Sommaire

Liste des figures

Chapitre 1

Définition et présentation du système Hitag-2

1.1 Description Hitag-2

Définition

Hitag-2 est un algorithme de chiffrement par flot utilisé dans les systèmes RKE (Remote Keyless Entry), également connus sous le nom de systèmes d'ouverture à distance des véhicules. Il a été conçu par Mikron, une société autrichienne qui a été acquise par Philips Semiconductors en 1995.

Description de l'algorithme

L'algorithme Hitag-2 est le cousin de "Crypto1", qui est utilisé dans les cartes NFC Mifare Classic. Il repose sur une clé secrète de 48 bits, un état interne de 48 bits, une fonction linéaire basée sur un registre à décalage à rétroaction linéaire (LFSR) agissant sur cet état interne, et une fonction de filtrage non linéaire f. Cette fonction de filtrage prend 20 bits en entrée et génère 1 bit en sortie à chaque cycle d'horloge.

La fonction de sortie peut être visualisée comme une composition de deux niveaux de multiplexeurs différents. Les quatre bits d'entrée des fonctions fa et fb sont utilisés pour adresser et sélectionner l'un des bits de données stockés dans l'état interne. Ces fonctions permettent de sélectionner 5 bits. La fonction de sortie fc prend ensuite les données provenant de fa et fb (5 bits) en entrée et produit 1 bit de données en sortie.

L'algorithme est capable de générer autant de bits que nécessaire pour un chiffrement par flot grâce à son keystream. Cependant, seuls les 32 premiers bits du keystream ks sont utilisés comme authentifiant dans le contexte d'utilisation de RKE (la figure ci-dessous décrit la structure de cet algorithme).

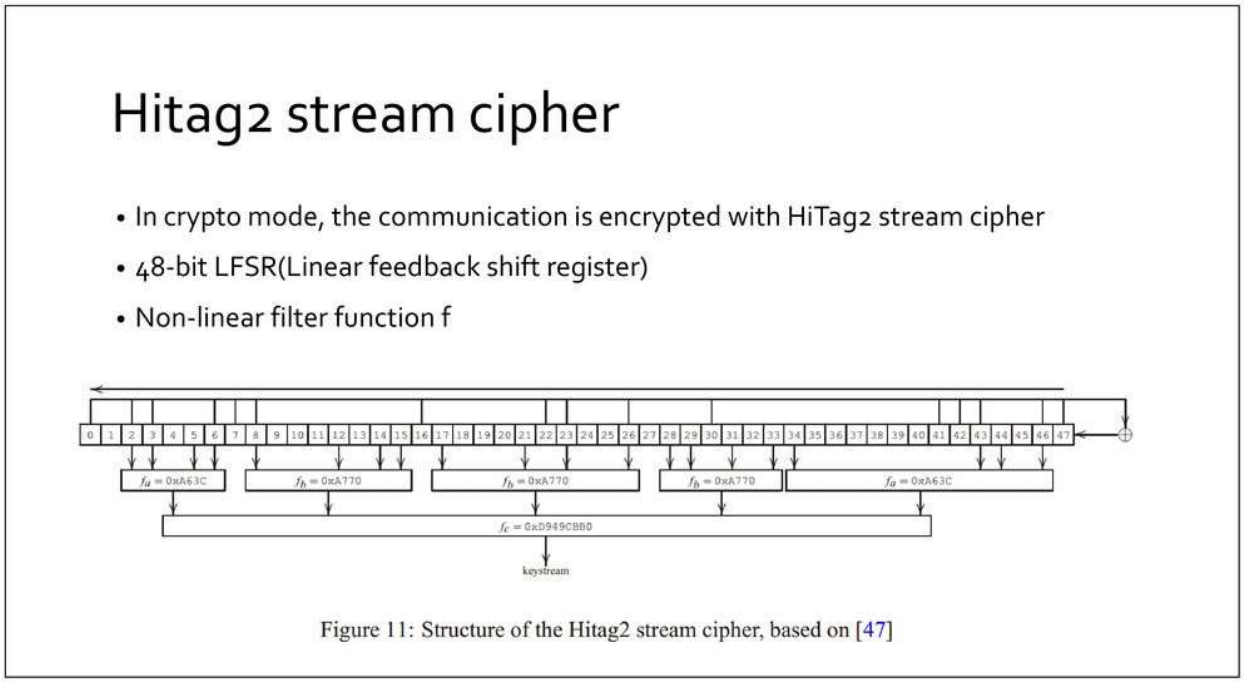

Figure 1.1: Hitag-2 structure algorithmique

Préliminaires

- Le corps fini à deux éléments 0 et 1 est noté F_2. L'opération d'addition XOR dans ce corps est représentée par le symbole $\oplus$. L'opération de multiplication (ET logique) est représentée par le symbole &. F_2^n représente l'ensemble résultant du produit cartésien de F_2 pris n fois : $F_2^n = F_2 \times \ldots \times F_2$ (n fois).
- Une chaîne x composée de n bits a son i-ème bit noté x_i, avec $0 \leq i < n$. La chaîne est représentée par $x_0 \ldots x_{n-1}$. Par exemple, en notation hexadécimale, $0x01$ (équivalente à 00000001).
- Selon la notation précédente, $(0x010000)_i$ sélectionne le i-ème bit de la chaîne de 24 bits $0x010000$. Ainsi, $(0x010000)_7$ est égal à 1, et $(0x010000)_i$ est égal à 0 pour tout i compris entre 0 et 23, à l'exception de $i = 7$, car la chaîne de 24 bits $0x010000$ est représentée par (000000010000000000000000).
- L'opération d'addition $\oplus$ entre deux chaînes de bits x et y de même taille n est définie comme l'addition bit à bit : chaque bit de x est combiné avec le bit correspondant de y pour donner une chaîne de n bits résultante.
- La concaténation de deux chaînes de bits x et y, qui peuvent avoir des tailles différentes, est indiquée par xy ou $x||y$.

Fonctionnement

Tout d'abord, nous commençons par présenter formellement la définition de la fonction de rétroaction linéaire L et de la fonction non linéaire f. Ensuite, nous procédons à une explication détaillée du fonctionnement de ce système.

- La fonction L est définie de $F_2^{48} \to F_2$ par $L(x_0 ... x_{47}) = x_0 \oplus x_2 \oplus x_3 \oplus x_6 \oplus x_7 \oplus x_8 \oplus x_{16} \oplus x_{22} \oplus x_{23} \oplus x_{26} \oplus x_{30} \oplus x_{41} \oplus x_{42} \oplus x_{43} \oplus x_{46} \oplus x_{47}$.

- La fonction non linéaire f est définie de $F_2^4 \to F_2$ par

  $f(x) = fc(fa(x_2x_3x_5x_6), fb(x_8x_{12}x_{14}x_{15}), fb(x_{17}x_{21}x_{23}x_{26}), fb(x_{28}x_{29}x_{31}x_{33}), fa(x_{34}x_{43}x_{44}x_{46}))$, où fa, fb, fc sont des sous-fonctions définies par :

  - fa de $F_2^4 \to F_2$: $fa(i) = (0xA63C)_i$
  - fb de $F_2^4 \to F_2$: $fb(i) = (0xA770)_i$
  - fc de $F_2^5 \to F_2$: $fc(i) = (0xD949CBB0)_i$

Maintenant, nous allons expliquer le déroulement de ce système qui est basé sur les puces de type PCF7946/7947. L'algorithme prend trois paramètres en entrée : un vecteur d'initialisation iv de 32 bits, une clé k de 48 bits et un identifiant id de 32 bits.

Le fonctionnement se déroule en trois phases : la phase d'initialisation, la phase de randomisation et la phase nominale.

Lors de la première et de la deuxième phase, l'état interne du registre évolue pendant 80 cycles et seule la fonction f intervient.

La phase d'initialisation consiste à assigner les valeurs de l'identifiant id aux 32 premiers bits de l'état interne, puis les 16 premiers bits de la clé k aux bits restants du registre (31-47).

La phase de randomisation consiste à faire évoluer l'état du registre pendant 32 cycles d'horloge, et plus exactement les 32 derniers bits (on verra dans la suite que cela représente une faille dans le système car les 16 premiers bits de l'état interne ne varient pas durant cette phase et ces bits sont ceux de la clé k), en faisant un XOR entre les valeurs du vecteur d'initialisation iv avec les 32 derniers bits de la clé k et la sortie de la fonction non linéaire f. Plus formellement (en notant "a" l'état du registre), on peut décrire cette variation mathématiquement par :

$$a_{48+i} = k_{16+i} \oplus iv_i \oplus f(a_i \dots a_{i+47})$$

Dans la phase nominale, le déroulement consiste à effectuer des décalages avec le LFSR sur l'état interne du registre. Et dans le cas de RKE, on prend que les 32 premiers bits sortant de l'algorithme. Plus formellement, on peut écrire :

$$Ks(i) = f(a_{32+i} \dots a_{79+i})$$

Le format de la trame du paquet radio envoyé par le PCF7946/7947 est donné dans la figure ci-dessous.

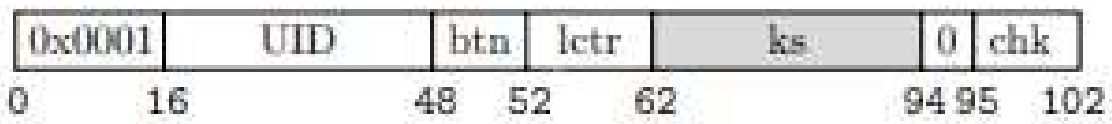

Figure 1.2: Trame PCF7946/7947

Explications relatives au paquet envoyé par le PCF

On voit que le paquet se compose de 6 champs, on donnera une définition de chacun.

- Le premier ($0x0001$) qu'on note SYNC, est une séquence de synchronisation de 16 bits envoyée en préambule pour assurer la bonne synchronisation d'horloge du côté du récepteur.

- UID représente l'identifiant unique.
- btn représente l'identifiant du bouton codé sur 4 bits.
- lctr représente les 10 bits de points faibles du compteur RKE.
- ks est le keystream de 32 bits généré par l'algorithme.
- chk est le checksum codé sur 8 bits qui est le XOR de tous les paquets précédents hors SYNC.

La taille de la trame est de 102 bits et 102 n'étant pas un multiple de 8, on rajoute un padding '10' juste après ks afin que chk ait un sens.

1.2 Faiblesses cryptographiques

Dans cette section, nous allons discuter des vulnérabilités et des faiblesses cryptographiques du système. En outre, nous aborderons également des notions connexes qui sont liées à ces faiblesses.

Le protocole HITAG 2 offre trois modes de fonctionnement : le mode public, le mode password et le mode crypto.

En mode public, le contenu des pages de données utilisateur est simplement diffusé par le transpondeur une fois qu'il est alimenté.

En mode password, le lecteur et le transpondeur s'authentifient mutuellement en échangeant leurs mots de passe. La communication est effectuée en clair, ce qui rend cette procédure d'authentification vulnérable aux attaques de rejeu.

En mode crypto, le lecteur et le transpondeur effectuent une authentification mutuelle à l'aide d'une clé partagée de 48 bits. La communication entre le lecteur et le transpondeur est chiffrée à l'aide d'un chiffre de flux propriétaire.

Nous nous concentrons sur le mode crypto. Notre objectif est d'analyser le système et de détecter toutes les faiblesses et vulnérabilités mathématiques qui pourraient exister dans ce mode.

Afin de mieux comprendre l'une de ces faiblesses, on doit comprendre le protocole d'authentification.

Dans ce protocole, le lecteur commence la communication en envoyant une commande d'authentification, à laquelle le transpondeur répond en envoyant son identifiant. À partir de ce point, la communication est chiffrée en utilisant le keystream par une opération de XOR. Le lecteur répond avec son défi chiffré (nR) et la réponse ($aR = 0xFFFFFFFF$), également chiffrée, pour prouver sa connaissance de la clé. Le transpondeur termine en envoyant sa réponse chiffrée (aT), correspondant au défi du lecteur.

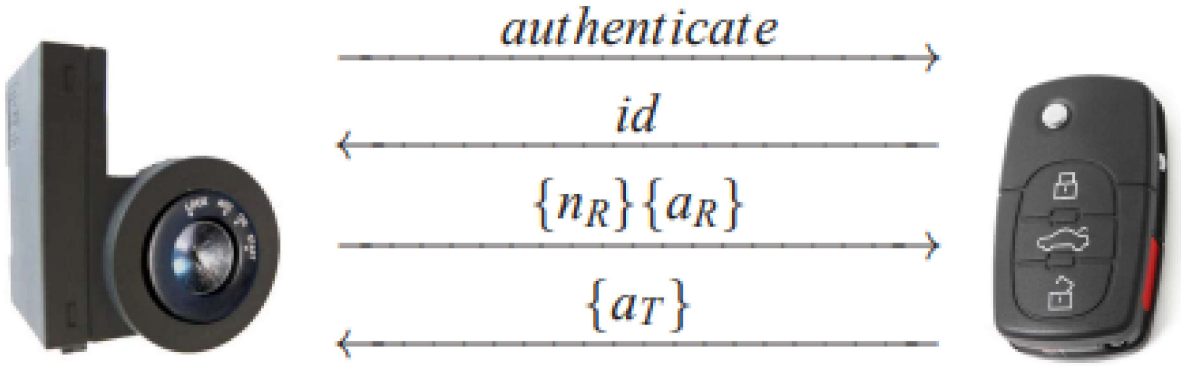

Figure 12: Hitag2 authentication protocol

Figure 1.3: Hitag-2 protocole d'authentification

Taille des clés

- L'état interne de 48 bits : l'état interne du chiffrement ne fait que 48 bits, ce qui le rend vulnérable à certaines attaques.

Dépendance entre les sessions

1. Les bits LFSR (Linear Feedback Shift Register) du chiffrement ne sont pas complètement initialisés à chaque nouvelle session, ce qui crée une dépendance entre les différentes sessions.
2. Les bits LFSR de 0 à 15 restent constants à travers différentes sessions, et seuls les bits LFSR de 16 à 47 varient, ce qui crée une forte dépendance entre eux.

Faible degré d'utilisation des bits de l'état interne dans la fonction de filtrage f

1. Faibles variations de ks lors de faibles variations des entrées id et iv.
2. À l'issue de la phase d'initialisation, l'état du registre interne est constant pour toutes les sessions d'un même transpondeur.

1.3 Attaques possibles

Le système Hitag-2 présente plusieurs vulnérabilités majeures et est sujet à différentes attaques.

1.3.1 Attaque par recherche exhaustive

La faible taille de la clé de 48 bits rend le système vulnérable à une attaque par recherche exhaustive. Cette attaque consiste à utiliser seulement 2 triplets (id, iv, ks) pour récupérer la clé k complète. Cependant, un seul triplet ne suffit pas pour retrouver de manière non ambiguë la clé k en raison des collisions possibles. Ainsi, un deuxième triplet est nécessaire pour lever cette ambiguïté. Des recherches exhaustives ont été réalisées sur différents types de processeurs, prenant plusieurs mois sur un processeur standard, mais seulement quelques heures sur une carte graphique performante.

1.3.2 Attaques algébriques

Le système Hitag-2 est également vulnérable aux attaques algébriques. Ces attaques exploitent les équations booléennes quadratiques multivariées qui décrivent les liens entre la sortie ks et les entrées id, iv et k. Des recherches récentes ont montré que ces équations peuvent être simplifiées dans le cas de Hitag-2, facilitant ainsi le travail d'un solveur SAT. Le temps estimé pour résoudre ces équations est d'environ 6 heures en utilisant 16 paquets avec iv choisi, ou 45 heures avec 4 paquets aléatoires. Des attaques algébriques étendues ont également été proposées en utilisant des solveurs spécifiques aux algorithmes cryptographiques. Toutefois, peu de détails sont disponibles sur le contexte précis de ces attaques.

1.3.3 Attaques marginales

Les attaques plus marginales, telles que les attaques par Time Memory Trade Offs et les cube attacks, ne sont pas détaillées dans cet article. Les attaques par Time Memory Trade Offs nécessitent un contexte spécifique où un oracle de keystream fournit une sortie de grande taille, tandis que les cube attacks nécessitent 500 traces générées à partir d'iv choisis.

1.3.4 Attaques par corrélation

L'idée principale de la cryptanalyse par corrélation est de réduire considérablement l'espace de recherche en identifiant des clés candidates ayant un bon score de corrélation. Ce score est calculé en comparant les bits de sortie observés ks avec les bits de clé devinés. Le processus de filtrage permet de sélectionner les candidats les plus plausibles et d'éliminer les moins plausibles lors du passage d'une génération de bits à une autre.

Le calcul du score est réalisé pour des clés candidates de taille variable, de 16 à 48 bits. Par exemple, pour les clés candidates de 16 bits, les bits devinés agissent sur les bits de keystream ks correspondants, et le score est calculé en moyennant les sorties possibles de la fonction de filtrage f. Ce calcul est étendu aux autres clés candidates de taille supérieure, en ne conservant que les meilleurs candidats à chaque étape.

L'attaquant utilise un tableau de taille fixe pour stocker les candidats et élimine progressivement ceux ayant le score le plus faible. Cette contrainte permet de restreindre l'espace de recherche. À la dernière génération, il ne reste que très peu de candidats avec un score élevé, et la bonne clé peut être identifiée par une vérification exhaustive.

Il est important de noter que la connaissance complète de l'IV n'est pas requise pour cette cryptanalyse, car la partie haute du compteur CNTRH n'est pas transmise par le transpondeur. Cependant, les auteurs ont proposé une approche pour estimer la valeur de CNTRH en fonction de l'âge du véhicule, ce qui permet de contourner ce problème. Ainsi, l'attaquant peut relancer la cryptanalyse en utilisant des IV forgés selon des hypothèses plausibles de CNTRH pour trouver la bonne clé.

Le temps de cryptanalyse reste de l'ordre de quelques dizaines de minutes si le nombre d'hypothèses est faible, ce qui a été observé dans les tests sur les véhicules étudiés par les auteurs.

Chapitre 2

Attaque pratique sur Hitag-2

2.1 L'attaque guess and determine dans le cas général

Dans cette partie, nous nous concentrons sur l'attaque Guess and Determine, une méthode redoutable utilisée par les cybercriminels pour infiltrer les systèmes sécurisés. Nous décrivons en détail les mécanismes sous-jacents de cette attaque sophistiquée afin d'offrir une compréhension approfondie de son fonctionnement et de ses implications sur la sécurité informatique.

À travers une analyse rigoureuse, nous examinons les différentes phases de l'attaque Guess and Determine, en mettant notamment l'accent sur la phase de "devinette" où l'attaquant tente de trouver les valeurs appropriées pour exploiter les vulnérabilités du système. Nous mettons en évidence les techniques et les outils couramment utilisés par les attaquants pour réduire le nombre de possibilités, augmentant ainsi leurs chances de réussite.

De plus, nous explorons les conséquences potentielles de l'attaque Guess and Determine sur les systèmes et les données sensibles. Nous examinons les scénarios réels où cette technique a été utilisée avec succès pour contourner les mesures de sécurité et compromettre des informations confidentielles.

Dans notre étude, nous présentons également un exemple pratique d'utilisation d'un chiffrement par flux (stream cipher) avec une clé de 16 bits. Nous démontrons comment il est possible de réduire la complexité totale de l'attaque, initialement de 2^{16}, à seulement 2^{13}. Ce cas concret illustre l'impact significatif de certaines techniques sur la sécurité des systèmes utilisant des clés de chiffrement relativement courtes.

Exemple d'un chiffrement par flot

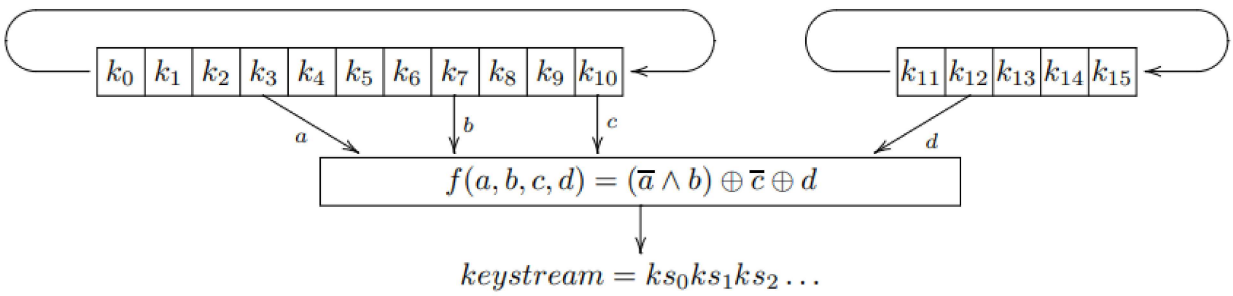

Figure 2.1: Stream cipher

La présente illustration met en évidence le fonctionnement d'un système de chiffrement à flot utilisant une clé de 16 bits. Le système utilise une fonction non linéaire f avec 4 bits en entrée, générant un bit à chaque cycle d'horloge.

À chaque cycle d'horloge, la fonction f utilise les valeurs a, b, c et d en entrée. Dans chaque bloc (à gauche de 11 bits et à droite de 5 bits), les bits subissent un décalage vers la gauche sans que ces valeurs ne soient modifiées. Par exemple, à l'instant $t = 1$, nous aurons un bloc de $k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}k_0k_{12}k_{13}k_{14}k_{15}k_{11}$.

Lorsqu'un registre n'utilise pas son état interne complet pour calculer un keystream, cela le rend vulnérable à l'attaque Guess and Determine (GD) (on verra pourquoi dans la suite).

Dans le cadre de cette attaque, l'attaquant tente de deviner les bits utilisés (a, b, c, d), puis calcule la sortie et l'évalue par rapport au bit correspondant du flux de clés qui a été récupéré.

À chaque évaluation, un certain nombre de candidats sera éliminé, et seuls les bits supplémentaires nécessaires seront devinés. Cela permet de réduire le nombre de possibilités et de concentrer les efforts sur les bits restants.

Explication de l'attaque

La fonction f utilisée dans ce système, et dans la plupart des cas, est une fonction booléenne équilibrée. Cela signifie qu'elle renvoie autant de 0 que de 1. Ces fonctions sont choisies pour générer des nombres pseudo-aléatoires et pour être résistantes aux attaques statistiques.

Grâce à une attaque de force brute, la complexité serait de 2^{16}, cependant, avec l'attaque Guess and Determine (GD), nous pouvons la réduire à 2^{13} dans cet exemple donné.

Comme illustré dans la figure, la fonction f nécessite 4 bits en entrée pour générer un bit ks en sortie. Effectivement, étant donné que seulement 4 bits sont utilisés à chaque itération, il serait inutile d'essayer de deviner tous les bits en une seule fois. La force de l'attaque Guess and Determine repose précisément sur cette observation. L'attaquant se concentre sur la devinette des seuls bits utilisés à chaque itération et s'assure qu'ils produisent le même keystream. Cela permet de réduire considérablement la complexité de l'attaque, car seule une partie des bits est nécessaire pour obtenir le résultat désiré.

Déroulement de l'attaque

Supposons qu'un attaquant dispose de la suite chiffrante $ks_0ks_1...ks_{15}$. Nous aurons uniquement besoin des 5 premiers $ks_0...ks_4$. Nous pouvons les écrire comme :

$$
\begin{aligned}
ks_0 &= f(k_3, k_7, k_{10}, k_{12}) \\
ks_1 &= f(k_4, k_8, k_0, k_{13}) \\
ks_2 &= f(k_5, k_9, k_1, k_{14}) \\
ks_3 &= f(k_6, k_{10}, k_2, k_{15}) \\
ks_4 &= f(k_7, k_0, k_3, k_{11})
\end{aligned}
$$

Nous observons que ks_0 est déterminé par les bits k_3, k_7, k_{10} et k_{12}. L'attaquant doit donc deviner 16 candidats possibles ($2^4 = 16$). Supposons que ks_0 soit égal à 1. Dans ce cas, la moitié des 16 candidats sera éliminée grâce à l'équilibrage de la fonction $f()$. Ainsi, l'attaquant réussit à réduire le nombre de candidats potentiels à deviner à 8 ($2^3 = 8$).

De même pour ks_1, l'attaquant doit deviner les 4 bits k_4, k_8, k_0 et k_{13}. On constate que ces 4 bits sont complètement différents de ceux qui ont produit ks_0. Par conséquent, l'attaquant devra deviner parmi les 16 candidats possibles. Cependant, grâce à l'équilibrage de la fonction $f()$, la moitié des candidats peut être éliminée, ce qui réduit le nombre de candidats restants à $2^3 = 8$. L'attaquant a donc maintenant deviné une clé de 8 bits : k_0, k_3, k_4, k_7, k_8, k_{10}, k_{12}, k_{13}.

Contrairement aux 256 candidats possibles lorsque 8 bits sont inconnus, après avoir calculé les $2^7 = 128$ candidats possibles, on peut encore éliminer la moitié d'entre eux. Cela réduit le nombre de candidats restants à $2^6 = 64$.

L'adversaire applique la même technique pour le 3e bit ks_2. Il va deviner à nouveau une clé de 12 bits après $2^6 \times 2^4 = 2^{10}$ calculs, et la moitié sera éliminée, il en reste donc 2^9 candidats pour une clé de 12 bits.

Pour le 4e bit $ks_3 = f(k_6, k_{10}, k_2, k_{15})$, l'adversaire n'aura que 3 bits à deviner car le k_{10} a déjà été deviné précédemment pour calculer ks_0, ce qui fait $2^9 \times 2^3 = 2^{12}$, et après avoir évalué ks_3, l'espace des candidats diminue à 2^{11}.

Il reste maintenant à déterminer k_{11} (qui n'a pas encore été utilisé pour le calcul d'un ks). Sachant que $ks_4 = f(k_7, k_0, k_3, k_{11})$, l'adversaire n'aura qu'un bit à deviner, donc l'espace des candidats sera de $2 \times 2^{11} = 2^{12}$, puis il sera réduit à nouveau à 2^{11}.

Finalement, nous avons une complexité totale de $2^4 + 2^7 + 2^{10} + (2 \times 2^{12}) = 2^{13}$.

2.2 Guess and determine appliquée à Hitag-2

L'idée, le principe et le fonctionnement de l'attaque

L'idée de l'attaque repose sur l'inversion de la fonction booléenne non linéaire f qui a 5 bits en entrée, et ces 5 bits sont les sorties des fonctions qui ont à leur tour 4 bits en entrée. Le but de l'attaque n'est pas de retrouver directement la clé secrète K, mais de retrouver l'état interne du registre, ce qui permet de trouver une clé équivalente.

Une fois que l'état interne réel est connu, cette idée peut être généralisée en inversant l'initialisation complète tout en devinant tous les bits de nonce inconnus. Cela permet d'obtenir une clé équivalente qui restera valide jusqu'à ce que tous les bits de compteur que nous pouvons observer débordent dans les bits d'état interne que nous avons devinés jusqu'à présent. Une telle clé équivalente, pour n'importe quelle valeur de $CNTR_L$, permet à l'adversaire de falsifier en moyenne les $2^9 = 512$ prochaines valeurs de ks. Après ce point, il existe plusieurs candidats pour une clé équivalente, car des inversions de bits en cascade dans les bits non observés du compteur peuvent s'être produites en incrémentant le compteur jusqu'au point de débordement des 10 bits les plus bas. À partir de tout nouveau triplet observé de KS, BTN et $CNTR_L$, ainsi que des informations que nous avons utilisées pour calculer une clé équivalente précédente, une nouvelle clé équivalente peut être calculée de manière plus efficace.

Supposons que l'attaquant a intercepté une suite $ks_0 ks_1 ... ks_{31}$. Il va donc essayer de deviner les bits qui produisent chaque bit ks_i. Il n'aura besoin que des 9 premiers bits $ks_0, ks_1 ... ks_8$. Plus formellement, selon le schéma du système Hitag-2, on peut écrire :

$$ks_0 = f(a_2, a_3, a_5, a_6, a_8, a_{12}, a_{14}, a_{15}, a_{17}, a_{21}, a_{23}, a_{26}, a_{28}, a_{29}, a_{31}, a_{33}, a_{34}, a_{43}, a_{44}, a_{46})$$

Ces 20 bits sont l'entrée des sous-fonctions fb et fa. On peut représenter ces bits sous forme de masques. Par exemple, si nous devons deviner (a_0, a_1, a_5), nous utilisons le masque (110001), où un bit à deviner est noté par 1.

Pour ks_0, il y aura donc 2^{20} candidats capables de produire ce bit. Cependant, la moitié des candidats sera éliminée lors de cette phase grâce au fait que f est une fonction équilibrée.

Ensuite, pour ks_1, le registre est décalé à gauche et on peut écrire :

$$ks_1 = f(a_3, a_4, a_6, a_7, a_9, a_{13}, a_{15}, a_{16}, a_{18}, a_{22}, a_{24}, a_{27}, a_{29}, a_{30}, a_{32}, a_{34}, a_{35}, a_{44}, a_{45}, a_{47})$$

Or, $a_3, a_6, a_{15}, a_{29}, a_{34}, a_{44}$ ont déjà été devinés, il y a donc 14 bits à deviner au lieu de 20.

À ce stade, l'attaquant aurait $2^{19} \times 2^{14} = 2^{33}$ candidats, puis il élimine la moitié des candidats, ce qui fait 2^{32} candidats restants.

En suivant la même logique, pour ks_2, il n'aura que 4 bits à deviner, ce qui fera $(2^{32} \times 2^4)/2 = 2^{35}$ candidats.

Pour ks_3, il n'aura que 3 bits à deviner, ce qui fera $(2^{35} \times 2^3)/2 = 2^{37}$ candidats.

À partir de ks_3 jusqu'à ks_8, il n'y a qu'un seul bit à deviner dans chaque étape (layer). Cet algorithme produira approximativement 2^{39} états possibles du LFSR qui ont généré la suite $ks_0...ks_8$. Il suffit donc de tester ces 2^{39} états contre les 23 bits restants $ks_9...ks_{31}$, ce qui produira une liste de 2^{16} états candidats.

Les deux figures suivantes présentent cette démarche. La première figure montre les bits à deviner dans chaque étape et ceux qui ont déjà été devinés, et la deuxième figure montre comment les décrire sous forme de masques.

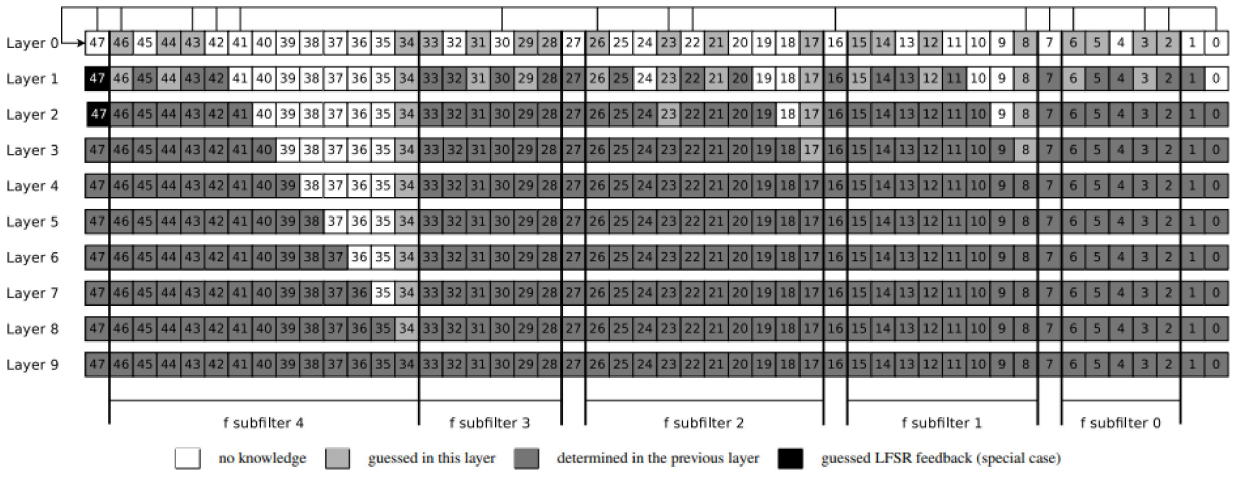

Figure 2.2: Layers

Layer	Mask	Bits	Guess LFSR feedback
0	0x5806b4a2d16c	20	yes
1	0x5004a4a29148	14	yes
2	0x000400820100	4	no
3	0x000400020100	3	no
4	0x000400000000	1	no
5	0x000400000000	1	no
6	0x000400000000	1	no
7	0x000400000000	1	no
8	0x000400000000	1	no

Table 2: Which bits to guess at each layer.

Figure 2.3: Bits à deviner

On peut ensuite réduire ces 2^{16} candidats à un seul candidat en suivant les étapes suivantes :

1. **Recoupement des bits de nonce :** Le protocole utilisé dans le chiffrement Hitag2 utilise des bits de nonce qui se chevauchent. Cela signifie que certains bits du nonce sont utilisés à la fois pour le premier échantillon de flux de clés et pour le second échantillon de flux de clés. Ces informations de chevauchement permettent de réduire la liste des états de chiffrement potentiels à un seul état réel qui l'a généré.

2. **Recul de chiffrement :** En utilisant un deuxième échantillon de flux de clés observé à partir du même émetteur et les informations de nonce observables utilisées pour générer les deux échantillons, le chiffrement peut être remonté en arrière en initialisant les bits de nonce connus les plus bas vers un état de chiffrement antérieur. À partir de cet état pré-initialisé, les bits de nonce observés associés au deuxième échantillon peuvent être chiffrés pour compléter un deuxième état initialisé. Si cet état génère le deuxième échantillon de flux de clés, les états initiaux corrects liés à ces deux échantillons sont identifiés.

3. **Prédiction de l'état suivant :** Comme les bits de compteur les moins significatifs se trouvent dans les bits de nonce connus, il est possible d'incrémenter simplement ces bits de compteur et de choisir les bits de code de bouton appropriés pour prédire l'état de chiffrement initial suivant.

Chapitre 3

Implémentation

Dans cette partie, on explique brièvement le principe de chaque attaque et optimisation possbile. Une explication encore mieux détaillée est donnée dans les prorgrammes implémentés (.py, .c et .ipynb)

3.1 Brute Force

Une clé de 48 bits est largement en dessous des standards modernes (et même de ceux des années 2000). Elle rend la recherche exhaustive accessible en un temps raisonnable. Celle-ci nécessite 2 triplets (id, iv, ks) pour récupérer les 48 bits de la clé K. Notons qu'un seul triplet ne suffit pas pour retrouver de manière non ambiguë K : puisque ks fait 32 bits, il y a en moyenne $2^{48-32} = 2^{16}$ clés K produisant le même triplet (id, iv, ks) à cause des collisions possibles, le second triplet permet donc de lever cette ambiguïté.

Cette attaque vise directement la clé secrète K. Elle itère de façon itérative sur toutes les 2^{48} clés possibles, et elle teste chacune d'entre elles pour voir si elle produit le bon keystream.

3.2 GD version naive récursive

Voici une explication un peu plus formelle de cette implémentation :

1. Tant que l'état n'est pas entièrement déterminé :
   (a) Déterminer tous les bits d'entrée du filtre de la couche en itérant un compteur.
   (b) Étendre la valeur du compteur avec le masque de la couche en utilisant la fonction d'expansion.
   (c) Combiner la valeur étendue avec l'état de la couche précédente.
   (d) Calculer et vérifier le résultat de la fonction f par rapport au bit de flux de clés observé pour cette couche.
   (e) Si la supposition était correcte, passer à la couche suivante avec un bit de rétroaction LFSR (supposé ou calculé) mis à jour.
2. Lorsque l'état est entièrement déterminé, le tester par rapport au reste du flux de clés observé en utilisant la fonction de test.

En résumé, cette description explique la procédure générale de l'attaque en couches (layer). À chaque couche, les bits d'entrée du filtre sont déterminés en itérant un compteur et en utilisant la fonction d'expansion (expand) pour obtenir une valeur étendue. Cette valeur est ensuite combinée avec l'état de la couche précédente. Ensuite, la fonction f est calculée et vérifiée par rapport au bit de flux de clés observé pour cette couche. Si la supposition est correcte, l'attaque passe à la couche suivante en mettant à jour le bit de rétroaction LFSR.