibilité maximale et permet de suivre l'évolution dynamique du réseau.

Ajout après association des motes Dans ce cas, plusieurs limitations apparaissent :
— Les clés de session des motes déjà associés ne sont pas connues.
— La table des liens existants doit être inférée par observation passive.
— Seuls les nouveaux motes rejoignant le réseau verront leurs clés capturées.

Les communications chiffrées avec des clés inconnues ne peuvent être déchiffrées, et seules les méta-données (adresses, timing, canaux) restent exploitables pour l'analyse de la topologie.

Exploration des liens non observés Afin de palier aux limitations du scénario où le sniffer est déployé après l'établissement du réseau, nous avons conçu un mécanisme d'exploration automatique permettant d'inférer l'existence de liens actifs par test systématique des offsets possibles.

Principe de fonctionnement À chaque slot, le firmware du nRF52840 détermine si le slot courant correspond à un lien connu. Si aucune correspondance n'est trouvée dans la table des liens, le dispositif considère qu'il peut s'agir d'un lien encore non identifié et calcule un offset de test selon la stratégie suivante :

$$\text{offset} = \left\lfloor \frac{\text{ASN}}{\max\{|\text{superframe}|\}} \right\rfloor \bmod N_{\text{actifs}}$$

où $\max\{|\text{superframe}|\}$ correspond à la longueur maximale parmi les *superframes* connues et N_{actifs} désigne le nombre de canaux actifs dans la *channel map*.

Cette heuristique garantit que l'ensemble des offsets possibles est testé cycliquement sur un nombre limité de slots. Le firmware bascule alors sur le canal correspondant et tente de capturer un éventuel paquet. Si une réception a lieu sur ce lien inconnu, le dispositif envoie une notification `DiscoveredCommunication` à *Whad-client*, contenant l'ASN de capture et le paquet reçu.

Validation des liens explorés À la réception d'une notification de type `DiscoveredCommunication`, *Whad-client* analyse le paquet pour en extraire :

— Les adresses source et destination.
— Le type de lien (transmission, réception, partagé).
— Les options de communication.
— L'offset effectif utilisé.

Un lien candidat est alors créé et stocké dans une structure temporaire. L'hôte teste ensuite plusieurs hypothèses de longueur de *superframe* (en ordre décroissant de longueur) et vérifie si d'autres communications découvertes correspondent au même pattern :

— Même offset calculé modulo la longueur de superframe testée.
— Même paire (source, destination).
— Périodicité cohérente avec les ASN observés.

Chaque correspondance incrémente un score de confiance associé au lien candidat. Un thread dédié s'exécutant en parallèle surveille en permanence ces scores et valide automatiquement les liens ayant dépassé un seuil prédéfini (typiquement 3 à 5 observations cohérentes). Une fois validé, le lien est intégré dans la table d'ordonnancement et *ButteRFly* est notifié pour qu'il suive activement ce lien lors des prochains slots correspondants.

Limitations de l'exploration Ce mécanisme présente plusieurs contraintes :

— **Temps de découverte** : La validation d'un lien nécessite plusieurs occurrences périodiques, ce qui peut prendre plusieurs cycles de *superframe*.
— **Liens à faible activité** : Les liens utilisés sporadiquement (par exemple, pour des événements exceptionnels) sont difficilement détectables par cette approche statistique.
— **Charge de traitement** : Le calcul et le test systématique d'offsets sur chaque slot libre induisent une charge de traitement non négligeable, limitant potentiellement la mise à l'échelle dans des réseaux très denses.

Capacités d'injection Au-delà de la capture passive, l'outil intègre des fonctionnalités d'injection de paquets permettant la réalisation d'attaques actives. *Whad-client* construit les paquets à injecter en utilisant Scapy, puis les transmet à *ButteRFly* via le protocole WHAD. Le chiffrement des payloads est effectué côté client avec les clés de session récupérées lors de l'observation des associations. Ces capacités ont été utilisées pour évaluer les attaques présentées en Section 3.

Fig. 6. Matériel (*gateway* et nœuds) WirelessHART utilisé lors de nos expériences

4.3 Résultats

Durant toutes nos expérimentations, nous avons utilisé du matériel de la marque Analog Devices/Dust de la gamme SmartMesh, montré en figure 6, afin de créer un réseau WirelessHART cible dans lequel nous avons mené nos attaques et l'évaluation de nos outils.

Evaluation du sniffer Afin d'évaluer les performances du sniffer, nous avons limité la channel map à quatre canaux : {11, 23, 24, 25}, les autres étant placés en liste noire.[5] D'une part, quatre sniffers mono-canal ont été déployés, chacun dédié à un canal et enregistrant les paquets capturés avec l'identifiant du canal correspondant. Ces quatre sniffers sont utilisés comme groupe témoin pour obtenir une mesure de référence. D'autre part, un dernier sniffer à saut de fréquence, développé durant nos travaux, a été utilisé pour enregistrer les trames observées en parallèle des quatre autres sniffers.

Les résultats montrent que l'intégralité du processus d'association a été capturée par le sniffer à saut de fréquence. Deux expérimentations complémentaires ont été réalisées afin d'évaluer les performances du sniffer.

La première expérimentation (A) excluant les paquets d'annonce a été reproduite sur une communication continue de 14 heures entre deux motes et la *gateway*, durant la nuit, dans un environnement de bureau.

[5] La channel map est contrôlée à l'aide de l'API *Explorer* du SDK SmartMesh.

Tableau 3. Nombre de paquets capturés par le sniffer. (Légende : Snif_4 désigne les 4 sniffers mono-canal et Snif_{FH} désigne le sniffer à saut de fréquence.)

Exp.	Paquets	Données	Keep-alive	Ack	Non accurate	Total
A	Snif_4	4514	3245	6840	2	14601
	Snif_{FH}	4511	3243	6834	5	14593
	Taux de perte	0.07%	0.09%	0.09%	N/A	0.05%
B	Snif_4	12929	59	12354	158	25500
	Snif_{FH}	11326	44	10765	153	22288
	Taux de perte	12.4%	25.42%	12.86%	N/A	12.6%

Le tableau 3 présente le nombre de paquets capturés par chaque type de sniffer. Le taux de perte observé pour le sniffer à saut de fréquence reste inférieur à 0,1 % pour l'ensemble des catégories de paquets, confirmant sa fiabilité pour l'analyse du trafic WirelessHART.

La seconde expérimentation (B) a été mise en place dans un environnement domestique sur une communication continue entre cinq motes et la *gateway*. L'expérience a été réalisée pendant une durée de 45mn, incluant la phase d'association, en générant du trafic intensivement et en continu par l'intermédiaire de la commande `Ping` de la *gateway* à destination de l'ensemble des nœuds. On constate une dégradation des résultats à 12,6 % de pertes totales, attribuables à la densité du trafic généré combinée à un environnement domestique plus bruité en 2,4 GHz (Wi-Fi, Bluetooth).

Évaluation de l'attaque de de-authentification massive Nous avons été en mesure d'évaluer l'attaque décrite dans la section 3.2, en implémentant et en injectant une requête de commande `Suspend` vers les voisins du *network manager*. La requête injectée usurpe l'identité de ce dernier au niveau de la couche réseau et celle de la *gateway* au niveau de la DLL. Nous avons mené deux expériences, afin d'évaluer respectivement [6] :

— un déni de service massif de l'ensemble des motes du réseau en spécifiant une durée de suspension longue (1000000 slots, soit 2.77h),

— la récupération des clés de session d'un mote en forçant une suspension courte (1000 slots, soit 10s) et en exploitant le mécanisme de ré-association automatique d'un mote au réseau.

[6] Les traces détaillées des deux expérimentations sont accessibles en ligne :
`https://homepages.laas.fr/rcayre/log_deauth_wihart/mass_deauth_attack_dos.log`
`https://homepages.laas.fr/rcayre/log_deauth_wihart/deauth_attack_steal_key.log`

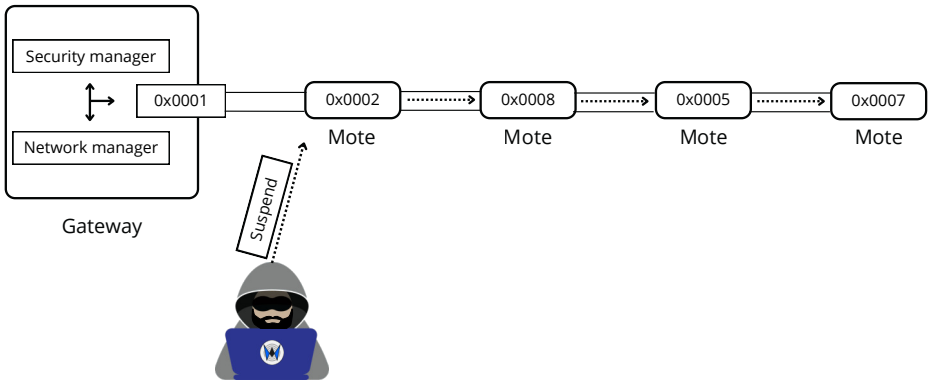

Fig. 7. Topologie de l'expérience 1 - Déni de service

Expérience 1 : déni de service La première expérience menée avait pour objectif d'évaluer la faisabilité d'un déni de service visant l'ensemble des motes du réseau. Dans ce cadre, nous avons notamment déployé un réseau de capteurs industriels Dust composé d'une *gateway* et de 4 motes répartis selon une topologie linéaire, illustrée en figure 7, ainsi qu'un nœud malveillant (dongle nRF52840 équipé du firmware ButteRFly) déployé depuis l'initialisation du réseau à portée radio, implémentant l'attaquant. Chaque mote a été placé dans un bureau différent, éloignés d'une quinzaine de mètres environ.

Dans cette configuration, nous avons expérimenté l'envoi de la commande Suspend depuis l'attaquant en mode *broadcast*, avec une durée de suspension de longue durée (1000000 slots) :

```
|###[ Wireless Hart Command Request header ]###
|  command_number= 0x3cc
|  len       = 10
|###[ Suspend Devices Request ]###
|     asn_suspend= 103493
|     asn_resume= 1103493
```

Nous avons pu observer par l'intermédiaire du dongle nRF52840 que tous les motes recevant la requête de la commande ont relayé à leur tour le message vers leurs voisins sur le prochain lien *broadcast*, l'ASN courant restant inférieur à l'ASN de suspension indiqué. Le diagramme de séquence 8 détaille les paquets transmis par les différents nœuds observés lors de l'attaque.

Comme attendu, ce mécanisme a entraîné la suspension de l'ensemble des motes. Leur remise en marche nécessite une intervention humaine afin de les redémarrer manuellement, l'ASN de reprise spécifié indiquant une suspension de plusieurs heures.

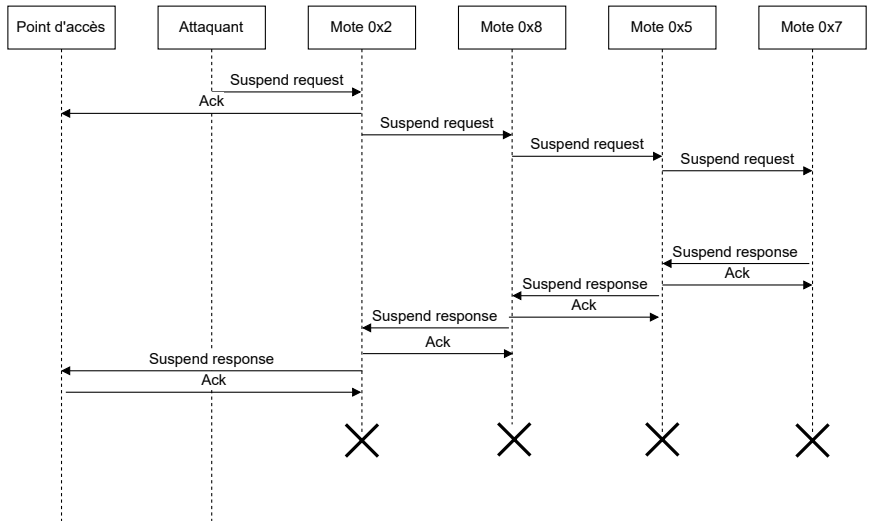

Fig. 8. Diagramme de séquence des paquets transmis.

Nous avons pu confirmer depuis la console de la *gateway* que les motes ne répondaient plus aux `Ping`, confirmant la réussite d'un déni de service massif du réseau :

```
> ping 2
> ping 5
> ping 7
> ping 8
> [07:35:43] Ping mote 2: reply #1: timed out
[07:35:43] Ping mote 2: sent 1, rcvd 0, 100% lost.
[07:37:07] Ping mote 5: reply #1: timed out
[07:37:07] Ping mote 5: sent 1, rcvd 0, 100% lost.
[07:37:25] Ping mote 8: reply #1: timed out
[07:37:25] Ping mote 8: sent 1, rcvd 0, 100% lost.
[07:37:43] Ping mote 7: reply #1: timed out
[07:37:43] Ping mote 7: sent 1, rcvd 0, 100% lost.
Mote #2 changed state to Lost
Mote #8 changed state to Lost
Mote #5 changed state to Lost
Mote #7 changed state to Lost
```

Expérience 2 : ré-association forcée Les capteurs industriels Dust utilisés lors de ces expérimentations implémentent un mécanisme de ré-association automatique au réseau suite à une suspension. En exploitant ce mécanisme et en le combinant à l'injection de commande `Suspend`, il nous a été possible de forcer une ré-association en injectant une requête de la commande indiquant une durée de suspension courte (1000 slots), permettant la récupération des clés de session d'un mote.

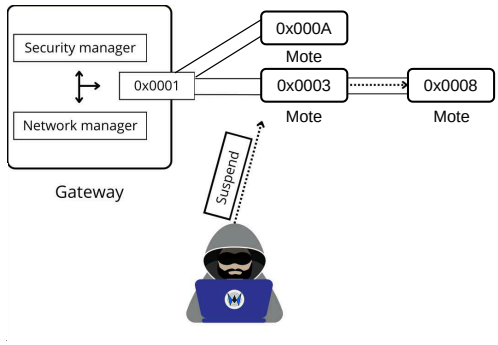

Fig. 9. Topologie de l'expérience 2 - Réassociation forcée

Pour cette expérience, nous avons déployé un réseau de capteurs industriels Dust composé d'une *gateway* et de 3 motes répartis selon la topologie illustrée en figure 9, ainsi qu'un attaquant pouvant sniffer le trafic et injecter des paquets (dongle nRF52840). Deux motes (3 et 10) ont été placés en triangle à dix mètres de la *gateway*, et un mote (8) a été placé à 15 mètres de la *gateway* et 5 mètres du mote 3, afin d'assurer la présence d'un voisin. Nous avons pu confirmer la présence d'une route entre 3 et 8 en observant le nombre de hops (2) lors d'un ping réalisé depuis la *gateway* à destination du mote 8 :

```
> ping 8
> [13:00:25] Ping mote 8: reply #1: 3.725s 2 hops [19.9C 2.115V]
[13:00:25] Ping mote 8: sent 1, rcvd 1, 0% lost. Ave.roundtrip: 3.725s hops: 2

> ping 10
> [13:00:33] Ping mote 10: reply #1: 3.852s 1 hops [19.5C 2.197V]
[13:00:33] Ping mote 10: sent 1, rcvd 1, 0% lost. Ave.roundtrip: 3.852s hops: 1

> ping 3
> [13:00:38] Ping mote 3: reply #1: 2.540s 1 hops [19.5C 2.373V]
[13:00:38] Ping mote 3: sent 1, rcvd 1, 0% lost. Ave.roundtrip: 2.540s hops: 1
```

Nous avons réalisé successivement :

— l'association des nœuds 3, 8 et 10,
— l'extinction du nœud 3,
— l'activation du sniffer,
— une ré-association du nœud 3.

L'attaquant a ainsi été en mesure d'observer uniquement l'association du nœud 3 et de récupérer ses clés de sessions *unicast* et les clés de *broadcast*, constituant les conditions initiales de l'expérimentation.

```
    <WirelessHartDecryptor>
  Join Key       : b'ABCDABCDABCDABCD'
  Network Key    : b'\x117\xac\n<S_\xce\xc3\xd0[X\x03JK\x86'
  Unicast Sessions Keys:
    id1=3, id2=63872, nonce=13
        -> b'\xb9\xd6\x04\x85\x81\xa0\xcc\x85r\xae\xa7\xd5\xf3\x0e$\x8c'
    id1=63873, id2=3, nonce=12
        -> b'?\x19\x19\xf0#\xb33\x1f\xb5Y\xf7\xfc\xc8\xb6\xe0\x16'
  Broadcast Sessions Keys:
    id1=65535, id2=63872, nonce=12
        -> b'@\x91\x97\x81\x05G\xe1\xa9K\xe2\x01\xb15\x95[\x84'
    id1=65535, id2=63873, nonce=1
        -> b'\xf7\xe5C\xd6\xdf\x9c@\xb5c\xca\xef5\xebY\xca\xc6'
```

Nous avons ensuite réalisé l'injection d'une commande *Suspend*, transmise en usurpant l'adresse 0x1 (*gateway*) vers l'adresse 0x3 (mote 3) au niveau DLL, et l'adresse du *network manager* (0xF980) comme adresse source et celle de *broadcast* (0xFFFF) en destination au niveau réseau, tout en spécifiant une durée d'ASN courte (1000 slots). Le paquet a été chiffré et authentifié avec les clés $\mathcal{K}_{\text{session\_broadcast}}$ et $\mathcal{K}_{\text{network}}$, récupérées lors de l'observation par l'attaquant de la phase d'association du mote 3.

```
###[ 802.15.4 Data ]###
     dest_panid= 0x4cd
     dest_addr = 0x3
     src_addr  = 0x1
###[ Wireless Hart Data Link header ]###
        reserved  = 0
        priority  = command
        network_key_use= yes
        pdu_type  = data
        mic       = 0x0
###[ Wireless Hart Network Layer header ]###
           [...]
           graph_id  = 0x0
           nwk_dest_addr= 0xffff
           nwk_src_addr= 0xf980
###[ Wireless Hart Network Security sub-layer header ]###
              reserved  = 0
              security_types= session_keyed
              counter   = 0xd
              nwk_mic   = 0x0
###[ Wireless Hart Transport Layer header ]###
                 [...]
                 \commands  \
                  |###[ Wireless Hart Command Request header ]###
                  |  command_number= 0x3cc
                  |  len       = 10
                  |###[ Suspend Devices Request ]###
                  |     asn_suspend= 311759
                  |     asn_resume= 312759
```

Suite à cette injection, nous avons pu observer la propagation de la *Suspend Request* de 0x3 vers 0x8, de 0x8 vers 0x1 et de 0x3 vers 0x1,

indiquant la diffusion par 0x3 du paquet vers son voisin 0x8. Sur la console de la *gateway*, suite à la prise en compte de la suspension, on observe une perte de connectivité pour les motes 3 et 8, immédiatement suivie par une ré-association des deux nœuds :

```
> Mote #8 changed state to Lost
Mote #8 changed state to Negot1
Mote #8 changed state to Negot2
Mote #8 changed state to Conn
Mote #3 changed state to Lost
Mote #3 changed state to Negot1
Mote #3 changed state to Negot2
Mote #3 changed state to Conn
Mote #8 changed state to Oper
Mote #3 changed state to Oper
```

Ces phases d'association sont directement observées par l'attaquant sur le sniffer, permettant la récupération :

— des nouvelles clés de session *unicast* du nœud 3

— des clés de session *unicast* associées au nœud 8.

```
<WirelessHartDecryptor>
  Join Key       : b'ABCDABCDABCDABCD'
  Network Key    : b'\x117\xac\n<S_\xce\xc3\xd0[X\x03JK\x86'
  Unicast Sessions Keys:
    id1=3, id2=63872, nonce=6
        -> b'.\xf5J%\xe5\x1b\x11\x1f\x00\x90\xdd\x84\xee\\\x9f\xc0'
    id1=63873, id2=3, nonce=13
        -> b"\xbfa3*Y\x04\xaeH\xc9\x92\xfci'\xa5=\x97"
    id1=8, id2=63872, nonce=7
        -> b'\xe0\x01Z\xeb\xb3\xfes\xc8\xc9\xfaq\xf1\xaf\x10K\x1d'
    id1=63873, id2=8, nonce=1
        -> b'\xfeK\xe6K\xf8&a>5\xb6\xea\xdeK]\xc6\xe5'
  Broadcast Sessions Keys:
    id1=65535, id2=63872, nonce=13
        -> b'@\x91\x97\x81\x05G\xe1\xa9K\xe2\x01\xb15\x95[\x84'
    id1=65535, id2=63873, nonce=1
        -> b'\xf7\xe5C\xd6\xdf\x9c@\xb5c\xca\xef5\xebY\xca\xc6'
```

L'attaquant est alors en mesure de déchiffrer l'intégralité du trafic associé au nœud 8 et d'usurper son identité, compromettant l'intégrité et la confidentialité du réseau.

Attaque de désynchronisation temporelle Nous avons réalisé une évaluation de l'attaque de désynchronisation présentée en Section 3.2. En injectant des requêtes `Ping` usurpant l'identité d'une source de temps légitime avec un décalage temporel contrôlé, nous avons été en mesure d'observer différentes réactions du réseau au paquet injecté, incluant une désynchronisation complète du mote visé.

Environnement expérimental Pour réaliser cette analyse, nous avons déployé un réseau cible composé d'un mote et d'une *gateway* Dust, au sein d'un environnement contrôlé (déploiement au sein d'une cage de Faraday). Nous avons également déployé au sein de l'environnement un dongle *nRF52840* embarquant le firmware *ButterFly*, simulant le rôle d'un attaquant. Lors de nos expérimentations, nous avons réalisé les opérations suivantes :

- **Association du mote légitime (8) à la *gateway* (1)**, observé dans son intégralité par le sniffer.
- **Génération de trafic légitime**, en déclenchant dix procédures de ping depuis la console de la *gateway* à destination du mote (8).
- **Injection d'une Ping Request intégrant un décalage temporel contrôlé**, à destination du mote légitime 8, en usurpant l'adresse de la *gateway* (1) comme source au niveau DLL, et en usurpant l'adresse du *network manager* (0xf980) au niveau réseau.

Le décalage temporel a été contrôlé en intégrant au sein de la primitive d'émission pour un slot dans le firmware du sniffer un retard temporel en micro-secondes, paramétrable depuis le client. En parallèle de chaque tentative d'injection, nous avons loggé l'intégralité du trafic observé par le sniffer, ainsi que les logs correspondants générés par le mote et par la *gateway*.

Campagnes d'injection réalisées Nous avons réalisé plusieurs campagnes d'injection différentes, visant à caractériser les différentes réactions du mote en fonction du décalage temporel lors de l'injection. Nous présentons ici deux campagnes d'injection présentant des résultats représentatifs :

- **Expérience 1** : Injection unique avec 1000us de décalage
- **Expérience 2** : Injections multiples, en incrémentant le décalage progressivement à partir de 2000us

Les résultats obtenus lors de ces expérimentations sont représentés graphiquement respectivement en Figure 10 et en Figure 11. La représentation graphique proposée permet une visualisation de l'ensemble de la communication en fonction du temps. Le graphique du haut correspond au décalage observé lors de la réception du paquet par rapport à l'instant de transmission idéal d'un paquet de donnée du point de vue du *Network Manager* (référentiel glissant). Les deux graphiques du bas représentent les valeurs d'ajustements temporels (*time adjustements*) extraites des paquets d'acquittements observés, indiqués en μs.

En effet, lors de la réception d'un paquet nécessitant un acquittement, le récepteur calcule une valeur de correction Δt, correspondant à la diffé-

rence entre l'instant de réception observé (t_{reel}) et l'instant de réception théorique idéal (t_{ideal}). Par conséquent :

- Une valeur de correction positive indique un paquet reçu **avant** l'instant idéal estimé, indiquant un **retard** de l'horloge du récepteur.
- Une valeur de correction négative indique un paquet reçu **après** l'instant idéal estimé, indiquant une **avance** de l'horloge du récepteur.

Cette valeur de correction est directement reportée à l'émetteur par l'intermédiaire d'un champ spécifique intégré au sein de l'acquittement :

```
<WirelessHart_DataLink_Acknowledgement [...] time_adjustment=64 |>>>>
```

Stratégie de synchronisation temporelle Lors d'une communication, l'un des équipements voisins sert de référence temporelle (ou *Time Source*) à l'autre, guidant la stratégie de synchronisation temporelle utilisée. Dans le cas des échanges observés lors de ces expérimentations entre la *gateway* et le *mote*, la source temporelle sélectionnée est la *gateway*. Ainsi, à chaque paquet de données reçu depuis la *gateway*, ou à chaque valeur de correction extraite depuis un acquittement transmis par la *gateway*, le mote va appliquer la correction qu'il a estimé (et potentiellement reporté via un acquittement) en soustrayant la valeur de $-\Delta t$ lors du calcul des bornes temporelles du prochain slot, afin de compenser la dérive d'horloge ainsi estimée. La figure 12 illustre le fonctionnement normal du mécanisme de synchronisation temporel, appliqué à la correction d'une dérive t_{ini} entre le Mote et la Gateway.

Analyse des résultats Les résultats obtenus lors de nos expérimentations mettent en évidence trois possibilités lors de l'injection d'un paquet de Ping Request décalé temporellement, émis à destination du mote en usurpant l'identité de la *gateway*.

Le premier cas observé (Cas A), illustré en Figure 13 correspond au cas le plus critique, menant à une désynchronisation complète des deux équipements, illustré par l'expérience 1.

Il correspond à une injection de Ping Request à la limite de la borne basse de la fenêtre d'écoute du mote : en appliquant un décalage temporel de 1000 μs lors de l'injection, nous avons pu observer un acquittement du mote indiquant une valeur d'ajustement temporel égale à 1073 μs :

```
[packet]  [timestamp=1911173480, channel=20]  | 4188b8cd040100080038000 4312ae2253234da
<Dot15d4FCS fcs=0xda34 |
    <Dot15d4Data  dest_panid=0x4cd dest_addr=0x1 src_addr=0x8 |
    <WirelessHart_DataLink_Hdr priority=command network_key_use=yes pdu_type=acknowledgment mic=0x2ae22532 |
    <WirelessHart_DataLink_Acknowledgement  response_code=success time_adjustment=1073 |>>>>
```

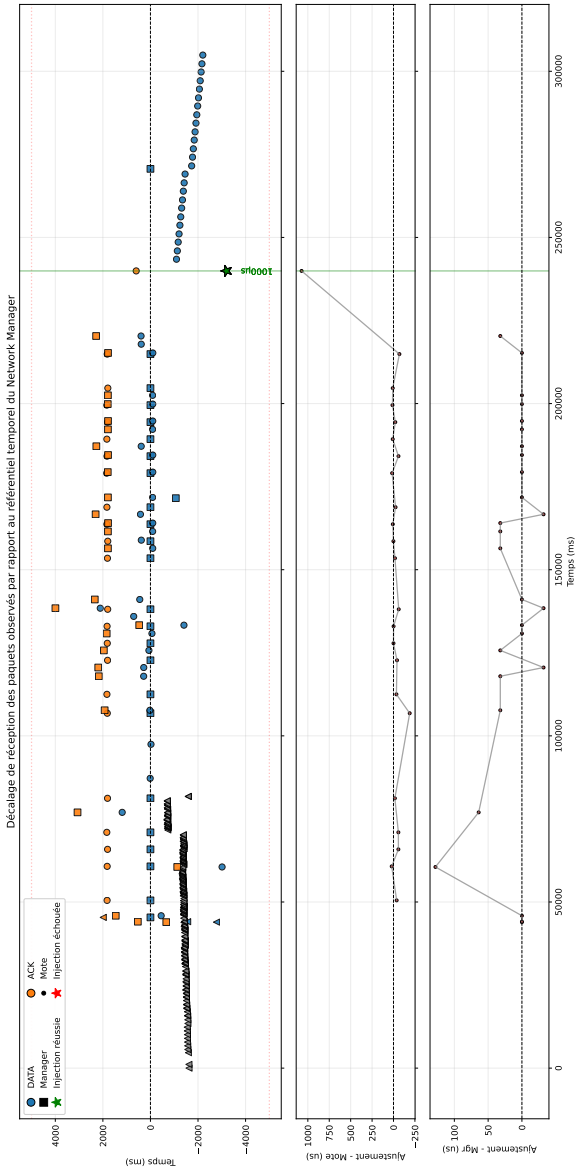

Fig. 10. Expérience 1 : résultats expérimentaux de l'injection unique (decalage faible)

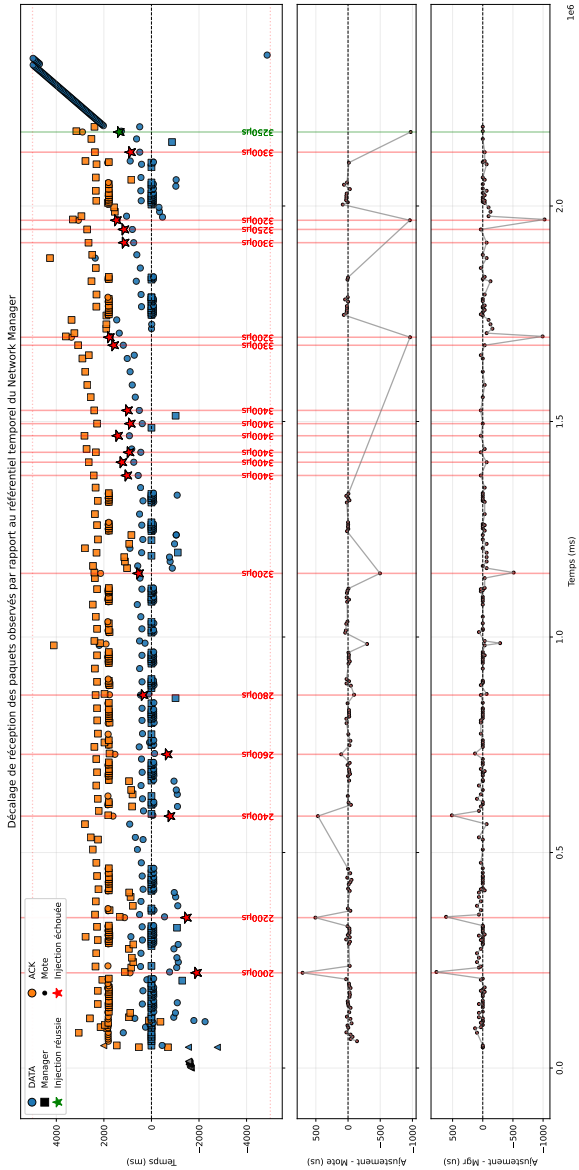

Fig. 11. Expérience 2 : résultats expérimentaux des injections multiples (decalages croissants)

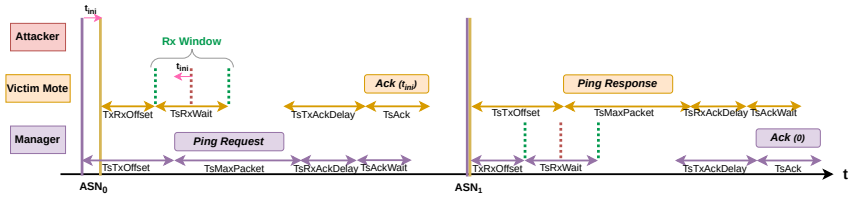

Fig. 12. Correction temporelle appliquée par le mote en condition normale (sans injection)

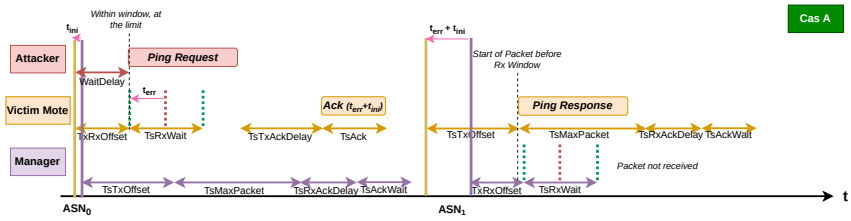

Fig. 13. Succès de l'attaque de désynchronisation temporelle

L'application de cette correction amène le mote à soustraire cette valeur lors du calcul des bornes temporelles du prochain slot, avançant son horloge de 1073 μs. Lors de la transmission par le mote de la *Ping Response*, le paquet est donc transmis avant l'ouverture de la fenêtre de réception de la *gateway* légitime, et ne sera jamais acquitté. Suite à cette desynchronisation durable, on observe des tentatives de retransmissions répétées de la *Ping Response* jusqu'à la déconnexion complète du mote. Ces transmissions non acquittées sont visibles dans la sorties du Sniffer :

```
[packet]  [ timestamp=1914691786, channel=22]
<Dot15d4FCS  fcs=0x6576 |
    <Dot15d4Data  dest_panid=0x4cd dest_addr=0x1 src_addr=0x8 |
    <WirelessHart_DataLink_Hdr priority=command network_key_use=yes pdu_type=data mic=0xd93d43f0 |
    <WirelessHart_Network_Hdr  asn_snippet=0x1ac8 nwk_dest_addr=0xf980 nwk_src_addr=0x8 |
    <WirelessHart_Network_Security_SubLayer_Hdr security_types=decrypted counter=0x14 nwk_mic=0x164f6de7 |
    <WirelessHart_Transport_Layer_Hdr  commands=[
        <WirelessHart_Vendor_Specific_Dust_Networks_Ping_Response
          status=0 expanded_device_type=0xe0a2 hops=1 temperature=315 voltage=3477 |>>
    ] |>>>>>>


[packet]  [ timestamp=1917201750, channel=21]
<Dot15d4FCS  fcs=0x6576 |
    <Dot15d4Data  dest_panid=0x4cd dest_addr=0x1 src_addr=0x8 |
    <WirelessHart_DataLink_Hdr priority=command network_key_use=yes pdu_type=data mic=0xf71f586 |
    <WirelessHart_Network_Hdr  asn_snippet=0x1ac8 nwk_dest_addr=0xf980 nwk_src_addr=0x8 |
    <WirelessHart_Network_Security_SubLayer_Hdr security_types=decrypted counter=0x14 nwk_mic=0x164f6de7 |
    <WirelessHart_Transport_Layer_Hdr  commands=[
        <WirelessHart_Vendor_Specific_Dust_Networks_Ping_Response
          status=0 expanded_device_type=0xe0a2 hops=1 temperature=315 voltage=3477 |>>
    ] |>>>>>>

    [...]
```

et dans les logs du mote :

```
> 234826 : MAC R: a=6840 t=7 ch=9 s=1 rc=0 rs=-25 ad=21464 qf=0 ql=0 qs=0 mic=24393f2b temp=31
234831 : RX ttl=126 asn=7347 gr=0 dst=8 src=63872 r=GR sec=s nc=0 mic=00000000 tr=Ulbl:Ucast:req:31
    cmdid=FC04 len=4: E0A20001;

234980 : TX ttl=249 asn=6856 gr=0 dst=63872 src=8 r=GR sec=s nc=0 mic=00000000 tr=Ulbl:Ucast:rsp:15
    cmdid=FC05 len=9: 00E0A20001013B0D95;

238346 : MAC T: a=7192 t=7 ch=11 d=1 rc=1 ad=0 po=-6951 pe=394 qf=0 ql=1 qs=0 mic=164f6de7 temp=31
240411 : TX ttl=249 asn=7399 gr=0 dst=63873 src=8 r=GR sec=s nc=0 mic=00000000 tr=Ulbl:Ucast:req:16
    cmdid=0000 len=5: 00FF010500; cmdid=0000 len=0: ; cmdid=69DB len=199: 8B0003066A000075300110OC630000...;

240856 : MAC T: a=7443 t=7 ch=10 d=1 rc=1 ad=0 po=-6951 pe=394 qf=0 ql=2 qs=0 mic=164f6de7 temp=31
243496 : MAC T: a=7707 t=7 ch=4 d=1 rc=1 ad=0 po=-6951 pe=394 qf=0 ql=2 qs=0 mic=164f6de7 temp=31
245956 : MAC T: a=7953 t=7 ch=14 d=1 rc=1 ad=0 po=-6951 pe=394 qf=0 ql=2 qs=0 mic=164f6de7 temp=31
248586 : MAC T: a=8216 t=7 ch=13 d=1 rc=1 ad=0 po=-6951 pe=394 qf=0 ql=2 qs=0 mic=164f6de7 temp=31
251096 : MAC T: a=8467 t=7 ch=12 d=1 rc=1 ad=0 po=-6951 pe=394 qf=0 ql=2 qs=0 mic=164f6de7 temp=31
253736 : MAC T: a=8731 t=7 ch=6 d=1 rc=1 ad=0 po=-6951 pe=394 qf=0 ql=2 qs=0 mic=164f6de7 temp=31
[...]
420523 : NET TX: mac nack
[...]
Disconnecting
```

Nous avons pu observer une situation similaire en injectant à proximité de la borne haute de la fenêtre temporelle (injection à 3250 μs), comme illustré sur la dernière injection de l'expérience 2. Il est donc à noter que le succès de la désynchronisation dépend de l'injection à proximité de l'une des limites de la fenêtre du mote (minimale ou maximale), résultant en un décalage suffisant de l'horloge du mote pour provoquer l'émission des paquets suivants hors de la fenêtre de réception de la *gateway*.

Les autres cas observés, illustrés au sein de l'expérience 2, correspondent respectivement :

— à une injection hors de la fenêtre de réception du mote, le paquet injecté étant ignoré par le mote (Cas B).

— à une injection dans la fenêtre de réception du mote, provoquant un décalage non suffisant pour que les émissions ultérieures du mote sortent de la fenêtre (Cas C).

Ces deux cas sont illustrés en Figure 14.

Dans l'expérience 2, nous incrémentons progressivement le décalage temporel à partir de 2000 μs, par pas de 200, en réalisant des injections dans les bornes de la fenêtre de réception du mote, résultant dans le Cas C. Les injections correspondantes mènent à la réception d'un acquittement transmis par le mote indiquant un ajustement temporel. Sur un slot ultérieur, le mote transmet à la *gateway* légitime une Ping Response, correctement acquitté par la *gateway*, indiquant un ajustement temporel du même ordre de grandeur :

```
[packet]  [timestamp=529183956, channel=21 ]
<Dot15d4Data  dest_panid=0x4cd dest_addr=0x1 src_addr=0x8 |
[...] |<WirelessHart_DataLink_Acknowledgement  response_code=success time_adjustment=706 |>>>>

[packet]  [timestamp=531082293, channel=19]
<Dot15d4Data  dest_panid=0x4cd dest_addr=0x1 src_addr=0x8 |
<WirelessHart_DataLink_Hdr  [...] pdu_type=data mic=0x8d802c96 |
<WirelessHart_Network_Hdr
nwk_dest_addr=0xf980 nwk_src_addr=0x8 |
[...] <WirelessHart_Vendor_Specific_Dust_Networks_Ping_Response [...]>>] |>>>>>>
```

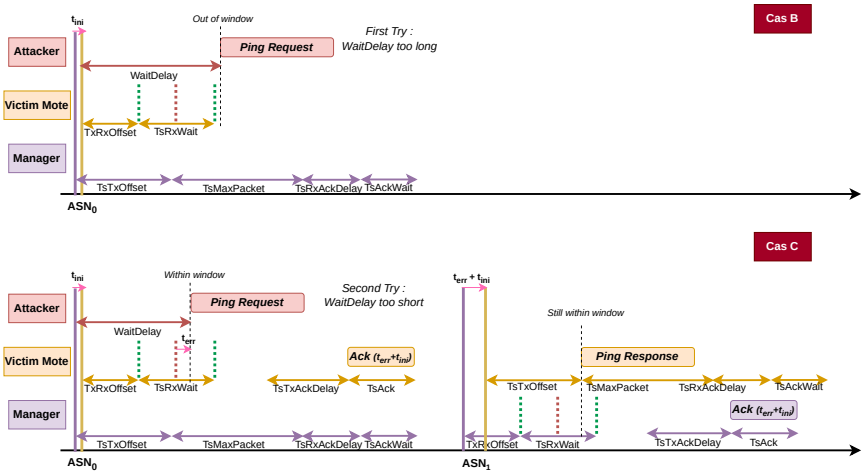

Fig. 14. Échecs de l'attaque de désynchronisation temporelle

```
[packet]  [timestamp=531084179, channel=19]  |
<Dot15d4Data  dest_panid=0x4cd dest_addr=0x8 src_addr=0x1 |
[...] |<WirelessHart_DataLink_Acknowledgement  response_code=success time_adjustment=768 |>>>>
```

Suite à cet échange, on observe une resynchronisation rapide du mote et de la *gateway*.

A partir d'un décalage temporel de 3400 μs, nous n'observons plus de paquet d'acquittements transmis par le mote suite à nos injections, marquant la sortie de sa propre fenêtre de réception. Cette situation correspond donc au Cas B. Nous en déduisons que le décalage temporel correspondant à la limite de la fenêtre se situe empiriquement entre 3200 μs et 3400 μs, et ajustons progressivement le décalage pour reproduire une injection en limite de fenêtre provoquant une désynchronisation réussie similaire à l'expérience 1, correspondant à un décalage de 3250 μs.

Les valeurs obtenues expérimentalement sont cohérentes avec la valeur théorique définissant la taille de la fenêtre de réception dans la spécification, nommée *TsRxWait* et devant durer 2200 μs.

5 Contre-mesures

Nous avons présenté différentes attaques qui compromettent la confidentialité, l'intégrité et l'authentification des communications dans un réseau WirelessHART. Dans cette section, nous proposons différentes

suggestions de contre-mesures permettant de supprimer ou de limiter les vecteurs d'attaques utilisés.

5.1 Utilisation d'une $\mathcal{K}_{\text{join}}$ différente pour chaque nœud

Tout d'abord, nous insistons sur l'utilisation de clés d'association $\mathcal{K}_{\text{join}}$ différentes pour chaque nœud du réseau. Dans la Section 4.3, nous avons montré que dans le cas où le réseau WirelessHART est configuré pour utiliser une clé d'association $\mathcal{K}_{\text{join}}$ commune à tous les nœuds, un attaquant pouvait, via une attaque active et passive, connaître les clés de session *unicast* $\mathcal{K}_{\text{session\_unicast}}$ des autres nœuds du réseau. Or, cette attaque s'appuie sur le fait que la procédure d'association d'un nœud au réseau base sa sécurité sur le secret $\mathcal{K}_{\text{join}}$. Ainsi, si cette clé est compromise via une *trash-can attack* sur un seul nœud comme nous l'avons fait, c'est la sécurité de tout le réseau qui est mise en péril. Bien que complexifiant le déploiement, utiliser une $\mathcal{K}_{\text{join}}$ différente pour chaque nœud permettrait d'empêcher un attaquant d'utiliser ce type d'attaques. En effet, même dans le cas où une $\mathcal{K}_{\text{join}}$ serait compromise pour un nœud donné, elle ne permettrait pas à l'attaquant de l'utiliser pour en extraire les $\mathcal{K}_{\text{session\_unicast}}$ des autres nœuds. La spécification du protocole propose déjà l'utilisation de cette option, mais forcer son utilisation au lieu de seulement la suggérer nous paraît nécessaire étant donné la criticité des attaques possibles dans le cas où une $\mathcal{K}_{\text{join}}$ unique serait compromise.

5.2 Retrait de l'envoi de la commande `Suspend` via messages de *broadcasts*

Utiliser des messages à destination d'adresses *unicasts* pour les messages de `Suspend` empêcherait, dans le cadre de notre modèle de menace, l'envoi par l'attaquant de cette commande en *broadcast*. Ces messages utiliseraient les $\mathcal{K}_{\text{session\_unicast}}$ entre chaque mote et le *network manager*. L'impact fonctionnel de cette contre-mesure pourrait cependant être significatif pour des réseaux comportant un nombre important de nœuds en introduisant une latence supplémentaire. A noter que cette contre-mesure n'est pleinement efficace uniquement si elle est utilisée conjointement avec celle présentée dans la Section 5.1. En effet, son efficacité se base sur la non connaissance par l'attaquant des $\mathcal{K}_{\text{session\_unicast}}$ entre le *network manager* et les nœuds victimes. Or, en l'état, l'attaquant peut les récupérer soit en forçant l'exécution d'une procédure d'association en utilisant une attaque de désynchronisation temporelle, soit en écoutant passivement l'association au réseau de nœuds rejoignant le réseau sans intervention de

l'attaquant. L'application forcée d'une $\mathcal{K}_{\text{join}}$ différente par nœud est donc nécessaire afin de garantir l'efficacité de la contre-mesure présentée dans cette section.

6 Conclusion

Dans cet article, nous avons présenté une analyse expérimentale approfondie de la sécurité du protocole WirelessHART, un protocole insuffisamment étudié par la communauté scientifique dû à une spécification complexe, difficile d'accès et entrelacée avec la spécification du protocole HART.

Nous avons développé et mis à disposition de la communauté un outillage open-source complet permettant l'analyse passive et active des réseaux WirelessHART. Notre sniffer, développé sur la base du framework WHAD et d'un dongle nRF52840, constitue une alternative accessible et peu coûteuse aux solutions basées sur radio logicielle. Nos travaux ont également permis de valider expérimentalement un scénario d'attaque de dé-authentification massive par injection de commandes `Suspend`, jusqu'alors uniquement décrit théoriquement. De plus, nous avons étendu cette attaque pour réaliser une usurpation d'identité complète d'un mote légitime en exploitant le comportement des motes Analog Devices lors des phases de reprise suite à un `Suspend`. Enfin, nous avons identifié et évalué une vulnérabilité de désynchronisation temporelle ciblant le mécanisme de synchronisation entre motes.

L'ensemble de ces attaques repose sur la compromission préalable de la clé d'association, dont la distribution et la gestion constituent un maillon critique dans le modèle de sécurité de WirelessHART. L'utilisation de clés par défaut connues ou faibles dans les environnements industriels rend ces scénarios d'attaque faciles d'exécution.

L'analyse comparative avec d'autres protocoles industriels sans fil pourrait permettre d'identifier des bonnes pratiques transposables et d'orienter l'évolution future des standards de communication industrielle.

7 Remerciements

Ce travail a bénéficié d'une aide de l'État gérée par l'Agence Nationale de la Recherche au titre de France 2030 portant la référence "ANR-22-PECY-0009".

Publication des outils

Le code des outils développés est intégré au sein des outils suivants :
— `https://github.com/whad-team/whad-client`
— `https://github.com/whad-team/butterfly`
L'intégralité du code est disponible sous licence libre (MIT).

Note sur l'IA

Les auteurs ont utilisé un assistant d'intelligence artificielle génératif exclusivement pour des corrections linguistiques et des reformulations, sans contribution au contenu scientifique du présent article.

Références

1. C. Alcaraz and J. Lopez. A security analysis for wireless sensor mesh networks in highly critical systems. In *IEEE Transactions on Systems, Man, and Cybernetics, Part C : Applications and Reviews*, volume 40, pages 419–428, 2010. `https://www.nics.uma.es/publications`
2. Lyes Bayou, David Espes, Nora Cuppens-Boulahia, and Frédéric Cuppens. Security analysis of wirelesshart communication scheme. In Frédéric Cuppens, Lingyu Wang, Nora Cuppens-Boulahia, Nadia Tawbi, and Joaquin Garcia-Alfaro, editors, *Foundations and Practice of Security*, pages 223–238, Cham, 2017. Springer International Publishing.
3. Lyes Bayou, David Espes, Nora Cuppens-Boulahia, and Frédéric Cuppens. Security issue of wirelesshart based scada systems. 07 2015. `10.1007/978-3-319-31811-0_14`
4. Damien Cauquil and Romain Cayre. One for all and all for WHAD : Wireless shenanigans made easy ! DEF CON 2024, DEF CON Security Conference, 8-11 August 2024, Las Vegas, NV, USA. `https://defcon.org/html/defcon-32/dc-32-speakers.html`
5. Romain Cayre. ButteRFly. `https://github.com/whad-team/butterfly`
6. I-Chun Chao, Kang Lee, Frederick Proctor, Chien-Chung Shen, and Shinn-Yan Lin. Software-defined radio based measurement platform for wireless networks. volume 2015, 10 2015.
7. Xia Cheng, Junyang Shi, and Mo Sha. Cracking the channel hopping sequences in ieee 802.15.4e-based industrial tsch networks. In *Proceedings of the International Conference on Internet of Things Design and Implementation*, IoTDI '19, page 130–141, New York, NY, USA, 2019. Association for Computing Machinery. `https://doi.org/10.1145/3302505.3310075`
8. Xia Cheng, Junyang Shi, and Mo Sha. Cracking the channel hopping sequences in ieee 802.15.4e-based industrial tsch networks. *IoTDI*, 2019.
9. Xia Cheng, Junyang Shi, Mo Sha, and Guo Linke. Revealing smart selective jamming attacks in wirelesshart networks. *IEEE/ACM Transactions on Networking*, 31(4), August 2023.

10. Max F. H. Duijsens. WirelessHART : A Security Analysis.
https://pure.tue.nl/ws/portalfiles/portal/47038470/800499-1.pdf
11. FieldComm Group. Documentation wirelesshart.
https://www.fieldcommgroup.org/hart-specifications
12. FieldComm Group. Wirelesshart user case studies, 2019.
https://www.fieldcommgroup.org/sites/default/files/imce_files/technology/documents/WirelessHART%20User%20Case%20Studies%20-%20web%20publishing.pdf
13. Martin Gunnarsson. Formal verification of the wirelesshart protocol - verifying old and finding new attacks. 2022.
https://api.semanticscholar.org/CorpusID:253781394
14. Harrison Kurunathan, Ricardo Severino, Anis Koubâa, and Eduardo Tovar. Ieee 802.15.4e in a nutshell : Survey and performance evaluation. *IEEE Communications Surveys & Tutorials*, 2018. Also available as CISTER Technical Report CISTER-TR-180203.
15. Novella Lorente and Eduardo Pablo. Reverse engineering wirelesshart hardware. Master's thesis, Radboud University, Nijmegen and Delft, The Netherlands, August 2015.
16. Fuyuan Luo, Tao Feng, and Lu Zheng. Formal security evaluation and improvement of wireless hart protocol in industrial wireless network. *Security and Communication Networks*, 2021(1) :8090547, 2021.
https://doi.org/10.1155/2021/8090547
17. Mark Nixon. A Comparison of WirelessHART and ISA100.11a. White Paper, Emerson Process Management, 2012.
18. Arun Mozhi Devan Panneer Selvam, Fawnizu Azmadi Hussin, Rosdiazli Ibrahim, Kishore Bingi, and Farooq Khanday. A survey on the application of wirelesshart for industrial process monitoring and control. *Sensors*, 21, 07 2021.
19. Duarte Raposo, André Rodrigues, Soraya Sinche, Jorge Sá Silva, and Fernando Boavida. Securing wirelesshart : Monitoring, exploring and detecting new vulnerabilities. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–9, 2018.
https://doi.org/10.1109/NCA.2018.8548060
20. Shahid Raza, Adriaan Slabbert, Thiemo Voigt, and Krister Landernäs. Security considerations for the wirelesshart protocol. In *2009 IEEE Conference on Emerging Technologies & Factory Automation*, pages 1–8, 2009.
21. Jianping Song, Song Han, Aloysius K. Mok, Deji Chen, Mike Lucas, Mark Nixon, and Wally Pratt. Wirelesshart : Applying wireless technology in real-time industrial process control. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2024.