

ICON OF COIL

Comprendre les ECM/EMM

Traduction la Horde .
Quatrième édition

Introduction.

Ce document est une tentative, sans la prétention d'être un document exhaustif, de collecte et de réorganisation de divers postes et sujets existants, sur le NET, sur le Seca2. Pour la compréhension du ce document, il est nécessaire de connaître le fonctionnement du système SECA1. Pour les débutants lisez les divers documents disponibles sur la toile. Tout ce qui est écrit dans ce document sert exclusivement à la recherche et à l'étude ludique.

La vision des chaînes cryptées sans abonnement est une infraction punissable par la loi. L'auteur n'est pas responsable des dommages éventuels ou des abus qui pourraient être occasionné par l'utilisation détournée de document.

Avant d'exposer quoi que ce soit et pour clarifier le tout, nous allons définir les termes les plus importants qui seront utilisés dans cette FAQ. J'utiliserais, autant que possible, une nomenclature informatique acceptée de tous.

Le Bit.

Un bit est tout d'abord la plus petite unité information utilisée en informatique, il ne peut avoir que la valeur de 0 ou de 1.

Le Quartet ou NIBBLE

Le Quarte vaut 4 bits dans une écriture en hexadécimal il représente une des deux parties. Le quartet fort se trouve à gauche par exemple ici **FA** (**FA** 51) le quartet faible se trouve à droite ici **51**

L' Octet ou BYTE

Un octet comporte 8 bits.

Le Mot ou WORD

Un mot est formé de deux octet, soit de 16 bits.

Le Double Mot ou DWORD

Le double mot, regroupe 4 octets, ou encore deux mots, soit un total de 32 bits.

Le Triple Mot ou FWORD

Le triple mot, regroupe 6 octets, ou encore trois mots, soit un total de 48 bits.

Le Quadruple Mot ou QWORD ou OTTETTO (pour les italiens et les espagnols)

Le quadruple mot, regroupe 8 octets, ou encore quatre mots, soit un total de 64 bits.

Pour ma part j'utiliserais le plus souvent la nomenclature française, celle que j'ai mis en rouge, sauf lorsque le terme anglais utilisé sera devenu, au fil de la FAQ, courant .

Bonne lecture et bonnes études à tous, ça va être passionnant.

Nouveauté du système Seca2

En partant du LOG d'une ECM, il est possible d'établir les premiers caractères du nouveau système :

C1 3C 01 BE 5C
10 01 CD 99 F1 E7 88 F1 E9 00 50 F9 09 5B 02 43
DD 03 35 39 1B C2 41 97 AF B3 8A A7 F7 9A FA 78
12 C9 BA 23 16 42 99 0E 9A F3 31 72 22 BC B8 C6
62 45 37 F9 7F 68 15 7C BB 7C A7 F2 3D F8 27 82
52 49 54 56 98 0C AB 94 26 95 74 A7 12 B6 83 4D
23 46 03 F1 E1 A5 66 31 05 96 46 48 [90 00]

Entre deux ECM tous les octets changent à l'exception de ceux qui sont mis en évidence (ceux en jaunes), après eux plus rien de reconnaissable, plus de NANO, pas de trace de la signature.

La raison en est que les données ont été cryptées.

Il est désormais notoire que sont présents en SEKA2, deux passages de cryptage : la

SUPERENCRYPTION (qui opère sur des blocs d'octets) et la SECONDARY SUPERENCRYPTION (**SSE** ou **ENVELOPPE**) qui agit sur l'ensemble des données.

Ces procédés de cryptage sont **obligatoires** pour les données des instructions C1 3C/38/40.

L'algorithme utilisé pour l'enveloppe n'est pas connu, même si on a émit comme hypothèse un (dé)cryptage du type RSA (pour ceux que ça intéressent, il existe sur la toile, beaucoup de documents sur l'argument), la SUPER-ENCRYPTION est elle, constituée du même algorithme que celui utilisé par le SECA1, auquel on a ajouté un procédé de masquage des octets.

SUPER-ENCRYPTION (SECA1)

On prend le corps de l'instruction et on subdivise en blocs de 8 octets, à chaque bloc on applique l'algorithme de cryptage avec la clé indiquée par P2. Les derniers octets qui ne peuvent pas former un bloc de huit octets, ne sont pas cryptés et restent donc en clair. Pour exemple si vous avez dans une trame, un corps de 34 octets à crypter, vous aurez 4 blocs de huit octets (32 octets) qui seront cryptés et deux octets (les deux derniers du corps) qui resteront en clairs.

L'obligation d'utiliser la SUPER-ENCRYPTION est fixée par le contenu de certains flags qui sont mémorisés dans l'EEPROM. Actuellement nous n'avons aucun procédé de modification de ces flags

La Structure des instructions C13C/38/40

La norme ISO7819 régissant le SECA1 à été maintenue :

C1 38/3C/40 P1 P2 LEN + PAQUET DE DONNEES

- **P1** : sur 1 octet de deux quartets organisés de la façon suivante :

Composition de l'octet de **P1 XY**

X quartet fort bit 7 6 5 4

Y quartet faible bit 3 2 1 0

Le quartet faible (3210) (LOW NIBBLE)

Bit de 0 à 3 : index du PROVIDER à qui on envoie l'instruction, nous avons en effet la possibilité d'aller jusqu'à 1111 (binaire) soit F en hexadécimal 16 providers gérables Seca (0) compris

Le quartet fort (7654) (HIGH NIBBLE)

Bit 4 : utilisation de la clef primaire bit à (0) ou de la clef primaire et secondaire bit à (1)

Bit 5 et 6 : indique comment initialiser le hash buffer pour le calcul de la signature.

Bit 7 : pas utilisé

- **P2** : sur un octet de deux quartets organisé de la façon suivante :

Composition de l'octet de **P2 VZ**

V quartet fort bit 7 6 5 4

Z quartet faible bit 3 2 1 0

Le quartet faible (3210) (LOW NIBBLE)

Bit de 0 à trois : index de la clef à utiliser pour le décryptage, nous avons donc en théorie la possibilité d'utiliser 16 clés, 1111 (binaire) maximum soit 0F (hexadécimal), soit 16 (décimal)

Le quartet fort (7654) (HIGH NIBBLE)

Bit 4 : signification inconnue doit être à 1

Bit 5 et 6 : sélectionne les tables et éventuellement le user Algo (pas actif sur V7)

Bit 7 : si la trame est sur-encryptée ou pas ; 1 sur-encrypt et 0 pas sur-encrypt pour le seca2 il doit être à 1.

- **LEN** : le nombre d'octets du paquet de données (on commence le calcul juste après l'octet de la LEN)

Exemple de LEN 5C (hexadécimal = 92 octets en vert) :

C1 40 01 B1 **5C**
10 01 12 08 F5 56 DE F0 11 12 D0 D8 40 3B 34 7A
EB A5 B7 30 41 50 5F 02 6D B2 03 AB 29 2B 29 7A
05 4F AF 83 18 75 1F 33 49 67 29 0C C0 22 C7 44
E3 BA 45 6D 1B 3A F3 56 07 A9 89 5D B4 5E 8A D1
1F 40 F4 50 D1 57 D0 96 88 5B EB 93 2A 10 CE E8
4D 36 1F 80 A7 65 A6 9C 3E 03 78 49

Nous avons déjà fait allusion aux deux octets mis en évidence (en jaunes) qui sont identiques pour toutes les INS loggués, ils constituent des paramètres pour la DE-ENVELOPPE et sont gérés de façon différentes par rapport aux autres données. On leur a donné comme définition de paramètre **P3** et **P4**, à l'intérieur du paquet de données nous avons un paramètre supplémentaire qui n'est pas visible car crypté.

Sa fonction sera expliquée par la suite, pour l'instant il suffit de savoir qu'il existe, qu'il correspond au dernier octet et qu'il a été défini comme paramètre **P5**.

A la lumière de tout cela nous pouvons donner une nouvelle représentation de la structure de l'INS :

CLA INS P1 P2 LEN P3 P4 +paquet de données+ P5

Processus d'exécution des INS

Dans les grandes lignes nous devrions avoir cette séquence logique (et temporelle) :

1 – Contrôle normal sur le protocole ISO7816 CLA/INS/LEN (possible erreur statut 6D00, 6E00, 6700)

2 – Contrôle sur P1, présence du Provider (statut 9004 si inexistant)

3 – Contrôle sur bit 4 de P2 (doit être = 1, statut 9024 dans le cas contraire)

4 – Contrôle sur bit 0 de P4 (doit être = 1, statut 9024 dans le cas contraire)

5 – Contrôle sur P3

Le traitement réservé à P3 est similaire à celui d'une NANOCOMMANDE mais :

1 - Doit avoir comme valeur minimum 10 (statut 9024 pour des valeurs inférieures)

2 - Il est complètement ignorer, avec les données auquel il est associé, pour les procédures suivantes :

- Décryptage SSE (DE-ENVELOPPE) et décryptage SE
- PARSING et exécution (Nous entendons par PARSING (littéralement analyse de la Période) dans notre contexte c'est l'examen des NANOCOMMANDES et de leurs paramètres.

3 - Fait partie des données pour le calcul de la signature.

4- P4 est le premier octet de donnée pour P3.

Pour savoir le nombre d'octets associé à P3, nous suivons le tableau classique des LEN

Exemple:

NANO 0x04 = NANO 0x4 de Longueur 0x0, n'est suivie d'aucunes données.

NANO 0x82 = NANO 0x2 de Longueur 0x8, suivie de 8 octets de données.

Les longueurs suivantes sont admissibles:

0 ...C correspondent à 0 ..12 octets,

Les D=16 octets

Les E=24 octets

Les F=32 octets

(IMPORTANT le 0 n'est pas une valeur admissible pour P3)

NANO 0xD1 = NANO 0x1 de longueur 0xD, suivie de 16 octets de données.

Tous les nombres seront donnés en hexadécimal.

Au terme de ce contrôle, nous pouvons obtenir le statut 6700 (Mauvaise longueur de données) si après les donnée de P3, il n'y a pas au moins 0x5A (90 décimal) octets de données, nous expliquerons le motif plus en avant.

6 – Contrôle sur bit 1,2 de P4 (doivent être = 0, statut 9036 dans le cas contraire)

P4 est un indicateur pour le DE-ENVELOPPE mais sa fonction précise est encore inconnue

NOTE :

Pour les réponses suivantes, les octets de statut ne justifie pas la séquence de 1 à 6 :

C1 38 01 91 00 Statut prévu : 9024

Statut obtenu : 9036

C1 38 01 91 01 19 Statut prévu : 6700

Statut obtenu : 9036

7 – Décryptage SSE (ou de-enveloppe)

Bien que l'algorithme ne soit pas connu, on a pu observer qu'il utilisait des clés différentes pour chaque fournisseur d'accès (mais les mêmes clés pour toutes les cartes d'un fournisseur d'accès) et qu'il opère toujours et seulement sur les derniers 0x5A octets de données.

Les conséquences sont :

- Une ECM/EMM adressée à un fournisseur d'accès n'est pas applicable à un autre fournisseur d'accès.
- Les données de P3 (données qui ne sont pas partie prenante au de-enveloppe) doivent être suivies d'au moins 0x5A d'octets (voilà expliqué le pourquoi de 6700 pendant le contrôle sur P3)
- Les éventuels octets ajoutés entre les données de P3 et les dernier 0x5A octets ne sont pas dans la SSE/ENVELOPPE (IMPORTANT)

Exemple :

Note : aucun des exemples
Ne viennent de LOG
réels

C1 40 01 B1 6A
43 51 6E B1 92 0F 87 17 D3 5C 87 47 34 5E 39 79
12 08 F5 56 DE F0 11 12 D0 D8 40 3B 34 7A EB A5
B7 30 41 50 5F 02 6D B2 03 AB 29 2B 29 7A 05 4F
AF 83 18 75 1F 33 49 67 29 0C C0 22 C7 44 E3 BA
45 6D 1B 3A F3 56 07 A9 89 5D B4 5E 8A D1 1F 40
F4 50 D1 57 D0 96 88 5B EB 93 2A 10 E8 4D 36
1F 80 A7 65 A6 9C 3E 03 78 49

Dans cette instruction, les octets mis en évidence en jaune représente P3 et ses données relatives (le quartet fort de P3 = 4, donc nous avons 4 octets de données qui le suivent.

Ceux mis en évidence en bleu clair représente les 0x5A sous De-Enveloppe, les autres (ceux qui sont sur fond blanc) sont en dehors de l'enveloppe et donc sont en **clair ou en SUPER-ENCRYPTION**.

Que trouvons-nous sous la DE-ENVELOPPE ?

On note immédiatement une grosse différence par rapport à SECA1 : la signature est en dehors de la SUPER-ENCRYPTION et n'a pas de position fixe. Reprenons l'instruction précédente et faisons une hypothèse sur une de-enveloppe possible :

C1 40 01 B1 6A
43 51 6E B1 92 0F 87 17 D3 5C 87 47 34 5E 39 79
D7 C6 1A C9 4F E8 40 6C 67 E3 AB 84 4C 29 3F DD
A3 F0 E6 37 86 6D 8A 23 CB 88 55 9D 47 78 B6 71
54 9F 82 D8 85 1C 3E 05 34 36 27 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 P5

P5 est maintenant en clair

Nous pouvons voir P3 avec ses données (en jaune) suivit des octets cryptés en SUPER-ENCRYPTION (en vert ; il est important de rappeler que la partie soulignée, n'a pas subit de de-enveloppe), après est présente la NANO82 + la signature en clair, le reste ne sont que des octets de remplissage jusqu'à l'avant dernier octet (compris).

L'instruction se termine par le paramètre P5, Celui-ci a comme fonction d'indiquer la position de la NANO 82, cela signifie que la carte ne recherche pas la signature en scannant toute la chaîne de commande mais qu'il va directement à la position pointée par P5

Nous en déduisons deux informations supplémentaires sur la position de la NANO 82

- La signature est longue de huit octets, il n'est donc pas admissible que la NANO 82 puisse être dans les 8 dernières positions de la chaîne de commande.
- Les données pour le calcul de la signature doivent être au moins 8, en d'autres mots, la NANO 82 doit être précédée d'au moins 8 octets.

8 – Contrôles sur la valeur assumé par P5

Le calcul que fait la carte pour déterminer la position de la NANO 82 est le suivant (calcul à 8 bits) :

$$\text{Position NANO 82} = \text{LEN} - (\text{P5} + 8)$$

Les valeurs comprises entre 128 et 255 ne sont pas admissibles (nous obtenons un statut de 9037). Si P5 assume la valeur dans la fourchette de 1 à 127, et que le résultat du calcul ne respecte pas les deux informations supplémentaires vues précédemment (voir point 7), nous obtenons un statut de 9037. Il y a un contrôle ultérieure (encore objet d'étude et au mécanisme inconnu) qui n'est pas surpassé donne un statut de 9038. La probabilité d'obtenir de ce statut en faisant varier la valeur des octets sous enveloppe est environs égal à $(1/3) \cdot (1/256)$.

9 - Contrôle sur le bit 7 de P2 (doit être = 1, statut 9035 dans le cas contraire)

C'est le bit qui indique la SUPER-ENCRYPTION.

10 – Recherche de la Key pour le calcul de la signature

Pour le calcul de la signature la clé indiquée par P2 est nécessaire ; selon l'INS considérée, toutes les clés ne sont pas utilisables.

INS 40 : Uniquement les clés de management (Key Index entre 0...B), dans le cas contraire nous obtenons un statut de 9013

INS 3C : Uniquement les clés opérationnelles (Key Index entre C...F), dans le cas contraire nous obtenons un statut de 904A

INS 38 : Pas d'information sur les clés utilisables

A ce stade commence la recherche de la clé dans l'EEPROM, si elle n'est pas présente, on obtiens un statut de 901D dans le cas d'une recherche d'une clé primaire manquante, ou alors un statut de 901F dans le cas d'une clé secondaire manquante. Si on utilise une clé 0F, l'éventuelle clé secondaire n'est pas prise en compte (même si le bit 4 de P1 =1). Une fois la clé repérée, on vérifie le CHECKSUM, si ce contrôle rate alors nous avons le statut 9028

ATTENTION

Le statut 9028 n'apparaît pas seulement au non aboutissement du processus de contrôle du CHECKSUM, il peut aussi apparaître par le non aboutissement des contrôles précédents sur le de-enveloppe, nous n'avons pas encore éclairci la raison.

Deux cas se distinguent à travers les temps de réponse :

9028 PRE-SSE : environs 7300 cycles d'horloge

9028 KEY-CHECKSUM : au dessus de 190000 cycles d'horloge

11 – Vérification de la présence de la NANO 82 et calcul de la signature

La carte prend la valeur calculée au point 8 et vérifie si la NANO 82 est effectivement présente à cette position. Si la NANO n'est pas trouvée on obtiens le statut 9002 (appelé aussi 9002_A ou 9002 type-1). Si la NANO 82 est bien présente alors on procède avec le calcul de la signature. Les octets pris en compte pour calculer la valeur de la signature sont tous les octets qui précèdent la NANO 82 jusqu'à P3 compris (P3 fait partie du calcul de la signature).

```
C1 40 01 B1 6A
43 51 6E B1 92 0F 87 17 D3 5C 87 47 34 5E 39 79
D7 C6 1A C9 4F E8 40 6C 67 E3 AB 84 4C 29 3F DD
A3 F0 E6 37 86 6D 8A 23 CB 88 55 9D 47 78 B6 71
54 9F 82 D8 85 1C 3E 05 34 36 27 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 P5
```

Dans cet exemple tous les octets mis en évidence, en bleu clair, font partie du calcul de la signature. La clé pour le calcul de la signature, comme déjà dit avant, est indiquée par le quartet faible de P2, tandis que les bits 6,5 indiquent les tables HASH à utiliser

Les tables du HASH sont des tables de conversion de données, utilisées par l'algorithme de dé(cryptage), on changeant de table on modifie le résultat du (dé)cryptage

Bit 6	Bit 5	Choix des Tables
0	0	Tables ROM les mêmes que dans SECA 1
0	1	Tables EEPROM (A)
1	0	Algorithme Alternatif
1	1	Tables EEPROM (B)

Si le bit 6 = 1 et le bit 5 = 0, on demande l'exécution du (dé)cryptage via un algorithme alternatif, mémorisé dans l'EEPROM de la carte. Dans la V7 cet algorithme n'est pas présent, une instruction éventuelle envoyée avec le bit 6 = 1 et le bit 5 = 0 donne une statut de 9034.

L'analyse des bits 6 et 5 sont effectués immédiatement avant la vérification de la présence de la NANO 82.

Est venu le moment du calcul de la signature : il est intéressant de noter que la longueur de l'instruction n'influence pas le temps utilisé pour le calcul, cela laisse penser que ce calcul est exécuté par un hardware dédié (CRYPTO-PROCESSEUR). Si la signature calculée est différente de celle présente dans l'instruction, non avons un statut de 9002 (appelé **aussi 9002_B ou 9002 type-2**)

9002_A et 9002_B se différencient par leur temps de réponse :

temps de réponse de (9002_B) > temps de réponse (9002_A)

12 - DECRYPT SUPER-ENCRYPTION

La phase de décryptage des données en SE est composée de deux parties : UNMASKING et DECRYPT agissant sur les octets compris entre P3 et ses données et la NANO 82

```
C1 40 01 B1 6A
43 51 6E B1 92 0F 87 17 D3 5C 87 47 34 5E 39 79
D7 C6 1A C9 4F E8 40 6C 67 E3 AB 84 4C 29 3F DD
A3 F0 E6 37 86 6D 8A 23 CB 88 55 9D 47 78 B6 71
54 9F 82 D8 85 1C 3E 05 34 36 27 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 P5
```

SUPERENCRYPTION

Dans le cas d'en cryptage d'une C1 3C , les premiers huit octets des données ne sont pas désencryptés

Note : dans le SECA1 une série plus petite que 8 octets étaient laissés en clair, par contre en SECA2 on a ajouté un processus de masquage (MASKING), de tous les octets de l'instruction, de tel façon à ne pas laissé un seul octet en clair.

13 - PRE-PARSING du corps de l'INS et exécution de l'instruction

Après la SUPER-ENCRYPTION nous avons une très normale INS de type SECA1, avec les canoniques NANO, mais nous ne sommes pas encore arrivé à leurs exécutions. Il existe une phase de pré-analyse des données, qui devrait contrôler si en sautant de NANO à NANO on arrive à la NANO 82 (NANO de signature). Un exemple rendra tout plus clair :

```
C1 40 01 B1 5C
10 01 [F0] FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF [22] 1A 3F [21] 1A 7F [90] 5D B9 5D 66 DF AC FF 00 45 [90] 5C 55 A8 EE
6F 9F 67 AA 92 [90] 5E EF AB 5A 97 FA BC 6F 91 [82] C9 2D C3S 6C C4 77 74 8B... etc. etc.
```

L'analyse des NANO commandes va de gauche à droite.

Nombre d'Octet	NANO	EXECUTION
1+ 1 Octet	10 01	P3 e P4, ne sont pas considérées dans l'exécution de l'INS, on les sautes
1 + 32	F0	32 octets de données (CUSTOMER WORD POINTER BITMAP)
1 + 2	22	Contrôle de la date de fin d'abonnement
1 + 2	21	Fixe la date de fin d'abonnement
1 + 1 + 8	90	Ecriture de la clé primaire (D)
1 + 1 + 8	90	Ecriture de la clé primaire (C)
1 + 1 + 8	90	Ecriture de la clé primaire (E)

A ce point nous arrivons à la NANO 82 et le processus se termine convenablement.

Processus d'exécution – (seulement pour version V7.0) (Version revue post sur un forum-sat)

ANALYSE DE L'HEADER (QUARTET FAIBLE=LOW NIBBLE - QUARTET FORT= HIGH NIBBLE

01) Vérifie conformité ISO -> si rate -> 6700, 6D00 ou 6E00
02) P1: PROVIDER existe quartet faible (P1)?-> si non -> 9004
03) Vérifie ATR protection LEVEL -> si oui -> 9B00
04) Vérifié ATR LEVEL 3 -> si oui -> 9024
05) P2: bit4=1 ? (présence par. extra) -> si 0 -> 9024
06) P4: bit0=1 ? -> si 0 -> 9024
07) P3: Quartet fort P3) > 0 ? -> si 0 -> 9024 (P3 sans paramètres)
08) P3: LL-Tab[HiByte(P3)] >= 5B ? -> si < 5B -> 6700 (<90 octets sous SSE)
ou Tab[...] est le tableau de la longueur des NANOCOMMANDES
High Nibble | Longueur (hex)

+-----
0x00 -- 0x0C | 0x00 -- 0x0C
0x0D | 0x10
0x0E | 0x18
0x0F | 0x20
09) P4: bit 2-1 = 0-0 ? -> si non -> 9036

DECODIFICATION SSE

10) P5: valeur en clair < 0 et égale à 8?-> si oui -> 9037
11) P5: valeur en clair > longueur maximum ? -> si oui -> 9037
12) P5: valeur licite ? -> si non -> 9038 ** a étudier **
13) P2: bit7=1 ? (SE obligatoire) -> si 0 -> 9035
14) P5: valeur en clair pointe à 82 ? -> si non -> 9002a (82 absent)

CHARGEMENT DE LA CLE DE L'EEPROM

15) P2: INS=3C et clé OPERATIVE -> si non -> 904A
P2: INS=40 et clé de MANAGEMENT -> si non -> 9013
16) P1 P2: la clé existe ? -> si non -> 901D (prim.)/901F (sec.)
EXEPTION : la clé 0F secondaire n'est pas prise en considération pour les
INS C138/C13C
17) Vérifie le CHECKSUM de la clé -> si pas ok -> 9028

CALCULE LA SIGNATURE A PARTIR DE P3 A 82
(à l'exclusion de P3, et de ses paramètres et 82, tout est en SE)

18) INS=40, ATR level2, KEYINDEX >7x -> si oui -> 9022
19) Sélectionne les tables HASH (P2 bit 6,5) -> si USER ALGO -> 9034
20) Confronte avec la signature de la commande -> si différente -> 9002b

DECRYPTAGE SE

21) Vérifie le PARSING -> si erroné -> 9600
22) INS=3C check NANO "interdit" -> si présent -> 9A00
INS=40 check NANO "interdit" -> si présent -> 9A00

EXECUTION

SIGNIFICATION DES OCTETS DE STATUT

0101 Erreur en phase de test-hardware: RAM interne
0102 Erreur en phase de test-hardware: EEPROM
0104 Erreur en phase de test-hardware: RAM externe
0105 Erreur de phase de test-hardware: Dispositifs ajoutés (ex: générateur de numéro aléatoires)
0110 Erreur en phase de test-hardware: ROM checksum erroné
61XX Ok : Octets XX restants
6200 Attention en Traitement
6281 Données de retour peuvent être altérées
6282 Fin d'enregistrement/fichier avant la lecture complète
6283 Les données retournées peuvent contenir une 'information structurale
6284 Fichier non valide
6285 Invalide format FCI
6286 Ecriture non réussie

62xx Avertissement inconnu : 62XX
 6300 Attention Traitement
 6381 Fichier complet
 6382 Correct après relance
 63cx Traitement avertissant : COUNTER=X
 63xx Avertissement inconnu : 63XX
 64xx Erreur d'exécution : 64XX
 6500 Erreur d'exécution
 6501 Erreurs de stockage. Problème de Lecture/Ecriture sur l'EEPROM ou autre problème matériel
 6591 Erreur mémoire
 65xx Erreur d'exécution
 6700 Mauvaise longueur des données d'entrée
 67xx Mauvaise longueur des données essayer XX
 6800 Fonction non supportée par la carte
 6881 Canal logique non supporté
 6882 Transmission sécurisé des messages non supportée
 68xx Fonction non supportée : 68XX
 6900 Commande non permise
 6981 Commande incompatible avec la structure du fichier
 6982 Mode de sécurité non satisfaisant
 6983 La méthode de vérification bloqué
 6984 Les données de référence invalidées
 6985 Conditions d'utilisation non satisfaites
 6986 Aucun EF actuel
 69xx Commande non permise : 69XX
 6A00 Paramètres incorrects de zone des données
 6A80 Paramètre faux dans la partie de données
 6A81 Fonction non supportée
 6A82 Fichier non trouvé
 6A83 Enregistrement non trouvé
 6A84 Cellules de mémoire insuffisantes dans dossier ou enregistrement
 6A85 LC contradictoire avec la structure de TLV
 6A86 P1-P2 incorrect
 6A87 LC contradictoire avec P1-CP2
 6A88 Données de référence non trouvées
 6Axx Mauvais paramètres 6AXX
 6B00 Paramètre/Octet(s) de référence (P1 ou P2) non correct
 6Cxx Le devrait être XX
 6D00 Instruction non supportée / Instruction invalide ou protégée / Instruction non libérée
 6E00 Classe d'instruction (Class) non supportée
 6F00 Pas de diagnostic précis possible
 9000 Commande exécutée sans erreur
 9001 Erreur d'écriture sur l'EEPROM
 9002 Signature erronée (9002_B) ou absente (9002_A)
 9003 Pas assez d'espace pour l'écriture d'un record pour un PROVIDER donné
 9004 Fournisseur non supporté
 9005 INS 40/44 NANO réservée à SECA (00)
 9006 INS 0x3C, NANO 0x15/0x19/0x2C: Pas de mémoire pour les PREVIEW records
 9007 Fournisseur bloqué avec la NANO 24 (utiliser INS 40 NANO 02/01 pour débloquent)
 9008 INS 0x40, NANO 0x01: pas autorisée
 9009 INS 0x40: PPUA pas dans la BITMAP F0,
 9010 INS 0x30: mauvais PIN
 9011 Le TESTBIT n'est pas à 0 : Instruction non supportée
 9013 INS 0x40/44: mauvaise clé, seules les MK' s sont autorisées
 9014 Instruction précédente 0x02 incorrecte sur une instruction 0x04
 9014 Instruction précédente 0x32/0x36 incorrecte sur une instruction 0x34/0x38
 9014 Instruction précédente 0x3C incorrecte sur une instruction 0x3A
 9014 Instruction précédente 0x06 incorrecte sur une instruction 0x40 NANO 0x87
 9014 Instruction précédente 0x5C incorrecte sur instruction 0x56/0x5A

9014 NANO précédente 0x1B incorrecte sur instruction 0x40/0x44 NANO 0xF6
 9014 NANO précédente 0x50 incorrecte sur instruction 0x44
 9015 INS 0x40, NANO 0x80: non autorisée
 9015 INS 0x3C, NANO 0x2C: non autorisée ou NANO 0x15: non autorisée
 9015 INS 0x32/0x36: donnée d'entrée non supportée
 9016 INS 0x40, NANO 0x87: décodage raté
 9016 INS 0x44: non autorisée
 9017 INS 0x50/0x54/0x56/0xAC/0x5A: non autorisée
 9018 Fournisseur existe déjà
 9019 INS 0x40 NANO 0x23 nouveau PROVIDER créé
 9019 INS 0x40 NANO 0x25 PROVIDER effacé
 9019 INS 0x40 NANO 0x41 nouveau PPUA affecté
 901A INS 0x3C, NANO 0x31/0x15/0x2C: achat via Jetons raté
 901B INS 0x40, NANO 0x15: aucun Jetons disponibles ou aucun Crédit Record disponible
 901C INS 0x3C, NANO 0x15/0x2C: achat via Jetons ok
 901D Clé Primaire inexistante
 901E PPV crédit hors limité INS 0x40 NANO 0x43
 901F Clé Secondaire inexistante
 9020 INS 0x44 NANO 0x90/0x91 erreur de clés
 9021 INS 0x04: la clé n'est pas 0x0F – refus
 9022 Le bit 7 de l'index de la clé est sur 1 (INS 0x40 en mode ART 2)
 9023 INS 0x44 NANO 71 doit être avant la NANO 0xD1
 9024 Erreur de Checksum, Instruction pas permise en ATR niveau 3
 9024 INS 38/3C/40 paramètre référence octet erroné (SE/SSE FLAGS)
 9025 INS 0x44 NANO 0xD1 le compteur de record est à zéro
 9026 INS 0x3C, NANO 0x31: plus aucun PREVIEW
 9027 INS 0x3C, NANO 0x19: phase de PREVIEW décryptée
 9028 Checksum des clés incorrects
 9028 INS B4 CHECKSUM différent de celui calculé, erreur phase initialisation de-enveloppe
 9029 INS 0x44, NANO 0xF6 erreur de checksum
 902B INS 0x44 NANO 50/92 erreur d'exécution de la NANO
 902C INS 0x40 NANO 50/92 paramètre 1 en dehors de la fourchette
 902D INS 0x40 NANO 50/92 erreur dans les indicateurs 8036-38h
 902E INS 0x40 NANO 50/92 erreur des derniers paramètres
 902F INS 0x40 NANO F7 NANO précédente n'est pas 0x50
 9030 INS 0x40 NANO 20/92 erreur ALGO utilisé
 9031 INS 0x40 NANO 0x1B erreur (bit 5 pas positionné)
 9031 INS 0x40 NANO 0xE0 erreur de vérification table HASH
 9031 INS 0xAC erreur de P2
 9031 INS 0x44 NANO 0x1B (bit 5 pas positionné)
 9031 INS 0x44 NANO E0 erreur en phase de vérification table HASH
 9033 INS 0x40 NANO 1B erreur (Index table d'ALGO pas à 59h)
 9033 INS 0x44 NANO 90/91 erreur table d'ALGO
 9033 INS 0x44 NANO 1B (Index table d'ALGO pas à DBH)
 9033 Ins 0x44 NANO E0 erreur
 9034 Erreur ENCRYPTION / DECRYPTION ALGO pas actif
 9035 Erreur SUPERENCRYPTION obligatoire INS 0x36/0x38, 0x3C, 0x40)
 9036 INS 38/3C/40 : paramètre /référence octet erroné (P4 bit 0,1)
 9037 INS 38/3C/40 Erreur Contrôle sur P5, P5 > que la longueur maximale(**Uniquement sur 7.0**)
 9038 Erreur de Contrôle sur P5, P5 = 00
 904A INS 3C : clé erronée , seul les clés OPERATIVES sont admises
 9090 EEPROM pas mise à jour (INS 0x40)
 90A0 EEPROM mise à jour INS 0x40)
 9301 INS 0x40, NANO 0x22/0x27: la date de la carte est au-delà de la date de validité
 9302 INS 0x3C, NANO 0xD1: aucun décodage
 9304 INS 0x3C, NANO 0x12: contrôle parental, niveau trop bas
 9305 INS 0x3C, NANO 0xF0: non valide pour cette zone géographique
 9401 INS 0x32: Valeur de P1 incorrecte
 9401 INS 0x36: Valeur de P1 incorrecte
 9402 INS 0x0C valeur de P2 incorrecte

9402 INS 0x32: Valeur de P2 incorrecte : seule valeur admise : P2 = 0
 9402 INS 0x30: mauvaise valeur de P2.
 9402 INS 0x5A pas accepté en mode ATR niveau 2
 9600 INS 0C P2=1 NANO 0x23-0x90 absente
 9600 INS 0x38: Chaîne d'entrée incorrecte (NANO incorrectes, signature manquante)
 9600 INS 0x3C NANO 82 problème
 9600 INS 0x3C: NANO 0x15/0x2C: résultat égal à 0- NANO traitées avec succès – aucun décodage
 960A Index Clés non supportés
 96xx INS 0x3C: XX NANO traitées, mais aucune NANO 0xD1 trouvée
 96xx INS 0x40: Toutes les NANO ont été traitées avec succès , mais erreur de PARSING
 97xx Mise à jour de l'EEPROM pas nécessaire pour quelque NANO XX est le BITMAP de ces NANO, par exemple 3C: aucun UPDATE pour NANO 2, 3, 4, 5 (le comptage commence à 0)
 98xx INS 0x40 NANO 0x32 événement nul XX NANO commande rencontré
 9A00 INS 56 Clé incorrecte
 9A00 INS 40/3C NANO non habilitée
 9Axx INS 44 NANO fermé (XX NANO commande rencontré)
 9B00 INS pas admise en ATR protection LEVEL (carte bloquée).
 99xx INS 0x40 NANO 0x42 et 0x30 date erronée: les crédits ont été changés aujourd'hui
 99xx date erronée

Analyse du PRE-PARSING - statut 9600

INS C1 40

Le PRE-PARSING pour l'instruction C1 40 fonctionne exactement comme nous l'avons décrite précédemment, c'est à dire la carte examine les NANOCOMMANDE présentes dans l'instruction, calculant la position de la NANO suivante en fonction de la NANO considérée à ce moment. A un certain point on arrivera sur la NANO 82 de signature ou on la dépassera. Dans le premier cas le contrôle est positif, dans le second cas on obtiendra le statut 9600. Il est à noter qu'il n'est pas nécessaire que la NANO rencontrée soit significative (c'est à dire qu'elle ai une fonction effective).

INS C1 3C – Le FUNNY BUG 2

De même pour l'INS 3C, le PRE-PARSING fonctionne comme pour l'instruction 40, au départ on avait fait l'hypothèse de la présence du FUNNY-BUG, avec un mécanisme similaire à celui présent dans les versions précédentes de la carte (c'est à dire en cas de dépassement de la signature en phase de PARSING).

FUNNY BUG (CITATION SECA1)

Nous nous trouvons à la conclusion du procédé d'examen des NANO relatives a l'instruction 3C , il s'agit, en particulier, de faire la somme du nombre d'octet de la dernière NANO au nombre total d'octets déjà comptabilisés et de vérifier si nous avons rejoint ou pas la longueur (LEN) déclaré de l'instruction.

En observant la ROM, nous notons que cette comparaison existe, mais contrairement au fonctionnement d'un algorithme correct, il ne suis pas une routine de contrôle, mais passe directement à la phase de fermeture (a la différence le l'INS C1 40). En effet il suit la copie du BUFFER des enregistrements en EEPROM et le masquage de la mémoire tampon des données. C'est justement le passage directe à cette phase de fermeture qui active le BUG : il en découle l'écriture de l'enregistrement en mémoire tampon (RECORD BUFFER) dans le premier enregistrement (RECORD).

Habituellement on trouve dans le RECORD BUFFER, la clé utilisé dans l'instruction, alors que les registres qui pointent au numéro de record sont réinitialisés, l'adressant au premier. On procède ainsi d'une NANO inconnue de tel sorte qu'on provoque un dépassement de la longueur totale de l'instruction, la clé ou éventuellement n'importe quel structure présente dans la mémoire tampon (BUFFER) est copiée sur l'enregistrement 1 de l'EEPROM.

En réalité la cause est différente.

FUNNY BUG SECA2 (CITATION)

Le FUNNY BUG se déclenche uniquement et exclusivement pour P3 de tel façon à sauter la signature.

Ce bug a été cité pour être complet, bien que aucune preuves sure de son existence n'ai été publiés.

INS 38 LE BUG 38/36

Ici nous observons un BUG macroscopique, qui à été prestement corrigé sur les version 7.1 : la routine de contrôle de la vieille C1 38 à été enlevée et substituée de la même routine de PRE-PARSING de l'INS C1 40.

En pratique cela veut dire que le contrôle sur le PRE-PARSING est dépassé si les données sont organisées selon les critères de l'instruction C1 40 et pas de l'INS C1 38.

Voyons comment fonctionnait le PRE-PARSING de la C1 38 en SECA 1

Instructions 38 et 36 – lecture des RECORDS et signature

Ce couple d'instructions fait une lecture des enregistrements (RECORDS) présents sur la carte, avec la gestion de la signature soit sur l'instruction envoyée, soit sur la réponse générale. De plus elle permet de modifier certains octets des enregistrement BX éventuellement présents.

On exécute d'abord la C1 38 (envoi de données) puis la C1 36 (demande de données)

C1 38 P1 P2 1A 16 xx 2A mm mm 2B nn nn 86 K0 K1 K2 K3 K4 K5 K6 K7 82 SIGNATURE

Les valeur **16**, **2A**, **2B**, **86** ne sont pas de vrais NANO,,il est impératif qu'ils soient dans la position indiquée, autrement nous aurons une erreur 9600 (voici comment fonctionnait le PRE-PARSING de l'instruction C1 38 en SECA 1, il s'agissait d'un contrôle sur la présence des pseudo – NANO dans des positions bien précises)

P1 et P2 ont la même signification que dans l'instruction C1 40, on peut utiliser une PK (PRIMARY KEY) ou une PK+SK (PRIMARY KEY + SECONDARY KEY) il est aussi possible d'utiliser la SUPER-ENCRYPTION.

L a valeur XX indique le type de record (enregistrement) à dumper, les octets mm se comportant de manière analogue aux données de l'instruction C1 34.

00 vv vv	PROVIDER PACKAGE BITMAP RECORD	
01 vv vv	PROVIDER PPV CREDIT RECORD	vv vv valeurs au choix
03 xx xx	PROVIDER PPV RECORD	xx xx contient EventID de la transmission PPV
04 vv yy	RECORD EX (SECA RECORD)	yy spécifie le type de record EX à trouver
06 zz zz	RECORDS GENERIQUES	zz zz numéro du record par lequel on débute

Attention : dans le cas ou nous avons 'xx' = 03 , en plus de montrer les PVV record, l'instruction va modifier le PPV-EVENT spot, c'est à dire le 10eme et le 11eme octet (en comptant par la gauche) de l'enregistrement traité, insérant les deux valeurs suivant 2B (nn nn). Après nous avons la valeur 86 suivit de 8 octets.

Le dump réel de la carte est exécuter avec l'instruction C1 36

C1 36 P1 P2 LEN

P1 doit être = à P1 de l'instruction précédente avec en plus le 6eme bit mit à 1.

P2 indique comme toujours la clé. Si le bit le plus significatif est à 1, la réponse de la carte est cryptée. LEN doit être supérieure à 13 (hexadécimal).

L'exécution de cette instruction va à son terme seulement si elle est précédée d'une instruction C1 38. La réponse à l'instruction C1 36 à la forme suivante : le premier octet c'est la longueur de la réponse, nous avons après la valeur 86 suivie des 8 octets avec l'instruction C1 38, enfin nous avons le dump des enregistrements selon la même signification de l'instruction C1 32 suivie de la valeur 82 et de la signature calculée sur la réponse.

Il est important de noter que c'est la seule instruction qui donne une réponse avec signature.

En SECA2 il n'est plus nécessaire d'avoir les valeurs **16, 2A, 2C, 86** d'où le bug 38/36.
En effet il suffit de suivre les canons de l'instruction C1 40 et de remplacer cette C1 40 par un C1 38.

C1 **40** 01 B1 5C
10 01 7E 3C 29 03 1B AC 43 31 82 A1 7E AE C4 26
96 F2 3E 0A C3 9E 76 3E C6 20 86 79 2E 4A 8D D9
18 14 95 3B 2E B6 54 D2 89 5F 76 0B 23 3B 63 4D
BF 00 EA 81 E5 24 BB 93 CA D4 D0 6F F8 38 5F 9C
5B EF AB 11 33 9B 17 8A EC CC 1D 6B 90 5D CD 2F
50 2C 90 A7 BE 29 68 37 7C 56 6F 33 **[90 00]**

Dans la C1 40 de cet exemple la C1 40 donne toujours (**90 00**) en réalité il existe une multitude de C1 40 avec des statuts divers qui sont aussi utilisable pour le BUG.

Les plus courants sont :

97 XY
90 19
93 01
90 09

En changeant le 40 en 38

C1 **38** 01 B1 5C
10 01 7E 3C 29 03 1B AC 43 31 82 A1 7E AE C4 26
96 F2 3E 0A C3 9E 76 3E C6 20 86 79 2E 4A 8D D9
18 14 95 3B 2E B6 54 D2 89 5F 76 0B 23 3B 63 4D
BF 00 EA 81 E5 24 BB 93 CA D4 D0 6F F8 38 5F 9C
5B EF AB 11 33 9B 17 8A EC CC 1D 6B 90 5D CD 2F
50 2C 90 A7 BE 29 68 37 7C 56 6F 33 **[90 00]**

Attention : si vous essayez de faire la même chose avec une INS 3C, vous n'obtiendrez pas un résultat identique

C1 **3C** 01 BD 5C
10 01 E6 12 CC 77 60 AE 96 FF F4 4D 58 03 99 F1
3F EF F3 A4 44 E6 1B 16 18 21 EB 40 D4 65 BC F5
66 60 6D 7D 54 29 28 06 52 95 29 D6 07 E0 11 23
1C 8E 89 29 89 2A 43 4E 11 98 2A 63 35 DB 56 F4
4B 03 82 B2 66 29 69 80 69 50 68 FC EC 0B 2E 3B
F2 A7 BC 04 CA A8 15 D9 60 7D 28 47 **[90 00]**

C1 **38** 01 BD 5C
10 01 E6 12 CC 77 60 AE 96 FF F4 4D 58 03 99 F1
3F EF F3 A4 44 E6 1B 16 18 21 EB 40 D4 65 BC F5
66 60 6D 7D 54 29 28 06 52 95 29 D6 07 E0 11 23
1C 8E 89 29 89 2A 43 4E 11 98 2A 63 35 DB 56 F4
4B 03 82 B2 66 29 69 80 69 50 68 FC EC 0B 2E 3B
F2 A7 BC 04 CA A8 15 D9 60 7D 28 47 **[90 02]** ou **[90 37]**

Conclusion : à parité d'octets cryptés, le de-enveloppe de l'instruction C1 3C donne un résultat différent de celui des instruction 40/38.

Il se pourrait que la clé utilisée soit différente ou encore l'initialisation de l'algorithme ou encore une autre gestion du paramètre P5 qui lui de toute façon se trouve dans la même position.

Revenons à notre C1 38 avec comme statut 90 00, ce résultat nous permet de lancer avec succès une C1 36 (autrement on aurait un statut 90 14).

C1 36 P1 P2 LEN

Analysons les paramètres de cette instruction :

P1 doit être égal à **2y** ou **3y** où **“y”** doit être égal au quartet (LOW NIBBLE) faible de P1 de la C1 38 précédemment envoyée, si cette condition n'est pas remplie nous aurons un statut de 94 01

P2 indique la clé et la table HASH à utiliser pour crypter la réponse (même signification que la 3C/38/40)

LEN doit être supérieure à 13 (hexadécimal) sinon nous obtenons une statut de 67 00

La réponse que nous obtenons dépend du corps de l'instruction C1 38. Cette dernière, si envoyée en suivant rigoureusement la syntaxe requise, aurait cette forme (après que la carte a décrypter l'instruction)

C1 38 P1 P2 LEN

P3 + **données de P3** + 16 **d1** 2A **d2 d3** 2B **d4 d5**
86 **b0 b1 b2 b3 b4 b5 b6 b7** 82 s0 s1 s2 s3 s4 s5
s6 s7 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 **P5**

Je vous rappelant qu'à cause du BUG, la carte ne contrôle pas la présence des NANO, mais prend directement **“d1”**, **“ d2 d3”**, **“ d4 d5”**, **“b0... b7”**, il en résulte un prélèvement des paramètres selon le critère suivant :

En comptant les octets à partir de celui qui suit immédiatement les données de P3

Le second pour d1
Le quatrième pour d2
Le cinquième pour d3
Le septième pour d4
Le huitième pour d5

Du dixième au dix-septième compris pour b0.....b7

Exemple :

```
C1 40 01 B0 61
10 01 92 BD 1F C2 E6 A6 C0 8B 1E F7 02 1E 7D F0
07 F2 31 2B 31 46 9E E9 1F 28 65 0D 12 68 F6 F1
25 B2 91 66 63 C0 EA 48 7D E3 1F EB E0 99 92 37
26 86 DB 0E C6 92 13 CD 61 E7 4C 70 45 27 2F A6
D9 B2 74 0C DF 7C E4 07 5D 62 B1 DD B0 13 F5 8C
9E 2A F2 28 12 55 1E 8D 76 43 19 31 01 6E B1 92
11
```

On les transforme en C1 38

```
C1 38 01 B0 61
10 01 92 BD 1F C2 E6 A6 C0 8B 1E F7 02 1E 7D F0
07 F2 31 2B 31 46 9E E9 1F 28 65 0D 12 68 F6 F1
25 B2 91 66 63 C0 EA 48 7D E3 1F EB E0 99 92 37
26 86 DB 0E C6 92 13 CD 61 E7 4C 70 45 27 2F A6
D9 B2 74 0C DF 7C E4 07 5D 62 B1 DD B0 13 F5 8C
9E 2A F2 28 12 55 1E 8D 76 43 19 31 01 6E B1 92
11 [90 00]
```

Les données sont prélevées après les avoir mis en clair (c'est à dire après les avoir décryptés).

Faisons l'hypothèse que le décryptage soit le suivant :

```
10 01 10 02 17 00 10 03 80 00 00 00 00 00 22 00
80 90 51 DE E2 04 D1 AF B3 FB B6 91 51 A1 37 9E
4B D0 49 4C 51 21 1A 7F 90 5C 75 13 DE 05 65 03
C4 57 B0 66 75 63 6B 20 26 20 73 75 63 6B 90 5D
9D 33 0E A2 6C 2D 3C 05 90 5E 06 73 8A B7 5D 9A
4C 5C 41 09 C3 22 21 82 D3 E8 54 CA 78 CD D2 61
P5
```

On comprend immédiatement quel sont les octets prélevés (en jaune) et utilisé pour la C1 36 qui suit. Elle peut générer une réponse cryptée ou en clair, dont la longueur et la signification varient selon la valeur qui est attribué à **d1**

Réponse avec d1 = 00 PROVIDER BITMAP PACKAGE RECORD

```
C1 38 01 B1 5C
10 01 7E 3C 29 03 1B AC 43 31 82 A1 7E AE C4 26
96 F2 3E 0A C3 9E 76 3E C6 20 86 79 2E 4A 8D D9
18 14 95 3B 2E B6 54 D2 89 5F 76 0B 23 3B 63 4D
BF 00 EA 81 E5 24 BB 93 CA D4 D0 6F F8 38 5F 9C
5B EF AB 11 33 9B 17 8A EC CC 1D 6B 90 5D CD 2F
50 2C 90 A7 BE 29 68 37 7C 56 6F 33 [90 00]
```

C1 36 21 00 LEN

36

86 b0 b1 b2 b3 b4 b5 b6 b7

83 yy yy yy yy yy yy yy yy

04 82 + octets à FF [90 35]

Dans ce cas la réponse est claire, sont présents quelques paramètres caractéristiques (qu'on appelle PSEUDO-NANO)

PSEUDO-NANO 86 est suivie des octets b0....b7 de l'instruction 38

PSEUDO-NANO 83 précède le BITMAP PACKAGE pour le PROVIDER indiqué par le quartet faible de P1

PSEUDO-NANO 04 indique qu'il n'y a pas d'informations ultérieures

NANO 82 c'est la NANO de la signature (si la réponse est en clair, la signature n'est pas présente)

Puis sont transmis autant d'octet à FF que nécessaire pour arriver à la juste longueur de la réponse.

```
C1 38 01 B1 5C
10 01 7E 3C 29 03 1B AC 43 31 82 A1 7E AE C4 26
96 F2 3E 0A C3 9E 76 3E C6 20 86 79 2E 4A 8D D9
18 14 95 3B 2E B6 54 D2 89 5F 76 0B 23 3B 63 4D
BF 00 EA 81 E5 24 BB 93 CA D4 D0 6F F8 38 5F 9C
5B EF AB 11 33 9B 17 8A EC CC 1D 6B 90 5D CD 2F
50 2C 90 A7 BE 29 68 37 7C 56 6F 33 [90 00]
```

C1 36 21 90 LEN

36

1C 2A 06 F8 58 E1 46 E3

B1 C6 53 51 6E 92 56 21

46 DE 4A F3 **82** 0E 0F F1

E4 1E 70 F0 7E ... [90 00]

Ceci est une réponse cryptée, seul la NANO 82 suivie de la signature est reconnaissable.

De plus il y a un octet supplémentaire par rapport à la C1 36 en claire, c'est celui qui est mis en évidence en jaune (1C hexadécimal, 28 en décimal).

Cette valeur indique combien de données significatives sont présentes dans la réponse.

Ces données sont : toutes celles qui le suivent jusqu'à la fin de la signature comprise.

Réponse avec d1 = 01 PROVIDER PPV CREDIT RECORD

C1 38 01 B1 5C
10 01 E9 2B BD 29 C1 89 4A DC 74 40 E8 8B 62 27
04 85 DD 6B 0A 8B AB 72 5A D5 18 36 5A 7B 1F AD
D4 89 C9 D7 CC 1A E9 94 35 8B 0F 10 96 09 9C DD
89 AB F7 CA F4 2A 41 87 32 35 7C 70 C7 3B DA 33
16 2D BB 2B DF C2 A2 4B 86 F5 92 BC C5 B3 92 A7
FC A7 5B 62 22 8E 42 12 DE 5D EE 25 [90 00]

En demandant une réponse en clair à la C1 36

C1 36 21 00 LEN

36

86 b0 b1 b2 b3 b4 b5 b6 b7

83 yy yy yy yy yy yy yy yy

04 82 + octets à FF [90 35]

PSEUDO-NANO 86 est suivie des octets b0.....b7 de l'instruction 38

PSEUDO-NANO 83 précède le CREDIT RECORD pour le PROVIDER indiqué par le quartet faible de P1

PSEUDO-NANO 04 indique qu'il n'y a pas d'informations ultérieures

NANO 82 c'est la NANO de la signature (si la réponse est en clair, la signature n'est pas présente)

Puis sont transmis autant d'octets à FF que nécessaire pour arriver à la juste longueur de la réponse.

C1 38 01 B1 5C
10 01 E9 2B BD 29 C1 89 4A DC 74 40 E8 8B 62 27
04 85 DD 6B 0A 8B AB 72 5A D5 18 36 5A 7B 1F AD
D4 89 C9 D7 CC 1A E9 94 35 8B 0F 10 96 09 9C DD
89 AB F7 CA F4 2A 41 87 32 35 7C 70 C7 3B DA 33
16 2D BB 2B DF C2 A2 4B 86 F5 92 BC C5 B3 92 A7
FC A7 5B 62 22 8E 42 12 DE 5D EE 25 [90 00]

Demandons une réponse cryptée

C1 36 21 90 LEN

36

1C 7B AA 06 E3 11 D4 D3

24 83 7B AD 4A 16 C7 1C

CD A8 A9 B8 **82** 19 58 38

84 ED 84 B1 F2 ... [90 00]

Ceci est une réponse cryptée, seul la NANO 82 suivie de la signature est reconnaissable.

De plus il y a un octet supplémentaire par rapport à la C1 36 en claire, c'est celui qui est mis en évidence en jaune (1C hexadécimal, 28 en décimal).

Cette valeur indique combien de données significatives sont présentes dans la réponse.

Ces données sont toutes celles qui le suivent jusqu'à la fin de la signature comprise.

De l'examen de la réponse cryptée il est impossible de distinguer entre le cas où d1 = 00 et d1 = 01

Réponse avec d1 = 03 PROVIDER PPV RECORD

C1 38 01 B1 5C
10 01 37 FA 7D D3 E0 B2 3B 1C 9B 04 D8 FF C1 38
8D C6 5C E1 38 7F FD C5 84 35 A8 1E D4 9C 25 48
C1 76 9C 4D A8 21 DE 66 E0 C8 CC 29 AB 50 8D 20
F5 AD 61 13 2C 5E A2 E3 56 5B F8 33 5A FA FC AD
97 85 84 49 C1 56 10 AD 4B 9C C1 9F BB 70 B5 CC
67 69 61 9C 70 34 3E 90 4E 39 2E 10 [90 00]

Record Bx : PPV RECORD (citation et adaptation)

ty ID ID dn cv ec ec dd dd sp sp Bx

Le record Bx est créé au moment de l'achat d'un EVENT PPV. En pratique, après une demande de données, le décodeur vérifie la présence de cet enregistrement pour un PPV EVENT ID, s'il est présent on peut visionner la transmission PPV.

Le premier octet signifie le **type 00 par achat et 01 par jeton**, les deux suivants indiquent l'**EVENT ID** auquel se réfère l'enregistrement, l'octet indiqué par **dn** est le numéro de diffusion, si par exemple le même EVENT est transmis sur plusieurs canaux, le PPV EVENT ID est le même pour tous les canaux, tandis que le numéro de diffusion change. Le numéro de diffusion change aussi entre deux émissions successives de l'EVENT sur le même canal. Il sert donc pour indiquer l'EVENT temporel et spatial. A chaque incrémentation du numéro de diffusion, correspond une diminution du nombre de visions encore disponibles. Si l'EVENT n'a pas encore été regardé, il assume la valeur de FF. L'octet **cv** est le nombre de visions encore disponibles. Suivent deux octets **ec ec** (jetons associés à l'événement, on les ajoute à travers INS 3C – NANO 15), puis les deux octets **dd** qui indiquent la date de la première vision, si l'EVENT n'a pas encore été vu, ils prennent la valeur de FF, enfin **sp** vous avez le PPV EVENT-SPOT qui normalement est à 00 et qui est modifiable avec les instructions 36/38.

Exemple :

Supposons avoir un EVENT PPV à 0E 38

00 0E 38 FF 04 00 00 FF FF 00 00 B1

Quand le film a été acheté mais pas encore vu

00 0E 38 0F 04 00 00 16 A1 00 00 B1

Quand le film a été vu au moins une fois

0E 38 = PPV-EVENT ID

0F = Numéro de diffusion / **FF** aucun numéro de diffusion a été fourni

04 = Nombre de diffusion disponible

16 A1 = Date de la première vision / FF pas encore vu

Dans la c1 38 est indiquée l'EVENT ID à rechercher, à travers les octets **d2 d3**, tandis que les données **d4 d5** représentent le PPV EVENT-SPOT. Si dans l'instruction, on spécifie un EVENT ID qui n'a pas été mémorisé dans la carte, il sera remplacé par la valeur hexadécimale immédiatement supérieure à celle spécifiée, en conséquence l'EVENT ID 00 00 va dans tous les cas retrouver un PPV EVENT mémorisé.

Si un EVENT est trouvé, nous aurons la réponse suivante en clair :

C1 36 21 00 LEN

36

86 b0 b1 b2 b3 b4 b5 b6 b7

B1 YY YY YY YY YY YY YY YY YY YY YY

04 82 + les Octets FF [90 35]

PSEUDO-NANO 86 est suivie des octets b0.....b7 de l'instruction C1 38

PSEUDO-NANO B1 précède le PPV record

PSEUDO-NANO 04 indique qu'il n'y a pas de données ultérieures

NANO 82 c'est celle de la signature (si la réponse est en clair, la signature n'est pas présente)

Puis sont transmis autant d'octets que nécessaire, à FF, pour arriver à la longueur (LEN) requise.

Réponse cryptée :

C1 36 21 90 LEN

36

1F 69 FE 5E 15 DC F3 62

AF FA DD 5D 07 06 55 A0

3C C8 58 28 45 AD 33 **82**

A5 6F DE 72 9F 99 D4 70...

[97 04]

la réponse est plus longue que dans les cas précédents (1F = 31 décimal). S'il y a plusieurs PPV record présents sur la carte, il est possible de faire une demande multiple avec une seule C1 36, il suffit d'utiliser des valeurs suffisamment importantes pour la longueur (LEN).

Il est nécessaire, à ce point de commenter le statut obtenu, parce qu'il est indicatif d'une tentative d'écriture dans l'EEPROM.

PPV-SPOT (citation)

Dans le cas où vous avez **d1** égal à 03, outre à monter le PPV RECORD, l'instruction va modifier le PPV-EVENT SPOT, c'est à dire le dixième et le onzième octet (en comptant par la gauche) de l'enregistrement traité, en insérant les valeurs **d4 d5**

Réponse avec d1 = 04 : Record Ex

Contrairement à ce qui a été écrit dans beaucoup de documents, il n'est pas nécessaire d'adresser cette instruction au PROVIDER 00 (SECA) pour avoir cette réponse.

```
C1 38 01 B1 5C
10 01 B0 DF 71 ED 4E 3E 40 A0 CA B4 34 9F 49 84
B2 22 56 D6 AD 5C 31 2D AE 51 AE CC F6 97 F1 10
5D 56 D8 EA CB B0 F4 5B D5 01 1B 40 01 C3 CC 7B
2B F7 26 44 BE D6 04 14 95 FB F6 5B 6C 5B BC C1
51 54 5B A0 7A C3 47 78 C4 EB 5D EC 3C A7 A3 CF
1F D5 32 87 A6 D0 C9 DD CD 8D 1B 01 [90 00]
```

Selon la valeur de **d3**, on sélectionne l'enregistrement Ex, pour lequel les données nous intéressent (d3 doit être égal au premier octet de l'enregistrement).

Demande en clair :

```
C1 36 21 00 LEN
```

```
36
86 b0 b1 b2 b3 b4 b5 b6 b7
B2 YY YY YY YY YY YY YY YY YY YY YY
04 82 + les octets FF [90 35]
```

PSEUDO-NANO 86 est suivie des octets b0....b7 de l'instruction C1 38

PSEUDO-NANO B2 précède le SECA RECORD (les premiers 0x0B octet d'une record E0)

PSEUDO-NANO 04 indique qu'il n'y a pas d'informations ultérieures

NANO 82 c'est la NANO de signature (si la réponse est en clair, la signature n'est pas présente)

Réponse cryptée :

```
C1 36 21 90 LEN
```

```
36
1F 22 ED 88 F1 77 36 13
00 CD EB 42 42 4C 95 69
DE 8D 13 B1 AB 48 5A 82
D4 17 33 69 FE 5E 15 32...
[90 00]
```

Il faut noter que la longueur de la réponse est la même que celle de **d1** = 03, il est cependant possible de distinguer les deux cas en observant l'octet de statut qui dans ce cas aura toujours la valeur 90 00

Réponse avec d1 = 06 Records génériques

Dans ce cas on va lire les enregistrements génériques, c'est à dire dans le format dans lequel ils sont mémorisés dans la carte (lisible aussi avec les instruction 34/32). L'enregistrement visualisé est le premier qui contient des données relatives au fournisseur d'accès indiqué par le quartet faible de P1, la recherche de l'enregistrement démarre de celui spécifié dans l'instruction C1 38 à travers les octets **d2d3**.

```
C1 38 01 B1 5C
10 01 00 4C 0D F3 5C BE 9E 54 66 A7 8D 33 49 BF
DC C0 30 ED 74 8B 49 64 83 6F F0 0F F5 FC 33 2E
C3 C9 86 B4 59 27 D8 93 36 7C 9D C7 C6 34 82 8B
43 7F 4A 20 43 36 28 D7 45 CA 38 69 FA 10 11 94
FE 9E A8 C3 4A 5E 7A C1 3A B0 61 3C E9 2B 80 98
43 90 81 59 CA E2 9F 68 1E 2B 2C 47 [90 00]
```

Réponse en clair :

C1 36 21 00 LEN

[illegible]

PSEUDO-NANO 86 est suivie des octets b0....b7 de l’instruction C1 38
PSEUDO-NANO D2 précède le RECORD
Les données associées à D2 sont organisé de la façon suivantes :

OCTETS	Nombre Décimal	Organisation
jj	2 (0x02 HEX)	Utilisé pour montrer l'index de l'enregistrement
YY	12 (0x0C HEX)	Indique le contenu de l'enregistrement
00	2 (0x02 HEX)	Egal à 00

PSEUDO-NANO 03 indique qu'il n'y a de possible informations ultérieures.
NANO 82 c'est la NANO de signature (si la réponse est en clair, la signature n'est pas présente).

Réponse cryptée :

C1 36 21 90 LEN

```

36
24 1D 02 62 22 02 76 60
BC E1 C5 7A DE CA 5C 3B
77 9F CC F0 E5 7B 7C 79
A1 44 D4 01 82 B4 A0 41
40 49 4D 5A 97 ... [90 00].

```

Il est possible de faire un dump de la carte avec une simple C1 36, il suffit d'utiliser une LEN qui soit assez longue

Le cas le plus fréquent une réponse courte

Il arrive souvent qu'en réponse à une C1 36, on obtienne quelque chose dans ce genre :

C1 36 21 90 LEN

36

13 A2 B1 24 F9 81 F9 33
C3 DC B7 82 DF 72 51 44
2D AE 60 BA... [90 00]

On parle dans ce cas d'une réponse courte, c'est à dire avec un minimum de données significatives. Le contenu correspondant, en clair peut assumer la forme suivante :

Pas d'informations ultérieures

C1 36 21 00 LEN

36

86 b0 b1 b2 b3 b4 b5 b6 b7
04 82 + les octets FF [90 35]

Possible informations ultérieures

C1 36 21 00 LEN

36

86 b0 b1 b2 b3 b4 b5 b6 b7
03 82 + les octets FF [90 35]

La première réponse (PSEUDO-NANO **04**) s'obtient uniquement dans les conditions reprises dans ce tableau.

Valeur assumée par d1	Valeur assumée par la longueur (LEN) en HEX	Condition de contournement
00	Supérieure à 1D	BITMAP PACKAGE pas présent
01	Supérieure à 1D	CREDIT RECORD pas présent
03	Supérieure à 20	EVENT PPV avec ID supérieure ou égal à d2d3 non présent
04	Supérieure à 20	RECORD Ex avec le premier octet égal à d3 pas présent
06	Supérieure à 25	RECORD avec index supérieur ou égal à d2d3 non présent

La seconde réponse (PSEUDO-NANO **03**) s'obtient dans de diverses occasions. Une des façons c'est celle d'utiliser des valeurs de **d1** différentes de 00, 01, 03, 04, 06. Les autres modes sont indiqués dans le tableau suivant.

Valeur assumée par d1	Valeur assumée par la longueur (LEN) en HEX	Condition de contournement
00	Plus petit ou égal à 1D	BITMAP PACKAGE pas présent
01	Plus petit ou égal à 1D	CREDIT RECORD pas présent
03	Plus petit ou égal à 20	EVENT PPV avec ID supérieure ou égal à d2d3 non présent
04	Plus petit ou égal à 20	RECORD Ex avec le premier octet égal à d3 pas présent
06	Plus petit ou égal à 25	RECORD avec index supérieur ou égal à d2d3 non présent

Dans la description des réponses possibles de la C1 36, nous avons fait une distinction entre réponses cryptées et réponses en clairs, voyons maintenant comment générer ces dernières.

C1 36 en clair du type 1 – Flag SUPER-ENCRYPTION

Si le bit 7 de P2, qui indique la SUPER-ENCRYPTION est égal à 0, on a une condition d'erreur, parce que sur les 7.x le SUPER-ENCRYPTION est obligatoire, même pour les C1 36 et en effet la carte nous répond avec un statut de 90 35.

Cependant, avant le statut, elle nous offre gratuitement, la réponse en clair jusqu'à la NANO 82 (la signature étant calculée sur des données cryptées et en l'absence de celles-ci, nous trouvons après la NANO 82 uniquement des octets de remplissage égaux à FF).

Exemple :

```
C1 36 21 00 14
36 86 11 21 AA 2A 12 20
22 2F 03 82 FF FF FF FF
FF FF FF FF [90 35]
```

C1 36 en clair du type 2 – CRYPT avec USER ALGO

Si le bit 7 de P2 est correctement positionné, mais les bits 5,6 sont tel que P2 = Cx ou encore Dx, le processus de décryptage de la réponse ne peut avoir lieu (nous avons dit plus haut que le USER ALGO n'était pas présent sur les V7). La carte répond alors avec une réponse en clair suivie d'un statut de 9034

Exemple :

```
C1 36 21 D0 14
36 86 11 21 AA 2A 12 20
22 2F 03 82 FF FF FF FF
FF FF FF FF [90 34]
```

C1 36 en clair du type 3 – Bug sur la LEN

Il arrive que la carte réponde avec l'instruction en clair (même si nous ne l'avons pas demandé) suivie de la valeur du statut à 9015.

Ceci arrive lorsque on utilise d1 avec une valeur différente de 00, 01, 03, 04, 06 et qu'on dépasse une certaine longueur (LEN) dépendant du d1 spécifié.

Exemple :

```
C1 36 23 F0 14
36 86 11 21 AA 2A 12 20
22 30 FF FF FF FF FF FF
FF FF FF FF [90 15]
```

Ce comportement connu sous le nom de bug de la LEN permet le dump d'une petite partie de la ROM.

BUG sur la LEN (Citation)

Le bug présent sur les V7 est le descendant direct du bug sur la LEN présent avec la 34/32 de la V6.0 et inférieures. Le bug était le suivant :

C1 34 00 00 03 **XX** 00 00
C1 32 00 00 **YY**

Selon la valeur assumé par **XX**, on a une valeur minimum admissible pour **YY**, en fonction d'une table contenue dans la ROM. Nous savons que les valeurs standards pour **XX** sont 00, 01, 03, 04, 06 mais la ROM ne nous interdit pas d'en utiliser d'autres !.

Pour évaluer la longueur minimale, on utilise un tableau dont les éléments sont lus en additionnant à l'adresse base de la table, la valeur **XX**, sans aucun contrôle.

Si à **XX** je remplace par une valeur > 06, je vais lire indirectement un morceau de la ROM (en fait les octets suivant de la table), j'ai bien dis indirectement, car il faut le faire de cette façon :

C1 34 00 00 03 07 00 00
C1 32 00 00 **00**

C1 34 00 00 03 07 00 00
C1 32 00 00 **01**

C1 34 00 00 03 07 00 00
C1 32 00 00 **02**

Incrémenter la LEN de la C1 32 jusqu'à ce que la réponse de la carte change (**il faut arriver à obtenir un statut de 90 15**) quand cela advient, vous avez trouvé la valeur de l'octet (l'octet vaut **YY - 1**)

Sur la V7 le bug à été retiré des instructions 34/32 mais il est resté sur les instructions 38/36, on pourrait arrivé à dumper une partie de la ROM, mais il faut beaucoup de patience et un beau paquet d'instructions. La C1 38 de SECA1 avait a structure suivante :

C1 38 P1 P2 LL 16 **XX** 2A mm mm 2B nn nn 86 b0 b1 b2 b3 b4 b5 b6 b7 82 + SIG

Ou **XX** a la même signification de la C1 34, nous devons donc travailler sur l'analogie présente dans la commande SECA2.

La seule chose à se rappeler c'est que la C1 36 n'accepte comme longueur (LEN) minimale 0x14, car avec des valeurs inférieures, nous avons un statut de 67 00 et qu'à la différence des versions précédentes nous pouvons tranquillement dépasser la valeur de 0x5Fh.

Le calcul de la LEN minimale advient en prenant les données dans un table en ROM. A partir de là, la gestion de la valeur trouvée, suit deux chemins différents, selon qu'on a une INS 32 ou une INS 36, car cette dernière restitue en sortie les mêmes données que la C1 32, plus d'autres octets (...), au total 20 octets en plus (0x13 HEXA). Pour la C1 36, nous retombons sur la valeur présente en ROM en prenant la LEN qui donne comme statut 90 15 et en lui retranchant 0x14.

9015 n'est pas envisageable pour les valeurs normales (00,01,03,04,06) de **d1**.

Quand on a une C1 36 qui ne donne pas de 90 15 pour aucune LEN, cela veut dire que **d1** pointe à une valeur en ROM qui est supérieure ou égale à EC ou alors pour un des motifs vu dans les points précédents. Les deux sont normalement repérables en vérifiant ou pas la transition :
NANO 03 → NANO 04 à partir d'une certaine longueur.

Contenu de la ROM lue à travers le BUG de la LEN

[En clair :](#)

```
secarom70.bin
00h: 09 09 03 0C 0C 06 11 87 03 85 3A 00 15 09 1C 01 : .....#.!:.....
10h: 1C 01 04 3F 00 26 12 FF 06 01 5F 41 00 37 44 00 : ...?.s.y...A.7D.
20h: 36 58 41 00 04 58 41 00 04 37 44 00 08 51 41 00 : 6XA...XA...7D..QA.
30h: 04 51 41 00 04 51 41 00 02 51 41 00 02 37 44 00 : .CA...CA...CA...7D.
40h: 30 37 41 00 04 37 44 00 20 37 44 00 0C 37 44 00 : 07A...7D. 7D..7D.
50h: FF 02 07 37 44 00 FF 02 02 CA 44 02 9B 02 0C 92 : y...7D.y...ED.s...
60h: 02 04 8C 02 04 8F 02 16 89 02 02 86 02 02 76 02 : ..G..D...s...t...v.
70h: 02 83 02 02 7C 02 02 73 02 02 5B 02 04 FF 01 00 : .f...|...s...[...y...
80h: 02 00 02 11 01 10 1B 01 48 62 00 08 E3 00 12 C3 : .....Hb...ä...Ä
90h: 00 FF 12 03 17 4B 00 1F 09 04 05 04 03 70 01 52 : .y...K.....p.F
a0h: 01 48 01 24 03 31 05 04 06 0F 1C 06 42 06 4E 04 : .E$.1.....B.N.
b0h: 16 02 01 08 FF 01 01 B8 4E 00 36 4C 00 01 41 4C : ....y...N.6L...AL
c0h: 00 01 42 4B 00 09 B8 4E 00 01 B3 4E 00 03 B8 4B : ..BK...N...N...K
d0h: 00 01 8B 4C 00 01 B3 4E 00 02 B3 4E 00 03 3E 4B : ...L...N...N...>K
e0h: 00 04 B3 4E 00 01 59 4D 00 04 B3 4E 00 01 FF 4B : ...N...YM...N...yK
f0h: 00 01 E4 4B 00 01 B3 4E 00 01 B3 4E 00 01 B3 4E : ..äK...N...N...N

09 09 03 0C 0C 06 11 87 03 85 3A 00 15 09 1C 01
1C 01 04 3F 00 26 12 -- 06 01 5F 41 00 37 44 00
36 58 41 00 04 58 41 00 04 37 44 00 08 51 41 00
04 51 41 00 04 51 41 00 02 51 41 00 02 37 44 00
30 37 41 00 04 37 44 00 20 37 44 00 0C 37 44 00
-- 02 07 37 44 00 -- 02 02 CA 44 02 9B 02 0C 92
02 04 8C 02 04 8F 02 16 89 02 02 86 02 02 76 02
02 83 02 02 7C 02 02 73 02 02 5B 02 04 -- 01 08
02 00 02 11 01 10 1B 01 48 62 00 08 E3 00 12 C3
00 -- 12 03 17 4B 00 1F 09 04 05 04 03 70 01 52
01 48 01 24 03 31 05 04 06 0F 1C 06 42 06 4E 04
16 02 01 08 -- 01 01 B8 4E 00 36 4C 00 01 41 4C
00 01 42 4B 00 09 B8 4E 00 01 B3 4E 00 03 B8 4B
00 01 8B 4C 00 01 B3 4E 00 02 B3 4E 00 03 3E 4B
00 04 B3 4E 00 01 59 4D 00 04 B3 4E 00 01 -- 4B
00 01 E4 4B 00 01 B3 4E 00 01 B3 4E 00 01 B3 4E
```

Les octets dont la valeur est FF sont des valeurs inconnues car non DUMPABLE, d'eux nous pouvons juste dire que ce sont de octets égaux ou supérieures à EC ([voir page précédente](#)).

Construction d'une C1 38/40 « CUSTOM » à partir d'une EMM

Pour cet exercice il est nécessaire d'avoir le matériel suivant ou analogue

MARKTIMER (SmartTimer) 1.6 ou supérieur
EMM d'activation ou EMM de mise à jour des clés opératives

Phase 1 préparation de la commande initiale

Dans le cas où on dispose d'une EMM LOGGUE **PRECAM** celle-ci se présente de la façon suivante :

86	00	65	0000	06xxyyzz	0070	00	B0	0203	509392031..
----	----	----	------	----------	------	----	----	------	-------------

Sans trop entrer dans les détails sur la structure d'une EMM PRECAM, nous pouvons distinguer les parties les plus importantes et nécessaires à reconstruire l'équivalent destiné à la carte :

65 Longueur de la commande PRECAM

06xxyyzz PPUA à qui est adressée l'EMM

70 Le PROVIDER à qui est adressé l'EMM (dans ce cas PROVIDER **01** : par la relation PROVIDER ID ↔ Numéro de PROVIDER, voir le tableau)

B0 Clé avec laquelle est cryptée et signée l'EMM (dans ce cas MK 00)

TABLEAU DES PROVIDERS

Nationalité de la carte	NUMERO DU PROVIDER QUARTET FAIBLE P1 (LOW NIBBLE)				
	0	1	2	3	4
Italienne (ver : 7.0)	0000 SECA	070 T£L£+DIGIT@L£	071 +C@LCI0	072 STR£AM	073 P@LC0
Française (ver : 7.1)	0000 SECA	080 C@N@LSATELLIT£	081 C@NAL+	082 SPARE-A	ABSENTE
Polonaise (ver : 7.0)	0000 SECA	065 CYFR@+			ABSENTE
Espagnole (ver : 7.0)	0000 SECA	064 C@N@LSAT£LIT£	066 C@N@LSAT£LIT£ 2	067 C@N@LSAT£LIT£3	ABSENTE
Hollandaise (ver : 7.3)	0000 SECA	006A C@N@LDIGITAAL	006B Opérateur 2	006C Opérateur 3	006D Opérateur 4

Autres ID de fournisseurs d'accès existants : 0076 - 0084 - 0086 (PRO TV) - 0088 (AB S@T)

La conversion de l'EMM dans la forme équivalente destinée à la carte s'obtient de cette façon :
 Avant tout on calcule la longueur réelle de la commande, sur la base de la longueur de la commande PRECAM
 (Dans l'exemple ci-dessus 65) :

LEN = (Longueur de la commande PRECAM - 4)

Donc $65 - 4 = 61$ (nous avons la longueur de la trame)

Puis on compile l'instruction :

C1 38 01 B0 61 10 01 509392031...

Si nous avons un EMM LOGGUE, il suffira de changer l'instruction 40 en instruction 38, éliminer l'octet de statut et l'octet ACK (ACK = octet d'acquiescement ou de confirmation de l'instruction par exemple dans C1 40 le premier octet de la réponse sera l'ACK et donc 40, il ne fait pas partie de partie de la chaîne de réponse.
 Selon les programme utilisés, l'ACK peut ne pas être présent.

Exemple avec l'octet ACK :

C1 **40** 01 B0 61 **40** 10 01 58 65 55 88 99 15 22... .86 -> 97 FA
 C1 **38** 01 B0 61 10 01 58 65 55 88 99 15 22... .86

Exemple sans ACK :

C1 **40** 01 B0 61 10 01 58 65 55 88 99 15 22... .86 -> 97 FA
 C1 **38** 01 B0 61 10 01 58 65 55 88 99 15 22... .86

Nous pouvons maintenant envoyer l'instruction :

C1 38 01 B0 61
 10 01 58 65 55 88 99 15 22 D3 06 AA D2 F8 E6 19
 29 E3 5F DF 96 6A 1C 42 4F 94 0D B8 01 43 2F D0
 7E FB 90 DE 42 2B C7 14 A7 A8 E2 56 AD F6 10 56
 71 EB 38 6B 5B 6E 5E 0E CE 8B B9 BC 0E 30 94 60
 39 F2 AC 73 20 1B 6B 3E C5 CF 82 29 2C 14 73 E6
 23 F2 D7 99 21 7E 73 E3 90 27 40 A0 A5 16 55 C6
 86 [90 00]

A la suite de laquelle nous enverrons le complément à la 38, c'est à dire la 36. Celle-ci a la possibilité de restituer, soit sous une forme cryptée, soit sous une forme en clair, des données qui sont dépendantes de l'instruction 38 précédente, en fonction des clés et des tables de cryptage sélectionnés par les paramètres de P1 et P2.

A travers un petit jeu très simple de variation des paramètres de P1 et de P2, nous allons aller à la recherche d'une réponse qui satisfasse les conditions imposées par les nouveaux algorithmes de cryptage.

Phase 2 Exécution du cycle INS 38 et INS 36

Comme nous l'avons déjà suggéré, dans la description des instructions 36/38, il existe des chemins balisés pour l'exécution des commandes.

Paramètres de P1 et de P2 admis par l'instruction C1 36

Pour les valeurs admissibles on fait référence aux cas utiles dans la pratique, par un traitement rigoureux de la syntaxe, faites référence à la description des instructions (page 14)

P1 Quartet fort Peut assumer la valeur **2** pour toutes les valeurs du quartet faible de P2
Peut assumer la valeur **3** seulement pour les valeurs du quartet faible de P2 = à 0 et 1.

P1 Quartet faible **Doit** assumer la même valeur des **quartets faibles de P1 de la C1 38**

P2 Quartet fort Peut assumer les valeurs 9,B,F (Avec D, on ne peut pas avoir de réponse cryptée voir page 25, C1 36 réponse de type 2).

P2 Quartet faible Peut assumer les valeurs 0,1, (2,3), C,D,E) en tenant compte que C,D,E ne sont pas utilisables pour créer un EMM, mais seulement un ECM.

C'est à dire valeurs admissibles de P1 et P2 (Exemple avec **C1 38 01 B0 LL 10 01...**)

C1	36	21	90	LL	C1	36	21	B0	LL	C1	36	21	F0	LL
C1	36	21	91	LL	C1	36	21	B1	LL	C1	36	21	F1	LL
C1	36	21	9C	LL	C1	36	21	BC	LL	C1	36	21	FC	LL
C1	36	21	9D	LL	C1	36	21	BD	LL	C1	36	21	FD	LL
C1	36	21	9E	LL	C1	36	21	BE	LL	C1	36	21	FE	LL
C1	36	31	90	LL	C1	36	21	B0	LL	C1	36	31	F0	LL
C1	36	31	91	LL	C1	36	21	B1	LL	C1	36	31	F1	LL

Et dans la mesure où on en dispose

C1	36	31	92	LL	C1	36	31	B2	LL	C1	36	31	F2	LL
C1	36	31	93	LL	C1	36	31	B3	LL	C1	36	31	F3	LL

Le cycle d'envoi des instructions C138 + C1 36 sera donc :

Envoi de C1 38 P1 P2 LEN P3 P4 + DONNEES
Envoi de C1 36 P1 P2 LL

Pour la LL nous utiliserons 14
(HEX) de cette façon nous aurons
que des réponses courtes

Ce type de cycle se termine un fois que toutes les combinaisons possibles, en regard de l'instruction C1 36 ont été envoyées.

Les réponses reçues à chaque instruction seront notées, à la fin des tests, on utilisera que celles qui auront la forme imposées par les paramètres P3 et P4 de la nouvelle codification.

Formes admissibles de la réponse à une instruction C1 36 (réponse courte)

1) La forme de la réponse doit être obligatoirement cryptée

Une vérification rapide pour savoir si on a obtenu une réponse cryptée se fait à travers l'étude de l'octet de statut, si on obtient un 90 00, la réponse est certainement cryptée

2) les premiers deux octets de la réponse (qui deviendront P1 et P2 de la C1 38) doivent obéir aux règles suivantes :

Le quartet fort de P1 doit être supérieure à 1 et inférieure à 9

Le quartet faible de P2 doit être égal à 1 ou à 9

C'est à dire que les valeurs des réponses admissibles sont :

C1 36 P1 P2 14 (36) 13	1x x1	dd dd dd dd ...
C1 36 P1 P2 14 (36) 13	1x x9	dd dd dd dd ...
C1 36 P1 P2 14 (36) 13	2x x1	dd dd dd dd ...
C1 36 P1 P2 14 (36) 13	2x x9	dd dd dd dd ...
C1 36 P1 P2 14 (36) 13	3x x1	dd dd dd dd ...
C1 36 P1 P2 14 (36) 13	3x x9	dd dd dd dd ...
C1 36 P1 P2 14 (36) 13	4x x1	dd dd dd dd ...
C1 36 P1 P2 14 (36) 13	4x x9	dd dd dd dd ...
C1 36 P1 P2 14 (36) 13	5x x1	dd dd dd dd ...
C1 36 P1 P2 14 (36) 13	5x x9	dd dd dd dd ...
C1 36 P1 P2 14 (36) 13	6x x1	dd dd dd dd ...
C1 36 P1 P2 14 (36) 13	6x x9	dd dd dd dd ...
C1 36 P1 P2 14 (36) 13	7x x1	dd dd dd dd ...
C1 36 P1 P2 14 (36) 13	7x x9	dd dd dd dd ...
C1 36 P1 P2 14 (36) 13	8x x1	dd dd dd dd ...
C1 36 P1 P2 14 (36) 13	8x x9	dd dd dd dd

Phase 3 : construction d'une nouvelle instruction utilisateur et contournement de l'enveloppe

Dans le cas où nous obtenons une réponse valide, c'est à dire correspondant à celles définies en phase 2, nous pouvons créer une nouvelle instruction utilisateur, par exemple nous utiliserons une réponse du type :

```
C1 36 21 B0 14
36
13 65 31 D3 B2 97 A5 D2
86 81 DE 82 77 B6 85 8D
A9 00 C0 AD [90 00]
```

Éliminons l'octet ACK (36, si présent), la longueur de la réponse (LEN 14) et les octets de statut (90 00)

```
65 31 D3 B2 97 A5 D2 86 81 DE 82 77 B6 85 8D A9 00 C0 AD
```

Composons une nouvelle instruction :

```
C1 38 21 B0 LL 65 31 D3 B2 97 A5 D2 86 81 DE 82 77 B6 85 8D A9 00 C0 AD
```

Et ajoutons au moins 0x5A avec une valeur égale à 0x00 : ceux-ci représenteront les octets qui subiront le de-enveloppe, mais qui ne prendront pas part au calcul et à l'exécution de la signature, grâce à la possibilité de contournement de l'enveloppe, à travers la procédure expliquée un peu plus loin.

Il est conseillé de, surtout pour les essais que nous développerons plus loin, de démarrer avec une longueur de **LL 0x75**, il convient donc d'ajouter 0x62 octets de valeur égal à 0x00. De ces 62 octets, les premiers 8 post signature seront en clairs et en dehors de l'enveloppe, le reste 0x5A, seront sous enveloppe (voir la théorie sur le processus d'exécution des instructions)

A la fin nous aurons :

```
C1 38 21 B0 76
65 31 D3 B2 97 A5 D2 86 81 DE 82 77 B6 85 8D A9
00 C0 AD 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
```


Si nous envoyons cette instruction nous aurons statut de 9002 (il est possible que ce soit aussi un 9037 ou encore un 9038, cela dépend du fournisseur d'accès), parce que l'enveloppe n'a pas encore été contournée. **Le contournement s'obtient en changeant la valeur du dernier octet de l'instruction à envoyer**, jusqu'à obtenir un statut de 9600 au lieu d'un statut de 9002/9037/9038.

Vous avez compris, le 9600 arrive car la variation d'un octet se répercute sur le résultat de tout ce qui est sous de-enveloppe et en particulier la valeur assumé par le dernier octet **P5**. Lorsque nous avons la juste valeur, (de tel façon qu'il adresse correctement la NANO 82 de l'instruction que nous avons construit), alors nous avons réussi à contourner l'enveloppe. (En fait nous pourrions changer n'importe quel octet sous enveloppe (les derniers 0x5A), mais par commodité et maintenant par convention, nous utilisons le dernier octet). **Le contournement, en fait ; consiste à faire des variations de P5 (ou d'une autre valeur sous enveloppe) pour se positionner sur la NANO 82.**

Il n'existe pas toujours une valeur pour le dernier octet qui puisse contourner l'enveloppe, alors on fait varier **aussi** l'avant dernier octet.

A la page 34 vous trouverez un tableau qui indique les justes valeurs à assigner au dernier et à l'avant dernier octet (NLB/LB) en fonction du nombre d'octets en clair (post signature) de l'instruction.

En consultant le tableau nous appliquons :

INS = **38**, octets en clair = **08**, PROVIDER **1** -> NLB= **0x00**, LB = **0x3C**

```
C1 38 21 B0 76
65 31 D3 B2 97 A5 D2 86 81 DE 82 77 B6 85 8D A9
00 C0 AD 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 3C
-> [96 00]
```

Dans le cas ou on serais dans une situation imprévue, on peut faire une variation manuellement, ou alors on utilise la fonction « FIND P5 » du programme MARKTIMER (fenêtre CMD STUDIO)

Remarque : les valeurs reportés dans le tableau, ne sont pas les seuls valides, il peut exister d'autres couples NLB/LB qui donnent le même résultat.

Nous avons juste mis en page les premiers que nous avons trouvé, en exécutant la variation à partir de 00 0033.

De plus ces valeurs sont trouvées pour les fournisseurs d'accès italiens.

ITALIE

	INS 38 & 40					INS 3C					
	00 00	00 70	00 71	00 72	00 73	00 00	00 70	00 71	00 72	00 73	
00	00 59	00 53	01 0D	01 14	01 A8	00 3A	00 63	02 AC	00 7B	01 9B	Nombre d'octets en clairs après la signature et avant les 5A octets 5A octets en SSE
01	00 58	00 08	00 DA	00 13	01 01	00 26	01 87	00 BB	00 2F	04 2E	
02	00 35	00 77	00 B0	00 55	00 69	00 4D	00 6D	01 39	00 42	00 2F	
03	00 D2	00 40	00 04	00 40	00 59	00 29	00 0E	01 75	00 04	00 10	
04	00 E3	00 27	00 8C	01 44	00 7C	00 2C	00 E9	00 56	00 62	01 C2	
05	00 2B	00 54	00 D4	01 DD	00 80	00 77	00 04	00 13	00 B4	00 14	
06	02 E9	00 8D	00 83	00 36	01 74	00 A3	02 05	01 1F	00 9B	00 3C	
07	00 66	00 47	01 11	00 D2	00 96	00 23	00 73	01 2B	00 EE	01 03	
08	00 09	00 3C	00 9B	00 09	00 0C	00 0E	00 08	00 DF	00 9A	01 45	
09	00 27	00 09	00 62	00 60	00 8D	00 6E	01 02	01 09	00 08	00 60	
0A	00 0B	01 05	01 BD	04 94	02 A6	00 E7	00 05	00 1F	00 A5	00 4E	
0B	00 B9	00 2E	00 B1	00 10	01 AE	00 35	00 10	00 3E	00 11	00 F3	
0C	01 38	00 83	00 EB	02 1C	00 36	00 02	00 77	00 75	00 05	00 A4	
0D	02 80	00 19	00 36	00 3A	00 10	00 27	00 B4	00 20	00 23	01 32	
0E	00 7E	00 17	00 42	00 F3	00 EA	00 11	01 76	00 21	00 51	00 51	
0F	00 54	00 2B	06 9B	00 B3	00 99	00 06	01 0F	02 48	01 1F	00 EE	
10	00 85	01 5F	00 91	00 48	00 27	00 0D	00 85	00 68	00 66	00 DA	
11	00 B4	01 63	00 6A	01 69	01 1B	00 61	00 5C	01 C9	00 2C	01 20	
12	00 DC	01 01	00 B3	00 B2	00 09	00 42	00 86	01 6A	00 87	01 84	
13	00 74	00 69	00 96	00 B7	00 77	00 58	00 A7	00 E4	00 E5	00 24	
14	02 B0	00 0A	00 C6	00 46	00 82	00 0A	01 B2	00 09	00 52	01 C7	
15	02 2B	00 33	02 43	00 49	00 04	00 49	00 22	00 31	00 39	00 75	
16	01 11	00 59	00 33	01 04	00 4F	00 1D	00 3F	00 0A	01 F5	01 5E	
17	03 5C	01 49	06 4D	00 7D	00 7E	00 A0	00 0D	00 33	00 92	00 1A	
18	00 1C	00 2A	00 3C	00 04	01 D3	00 2F	00 B9	00 64	00 5E	00 C3	
19	00 76	00 1E	01 1C	00 2C	00 34	00 51	00 01	00 23	00 5B	00 BF	
1A	00 37	01 0E	00 66	00 8D	00 2B	01 15	00 03	00 2C	00 38	00 84	
1B	01 9F	01 92	01 46	00 50	00 79	00 2B	00 8B	00 4E	00 8A	00 CF	
1C	00 0A	00 73	00 26	00 67	00 08	00 16	00 C5	00 22	00 7E	00 65	
1D	00 80	01 6F	01 DD	00 16	00 B3	00 AE	00 17	00 BD	00 9E	00 2B	
1E	00 B3	00 42	00 10	00 75	01 67	00 39	00 69	00 A4	00 7A	00 BB	
1F	01 49	00 03	00 8A	00 0A	00 66	01 1A	00 32	01 99	00 B1	00 AD	
20	00 69	00 15	00 5D	00 38	01 46	00 43	00 4C	00 69	00 8C	01 04	
21	02 16	00 0E	00 E7	00 23	00 A0	00 21	01 0A	00 9A	00 3B	00 42	
22	00 32	01 20	01 C3	00 C5	00 B6	01 76	00 84	00 6B	00 2A	01 07	
23	01 6A	00 66	00 20	00 81	00 70	00 0F	00 65	00 76	00 71	00 15	
24	00 3F	00 13	00 39	00 8E	01 D1	00 71	00 66	00 74	01 9C	00 13	
25	00 D1	00 36	01 05	00 35	00 54	00 73	01 2C	00 C5	00 15	00 4A	

ESPAGNE

	INS 38 & 40				INS 3C				
	00 00	00 64	00 66	00 67	00 00	00 64	00 66	00 67	
00	00 3D	00 6D	00 28	01 99	01 04	00 32	00 1C	00 0E	Nombre d'octets en clairs après la signature et avant les 5A octets 5A octets en SSE
01	00 4E	00 9C	00 0C	00 36	00 FB	00 D2	00 60	01 6C	
02	00 4A	00 11	00 BD	00 03	00 27	01 21	00 86	00 C9	
03	00 DA	00 01	00 22	00 12	00 01	00 C0	00 7D	00 39	
04	00 1C	00 89	00 10	00 27	00 2A	00 0C	00 4E	00 B7	
05	01 9A	00 9E	00 06	00 D7	00 0C	01 11	00 42	00 2C	
06	00 17	00 6B	00 1D	01 44	00 86	00 58	02 3C	01 D4	
07	01 A2	00 3A	00 03	00 24	00 8E	00 08	00 E9	00 76	
08	00 1A	00 54	00 31	00 D8	00 52	00 51	00 19	00 E6	
09	00 9D	00 8C	01 0E	00 1F	00 07	00 05	00 A2	01 BF	
0A	00 B0	01 5F	00 0A	00 59	00 67	01 EA	02 C9	01 12	
0B	00 30	00 03	00 58	03 6C	00 A0	00 EA	00 47	03 10	
0C	00 26	00 2E	00 6B	00 4A	00 26	00 72	00 2A	00 0C	
0D	02 22	00 1A	00 5A	00 3C	00 6A	00 1C	00 44	00 41	
0E	00 67	00 67	01 5F	00 FA	01 1C	00 37	00 15	00 01	
0F	00 C0	00 3E	00 17	01 3F	00 8D	00 D3	00 C1	00 E4	
10	00 90	01 FC	00 05	00 08	00 05	01 29	00 0B	00 C3	
11	01 0E	00 B2	00 79	00 0E	00 1B	00 A1	00 F8	00 A3	
12	00 39	00 55	00 48	00 A0	00 06	00 3A	00 70	00 6E	
13	00 2C	00 50	01 0A	00 19	00 CC	00 6D	02 0C	01 59	
14	00 8B	00 91	00 2B	00 65	01 2D	02 B3	00 0A	00 C0	
15	00 0B	00 87	01 D6	00 25	01 6A	00 13	00 07	00 E2	
16	00 24	00 40	00 A4	00 0C	00 17	00 0D	00 05	00 6B	
17	00 43	00 88	00 4E	01 41	00 0B	01 1B	00 3C	00 20	
18	00 70	01 09	00 E0	00 52	02 36	00 69	00 27	00 08	
19	00 3A	00 0E	00 41	01 34	00 9F	00 C6	00 13	00 7D	
1A	00 28	00 29	00 32	00 18	00 2C	00 36	00 20	00 1A	
1B	00 44	00 B7	00 C8	00 BF	00 56	00 53	00 94	00 89	
1C	00 07	01 17	00 11	00 23	00 0E	01 8C	00 37	00 04	
1D	00 E9	00 18	00 43	00 07	00 F2	00 45	00 45	00 D8	
1E	01 AC	00 3D	00 1C	00 1B	00 97	00 80	00 26	03 C5	
1F	00 33	00 13	00 64	00 2C	01 BE	00 2C	00 8E	01 00	
20	00 E8	00 D2	00 4B	00 1C	00 59	01 72	00 DF	00 3A	
21	00 C7	00 9B	02 FB	00 96	01 02	00 3D	01 4C	01 30	
22	01 1A	01 AF	00 B1	00 48	00 B5	00 BF	00 0D	00 FE	
23	00 CF	00 7F	00 02	00 33	01 E1	00 0E	00 0C	00 1D	
24	00 B1	00 99	00 12	00 02	00 7E	00 02	00 90	00 9C	
25	01 09	00 FF	03 33	00 7B	00 02	00 DD	00 85	00 14	

Phase 4 : Reset Bug et obtention d'un statut 90 00 sur une instruction « CUSTOM »

Pour passer d'un statut 9600 à un statut 9000, il suffit simplement d'exécuter un cycle, qui pour les maniaques des procédures manuelles consiste à faire en boucle cette procédure :

RESET

SEND

Pour les plus fainéants utilisez la touche MULTISEND du MARKTTIMER (menu déroulant sur RESET BUG) pour résoudre le problème.

Voici un petit exemple de LOG

[illegible]

C'est quoi le RESET- BUG ?

Le statut 96 00 indique que le contrôle sur le PRE-PARSING n'a pas put être franchi, alors comment est-il possible qu'après certain nombre de RESET on obtienne un 90 00 ?.

La cause est à rechercher dans un bug dans l'implémentation de l'UNMASKING, qui se manifeste lorsque vous avez des données inférieures à 8 octets, reprenons l'exemple donné en phase 3 :

```
C1 38 21 B0 76
65 31 D3 B2 97 A5 D2 86 81 DE 82 77 B6 85 8D A9
00 C0 AD 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 09
```

Les octets en vert (P3 et ses paramètres relatifs) ne sont pas pris en compte pour le DECRYPT, il ne reste donc avant la NANO 82 de signature, que trois octets, bien insuffisants pour former un quadruple mot (8 octets), dans ce cas l'UNMASKING subit une crise de position, dans le sens que le résultat ne dépend pas seulement des données (nos trois octets), mais aussi du contenu restant du command BUFFER et d'une partie de RAM qui lui est adjacente où (entre autre chose) sont présents des quantités qui **varient à chaque RESET** (le mécanisme est actuellement en étude).

La conséquence est qu'à chaque RESET, nous avons un résultat différent du DECRYPT, avec la possibilité d'avoir un **PARSING correct** (et du coup un 90 00)

Le nombre d'octets disponibles pour le DECRYPT dépend de P3, à égalité de longueur de l'instruction, plus est grand le quartet fort de P3 et moins vous avez de données pour le DECRYPT. Vous pouvez carrément ne pas avoir de données à décrypter : dans ce cas bien sur, la carte n'exécute pas l'instruction et répond par un ATR ou se bloque et ne se remet en place qu'après un reset (on n'exclue pas une écriture intempestive dans l'EEPROM).

Phase 4 bis : « OCTET BUG » et obtention d'un statut 90 00 sur une instruction « CUSTOM »

Cette méthode arrive au même résultat que le RESET BUG, mais possède indiscutablement des propriétés qui la rende plus attractive.

La méthode consiste à varier la valeur d'un ou plusieurs octets, mis entre le dernier octet de la signature et le premier octet sous enveloppe. A titre d'exemple, faisons varier l'octet mis en évidence dans notre instruction :

[illegible]

Dans ce cas aussi, pour les amoureux du moindre effort, le programme MARKTIMER, nous vient en aide.

Nous allons jouer sur la variation de l'octet 20 c'est ce que nous mettons dans « INDEX BYTE » (dans le programme le rang est exprimé en décimal, nous allons donner à cet octet toutes les valeurs qu'il peut assumer donc nous mettons « 00 » dans la case « FROM » et « FF » dans la case « TO » il ne nous reste plus qu'à mettre la condition d'arrêt qui sera lorsque nous aurons un statut de 90 00 nous mettons 9000 dans la case « STATUS », on lance la fonction « SCAN » et il y a plus qu'à attendre le résultat.

[illegible]

Qu'est ce que le "BUG BYTE » ou chez nous l'OCTET-BUG

Nous avons dit que l'UNMASKING BUGGE dépend des autres données, en plus des octets à mettre en clair, décrivons rapidement ce comportement, (vous aurez l'analyse détaillée dans le décryptage du processus de SUPER-ENCRYPTION). L'UNMASKING exploite comme paramètre la somme des mots (2 octets pour le mot) modulo 0x4000 d'une partie des données du corps de l'instruction à démasquer. Quand les données sont inférieures à 8, l'index utilisé pour prélever les données du calcul assument une valeur négative, en faisant ainsi, on étend la somme aux données présente dans le COMMANDBUFFER (qui suivent les octets à démasquer) en plus d'une partie de la région de la RAM suivante. Si un de n'importe quel donnée varie en gardant la même somme des mots modulo 0x400, le démasquage ne change pas.

Et vice-versa, en changeant la sommes des mots, on change le résultat avec la possibilité d'arriver à un PARSING correct.

Caractéristiques et propriété de l'OCTET-BUG

(Recueil revu de divers postes)

On construit une C1 38 (à partir d'une C1 36 courte) avec une LEN assez longue pour permettre la variation des octets après la signature et avant les derniers 0x5A

Sans jamais faire de RESET, on fait varier un des octets en clair, après la signature, jusqu'à obtenir un 90 00.

Nous appellerons cet octet, l'octet « i », on modifie maintenant cet octet de cette façon : « **i+2n** » ou « **i-2n** ». De tel façon que pour chaque « n » le nouveau octet obtenu soit toujours dans les limites exposé au dessus et donc entre le dernier octet de la signature et le premier avant les dernier 0x5A)

On remarque des choses très intéressantes :

- La séquence des statut 9600/9000 obtenue est identique
- Il a une répétitivité du quartet faible chaque 4 quartets forts
- Répétant la même procédure avec une trame sous SSE différent, le phénomène se répète mais avec d'autres valeurs.

Un petit exemple clarificateur :

...82 s0 s1 s2 s3 s4 s5 s6 s7 00 00 00 00 00 00 00 00 00 00 96 00

Supposons qu'un octet, qui nous donne le statut 90 00 soit égal à 0x3C

...82 s0 s1 s2 s3 s4 s5 s6 s7 00 **3C** 00 00 00 00 00 00 00 00 00 90 00
...82 s0 s1 s2 s3 s4 s5 s6 s7 00 **7C** 00 00 00 00 00 00 00 00 00 90 00
...82 s0 s1 s2 s3 s4 s5 s6 s7 00 **BC** 00 00 00 00 00 00 00 00 00 90 00
...82 s0 s1 s2 s3 s4 s5 s6 s7 00 **FC** 00 00 00 00 00 00 00 00 00 90 00

Il existe une logique dans la séquence des statuts qu'on obtient. Ces octets entre en jeux quand on doit décrypter un quadruple mot (8 octets) incomplet.

En réalité les octets doivent être considérés en couple... faisons un exemple :

```
C1 38 21 B0 86
51 91 95 48 45 C9 B7 69 A1 0E 82 4B 7F EC 94 0C
60 E4 F4 00 15 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 1E [90 00]
```

Si nous sommes dans la condition « **Quartet fort de P3 impair** » ; on subdivise les octets en clair par couple à partir du second octet en clair.

Si nous sommes dans la condition « **Quartet fort de P3 pair** » nous avons le même raisonnement mais les couples d'octets sont prises à partir du premier octet en clair.

```
C1 38 21 B0 86
51 91 95 48 45 C9 B7 69 A1 0E 82 4B 7F EC 94 0C
60 E4 F4 00 15 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 1E [90 00]
```

Pour l'instant ne prenons pas en considération les octets sous enveloppe.
Ces couples d'octets doivent être vues comme **quantité à 16 bits (MOT)**, en faisant ainsi on découvre que toutes les combinaisons possible de mots, donnent la même somme modulo **0x4000** (16384 en décimal), ils donnent en fait un résultat identique.

Pour l'instruction montrée avant, nous avons : **somme (Mot mod 0x4000) = (1500 mod 0x4000) = 1500**

Deux instructions équivalentes pourraient par exemple être celle-ci :

C1 38 21 B0 86

51 91 95 48 45 C9 B7 69 A1 0E 82 4B 7F EC 94 0C
 60 E4 F4 00 **55 00 00 00 00 00 00 00 00 00 00 00**
00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 1E [90 00]

**(somme Mot mod 0x4000) =
 (5500 mod 0x4000) = 1500**

Et :

C1 38 21 B0 86

51 91 95 48 45 C9 B7 69 A1 0E 82 4B 7F EC 94 0C
 60 E4 F4 00 **15 00 3F FE 40 02 00 00 00 00 00 00**
00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 1E [90 00]

**(somme Mot mod 0x4000) =
 (1500 + 3FFE + 4002 mod
 0x4000) = (9500 mod 0x4000)
 = 1500**

Généralisons :

C1 38/40 P1 P2 LEN

P3 P4 x0 x1 x2 x3 x4 x5 x6 x7

82 s0 s1 s2 s3 s4 s5 s6 s7

q00 q01 q02 q03 q04 q05 q06 q07

q08 q09 q10 q11 q12 q13 q14 q15

q16 q17 q18 q19 q20 q21 q22 q23 q24 + 90 octets sous SEE/Enveloppe

En tenant compte de la distinction entre quartets fort de P3 pair et impair, les octets en clair sont subdivisés en mot (dans l'exemple q00.....q24, mais peuvent être en quantité différente)

QUARTE FORT P3 IMPAIR	QUARTET FORT P3 PAIR
(q01 q02)	(q00 q01)
(q03 q04)	(q02 q03)
(q05 q06)	(q04 q05)
(q07 q08)	(q06 q07)
(q09 q10)	(q08 q09)
(q11 q12)	(q10 q11)
(q13 q14)	(q12 q13)
(q15 q16)	(q14 q15)
(q17 q18)	(q16 q17)
(q19 q20)	(q18 q19)
(q21 q22)	(q20 q21)
(q23 q24)	(q22 q23)
L'octet non couplé doit être considéré comme (00 q00)	L'octet non couplé Doit être considéré comme (q24 00)

Note : la valeur 00 à été introduite pour traiter plus commodément l'octet non couplé, il ne se référer à aucun octet réellement présent dans la RAM.

Fixons les octets : P3 P4 x0 x1 x2 x3 x4 x5 x6 x7 82 s0 s1 s2 s3 s4 s5 s6 s7 et faisons varier les octets en clairs, nous obtenons un maximum de 16384 DECRYPT divers (0x4000 en hexadécimal) il est, de plus, possible identifier l'ensemble de toutes les combinaisons qui donnent lieu au même DECRYPT, grâce à une simple relation :

Quartet fort P3 impair

$[(00\ q0) + (q1\ q2) + (q3\ q4) + (q5\ q6) + (q7\ q8) + (q9\ q10) + (q11\ q12) + (q13\ q14) + (q15\ q16) + (q17\ q18) + (q19\ q20) + (q21\ q22) + (q23\ q24)] \text{ modulo } 0x4000$

Quartet fort P3 pair

$[(q0\ q1) + (q2\ q3) + (q4\ q5) + (q6\ q7) + (q8\ q9) + (q10\ q11) + (q12\ q13) + (q14\ q15) + (q16\ q17) + (q18\ q19) + (q20\ q21) + (q22\ q23) + (q24\ 00)] \text{ modulo } 0x4000$

Bien jusqu'à ici tout va bien, mais un simple RESET est suffisant pour chambouler tout ça. A chaque RESET, le DECRYPT de la commande semble changer et donc ses effets. Continuons avec une autre expérience.

C1 38 21 F0 71
 65 11 A4 96 FA 52 EF 7A 88 7E 82 0E ED 18 95 B0
 CA 14 97 **XX YY** 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 27

On fait varier un MOT en clair de l'instruction (les octets XX et YY) et on note les valeurs qui correspondes à un statut 90 00 (on construit ainsi la chaîne dite : « succession des 90 00 »).

Après ça on envoie le RESET et on répète le processus. La preuve s'effectue avec un « MOT » de 0000 à 3FFF : vu la périodicité du comportement (selon la somme MOTS modulo 0x4000), il n'est pas nécessaire d'utiliser d'autres valeurs.

On obtiens deux suites de valeurs, le premier enseignement c'est que le nombre des « MOTS » qui donnent lieu à un 90 00 est constant à chaque RESET.

Le second enseignement, bien plus important, c'est que les successions sont liées entre elles.

En prenant les valeurs obtenues et calculant les « DELTA », c'est à dire les différences entre les « MOTS », on construit la « succession des delta ». En confrontant les deux séries on remarque qu'on obtiens les même valeurs, seul le point de départ change (c'est comme si les successions avaient été translatées).

Succession (A)		Succession (B)	
MOT	DELTA	MOT	DELTA
0110	17	35CD	17
011D	13	35DA	13
0127	10	35E4	10
012E	7	35EB	7
0130	2	35ED	2
0135	5	35F2	5
015F	42	361C	42
0162	3	361F	3
0189	39	3646	39
019D	20	365A	20
019F	2	365C	2
01BD	30	367A	30
01C0	3	367D	3
01C1	1	367E	1
01C9	8	3686	8
01DD	20	369A	20
01E6	9	36A3	9
01EE	8	36AB	8
01F7	9	36B4	9
01FC	5	36B9	5
020B	15	36C8	15

A la lumière du fait que :

- Entre les deux successions (A) et (B), nous avons fait un RESET, mais aucun octet de la commande n'a été modifié.
- Le statut 9600/9000 dépend de : « **somme mots modulo 0x4000** »

On peut conclure qu'il est possible de lier le RESET-BUG avec l'OCTET-BUG (BYTE-BUG), en pratique, aux effets du PRE-PARSING, la variabilité due au RESET peut être considérée comme un « MOT » inconnu à ajouter à la somme.

En effet, une fois alignée la succession des deltas, on voit que la différence entre les « MOTS » homologues, est constante pour toute la séquence (mais il change à chaque RESET).

Voici quelques valeurs en exemple :

35CD - 0110 = 34BD

35DA - 011D = 34BD

35E4 - 0127 = 34BD

35EB - 012E = 34BD

Il est maintenant possible de tirer la conclusion suivante :

Les octets « RANDOM » entre en compte dans le calcul de la somme des « MOTS »

QUARTE FORT P3 impair

$[(\text{random WORD}) + (q0\ q0) + (q1\ q2) + (q3\ q4) + (q5\ q6) + (q7\ q8) + (q9\ q10) + (q11\ q12) + (q13\ q14) + (q15\ q16) + (q17\ q18) + (q19\ q20) + (q21\ q22) + (q23\ q24)] \text{ modulo } 0x4000$

QUARTET FORT P3 Pair

$[(\text{random WORD}) + (q0\ q1) + (q2\ q3) + (q4\ q5) + (q6\ q7) + (q8\ q9) + (q10\ q11) + (q12\ q13) + (q14\ q15) + (q16\ q17) + (q18\ q19) + (q20\ q21) + (q22\ q23) + (q24\ 00)] \text{ modulo } 0x4000$

Autre propriété liée au BUG :

Dans la somme des « MOTS » entre aussi les octets sous enveloppe même s'ils ont été décryptés.

Vérification de la propriété :

Instruction <1>

```
C1 38 21 F0 71 38 65 11 A4 96 FA 52 EF 7A 88 7E 82 0E ED 18 95 B0 CA 14 97 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 27 90 00
```

Instruction <2>

```
C1 38 21 F0 71 38 65 11 A4 96 FA 52 EF 7A 88 7E 82 0E ED 18 95 B0 CA 14 97 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
34 00 27 90 00
```

les deux instructions donnent lieu à un de-enveloppe différent. On peut vérifier à travers une lecture indirecte du premier octet sous enveloppe avec le bug 38/36 (en vert)

```
C1 36 21 00 14 36 86 CA 14 97 03 00 00 00 24 03 82 FF FF FF FF FF FF FF FF 90 35
```

```
C1 36 21 00 14 36 86 CA 14 97 00 00 00 00 EA 03 82 FF FF FF FF FF FF FF FF 90 35
```

En construisant le tableau des "90 00", sans faire de RESET entre l'instruction <1> et l'instruction <2>, de tel façon de ne pas varier le « RADOMWORD », lié au RESET, il résultera que l'unique variable est le résultat du de-enveloppe/de-SSE. Encore une fois nous obtenons une successions de deltas égaux, avec un certain désalignement, dans ce cas-ci, non introduit par le RESET, mais du de-enveloppe qui est entré à faire partie du sommatoire des « MOTS ».

On doit observer que la somme ne dépend pas des octets sautés de P3, mais dans cette somme entre aussi les octets qui subissent le PARSING, la NANO 82 et la signature. Le type « tableau des 9000 » change avec la variation de P3 et P4, la dépendance n'est pas directe mais en général du type : **f(P3P4)**.

En résumant, on a vu que l'UNMASKING BUGGE dépend de :

1. Le contenu des octets en SE
2. Le contenu des valeurs assumés par P3 P4
3. Le contenu des octets en (de)ENVELOPPE
4. Le contenu des octets en clair post signature
5. Le reset (Contenu de la RAM hors COMMAND-BUFFER)

Phase 5A : utilisation d'une instruction « CUSTOM » pour générer une EMM

Après qu'on ai obtenu un statut de 9000 à travers le RESET-BUG ou l'OCTET-BUG, s'ouvrent diverses possibilités :

Générer d'ultérieures INS « CUSTOM » à travers l'instruction 38/36 (en particulier pour chercher des réponses longues) ou bien convertir notre INS 38 en sa correspondante INS 40. Analysons le second cas, en nous rappelant que notre instruction « CUSTOM » (utilisateur) est ainsi formée (cas de statut 9000 obtenu avec l'OCTET-BUG)

C1 38 21 B0 76 65 31... 00 09

Substituons l'octet 38 avec l'octet 40 nous obtenons :

C1 40 21 B0 76 65 31... 00 09

C'est à dire dans sa forme étendue :

C1 40 21 B0 76
65 31 D3 B2 97 A5 D2 86 81 DE 82 77 B6 85 8D A9
00 C0 AD 02 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 09

Envoyons cette instruction sans faire de reset :

C1 40 21 B0 76
65 31 D3 B2 97 A5 D2 86 81 DE 82 77 B6 85 8D A9
00 C0 AD 02 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 09 **[90 00]**



ATTENTION :

Les C1 40 ainsi construites, si envoyées, peuvent causer l'altération ou l'effacement des données mémorisées dans la carte avec le risque de la rendre complètement inutilisable. Soyer donc prudents.

On peut aussi avoir d'autres Statuts

En conclusion l'EMM utilisateur à été accepté.

Pour vérifier si l'EMM a écrit quelque chose en EEPROM, on peut utiliser, la encore très nébuleuse instruction **C1 48**.

C1 48 00 00 01 FF

A chaque C1 40 suivant la C1 48, nous pouvons obtenir un statut de 90A0 ou 9090, selon qu'il y ai eu une écriture dans l'EEPROM ou pas. L'effet de la C1 48 cesse avec un simple RESET.

Phase 5B : utilisation d'une instruction « CUSTOM » pour générer d'autres réponses

Il pourrait être intéressant de poursuivre, une fois l'INS « CUSTOM » construite, à la construction d'un grand nombre d'instructions valides. Prenons une INS générique, construite à partir d'une réponse courte et analysons là :

C1 38 P1 P2 LEN

P3 P4 x0 x1 x2 x3 x4 x5 x6 x7 82 s0 s1 s2 s3 s4 s5 s6 s7

+ les octets éventuels en clair + 90 octets sous enveloppe

En se référant aux P3, nous avons déjà vu que son quartet fort, nous indique combien d'octets ne pas considérer pour l'exécution de l'instruction, en plus pour les quartets forts trop grand (supérieure à 8), la carte ne répond pas comme il faut, en particulier pour P3 = 9x on obtiens une sortie en ATR et pour P3 supérieure, nous n'avons aucune réponse.

P3=1x -> données sautées = P4
P3=2x -> données sautées = P4 x0
P3=3x -> données sautées = P4 x0 x1
P3=4x -> données sautées = P4 x0 x1 x2
P3=5x -> données sautées = P4 x0 x1 x2 x3
P3=6x -> données sautées = P4 x0 x1 x2 x3 x4
P3=7x -> données sautées = P4 x0 x1 x2 x3 x4 x5
P3=8x -> données sautées = P4 x0 x1 x2 x3 x4 x5 x6

Les octets SE qui seront décryptés seront les suivants :

P3=1x -> Octet en SE = x0 x1 x2 x3 x4 x5 x6 x7
P3=2x -> Octet en SE = x1 x2 x3 x4 x5 x6 x7
P3=3x -> Octet en SE = x2 x3 x4 x5 x6 x7
P3=4x -> Octet en SE = x3 x4 x5 x6 x7
P3=5x -> Octet en SE = x4 x5 x6 x7
P3=6x -> Octet en SE = x5 x6 x7
P3=7x -> Octet en SE = x6 x7
P3=8x -> Octet en SE = x7

La première question à se poser c'est : vu que l'instruction C1 38 demande un certain nombre de paramètres, où va t'elle les prélever en cas de données insuffisantes ?

La réponse est immédiate, elle utilise les octets successifs à x0....x7, c'est à dire la NANO 82, la signature et les octets qui la suivent si nécessaire. Se rappelant que la C1 38 se présente sous cette forme :

C1 38 P1 P2 LEN

16 d1 2A d2 d3 2B d4 d5 86 b0 b1 b2 b3 b4 b5 b6 b7 82 s0 s1 s2 s3 s4 s5 s6 s7 etc...

et que l'INS CUSTOM est construite de cette façon :

C1 38 P1 P2 LEN

P3 P4 x0 x1 x2 x3 x4 x5 x6 x7 82 s0 s1 s2 s3 s4 s5 s6 s7 m0 m1 m2 m3 m4 m5 m6 etc

Il en résultera les correspondances suivantes :

P3	.d1	.d2d3	.d4d5	.b0 b1 b2 b3 b4 b5 b6 b7
1y	x1	x3 x4	x6 x7	s0 s1 s2 s3 s4 s5 s6 s7
2y	x2	x4 x5	x7 82	s1 s2 s3 s4 s5 s6 s7 m0
3y	x3	x5 x6	82 s0	s2 s3 s4 s5 s6 s7 m0 m1
4y	x4	x6 x7	s0 s1	s3 s4 s5 s6 s7 m0 m1 m2
5y	x5	x7 82	s1 s2	s4 s5 s6 s7 m0 m1 m2 m3
6y	x6	82 s0	s2 s3	s5 s6 s7 m0 m1 m2 m3 m4
7y	x7	s0 s1	s3 s4	s6 s7 m0 m1 m2 m3 m4 m5
8y	82	s1 s2	s4 s5	s7 m0 m1 m2 m3 m4 m5 m6

Quand on construit une instruction, nous connaissons quels octets ne seront pas décryptés, alors nous pouvons savoir anticipativement quelles valeurs assument (quelques unes ou tous) les paramètres de la C1 38.

Voyons comment nous servir de cette caractéristique pour générer d'autres réponses.

LEN 0x13 – Génération multiple

Nous avons déjà vu quel était le contenu en clair des réponses de longueur 13 :

C1 36 21 00 LEN

36

86 b0 b1 b2 b3 b4 b5 b6 b7

03/04 82 ..

En agissant sur b0....b7, il est possible faire varier le CRYPT de la réponse, de tel manière à générer une famille de C1 36 différentes entre elles.

Les méthodes pour faire varier b0....b7 se découvrent en examinant le tableau précédent : ou en changeant les octets de la signature (mais ceci imposerait la recherche d'une nouvelle INS CUSTOM) ou alors on change les octets m0,m1 etc.... qui je le rappelle peuvent être en clair ([voir page 32](#)), bien sur il faut tenir compte de l'OCTET-BUG, donc la variation des octets doit être effectuée de façon à maintenir constant (la somme « MOTS » modulo 0x4000).

Supposons que nous avons comme valide, l'instruction de la page 32...

```
C1 36 21 B0 14
36
13 65 31 D3 B2 97 A5 D2
86 81 DE 82 77 B6 85 8D
A9 00 C0 AD [90 00]
```

Nous pouvons l'utiliser dans un script pour WINEXPLORER qui exécute la génération multiple des réponses.

Ce qui suit à été commenté pour la compréhension de la méthode :

```
' Générateur d'INS courte (LEN = 0x13)
' Filtre P3 : (>0F) et (<90) - Filtre P4 : (=x1) OR (=x9)
' Fonction cnvByte
' Converti un octet de l'hexadécimal à une chaîne

Private Function cnvByte(Bdata)
HEXa =
Array("0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F")
TBdata = (Bdata AND 255)
cnvByte = HEXa(TBdata\16) + HEXa(TBdata AND 15) + " "
End Function

' Fonction cnvWord
' Converti un mot (2 octets)
' d'hexadécimal à chaîne (nécessité de la fonction cnvWord)

Private Function cnvWord(Wdata)
TWdata = (Wdata AND 65535)
cnvWord = cnvByte(TWdata\256) + cnvByte(TWdata)
End Function

' Fonction cnvWordSWP
' converti un mot (2 octets)
' d'hexadécimal en chaîne et échange les octets nécessaires de la fonction
' nvWord)

Private Function cnvWordSWP(Wdata)
TWdata = (Wdata AND 65535)
cnvWordSWP = cnvByte(TWdata) + cnvByte(TWdata\256)
End Function
```

[illegible]

```

' Recherche du statut 9000 et réponse de la longueur voulue

Sc.Print("Building root-command...")
For Root = 0 to 16383
Sc.Write(Ins38)
Sc.Read(01)

' Comme on peut le voir, la recherche de la bonne réponse est
' faite en utilisant l'OCTET-BUG: "Root" est un MOT que l'on fait
' varié de 0000 à 3FFF jusqu'à l'obtention du statut 9000
' A coté de lui est présent un MOT qui est tenu fixe à FFFF
' et qui nous sera très commode par la suite.
' ATTENTION de la façon que la commande est construite, le script fonctionnera
' seulement si les données de départ de P3 = 4x ; 6x ou 8x
' Pour utiliser P3 = 3x, 5x, 7x on met "cnvWordSWP" a la place de "cnvWord"

Sc.Write(Data+cnvWord(Root)+cnvWord(&HFFFF)+Zeros+NLB_LB)
Sc.Read(02)
myByte = Sc.Getbyte(0)

' Voici le test sur le statut 90xx

If (myByte = &H90) Then
Sc.Write(Ins36)
Sc.Read(2)
LenAnsw = Sc.Getbyte(1)

' Voici le test pour voir si c'est une bonne INS LEN = 13)

If (LenAnsw = LenWanted) Then
Sc.Read(LenWanted+2)
RootFound = 1
Exit For
End If

' Celles qui donnent 9015 ont comme premier octet 86 (supérieure de 13)

If (LenAnsw > LenWanted) Then
Sc.Read(LenWanted+1)
Else
Sc.Read(LenWanted+2)
End If
End If
Next

' Si on arrive ici cela veut dire qu'on à trouvé la réponse
' juste ou bien que nous atteint les 16384 possibilités'.
' RootFound indique si on à trouvé la commande

If (RootFound = 1) Then
Sc.Print("FOUND!"&Chr(13)+"Ready to build answers (please wait)"&Chr(13))

```

```

' Je fait une boucle sur 2 MOTS
' et je montre les réponses filtrées
' Tentatives: 65536

For Param = 0 to 65535
Sc.Write(Ins38)
Sc.Read(01)

' Ici on vois comment on trouve les bonnes réponses:
' à partir de la valeur Root on fait varier deux MOTS de manière
' complémentaire, en maintenant ainsi constante la(somme MOT modulo 0x4000)
' Ici aussi (pour P3 = 3x, 5x, 7x) on met "cnvWordSWP" a la place de
' "cnvWord"

Sc.Write(Data+cnvWord(Root+Param)+cnvWord(&HFFFF-Param)+Zeros+NLB_LB)
Sc.Read(02)
Sc.Write(Ins36)
Sc.Read(LenWanted+4)

' Ici on prélève les premiers deux octets de la réponse et on
' évalue si ce sont des valeurs valides pour P3 e P4

myP3 = Sc.Getbyte(2)
myP4 = Sc.Getbyte(3)
myP4c= myP4 AND &H0F

' Si P3 P4 sont ok alors je montre la réponse

If ((myP3>&H0F) AND (myP3<&H90) AND ((myP4c = 1) OR (myP4c = 9))) Then
Sc.Print(cnvByte(myP3))
Sc.Print(cnvByte(myP4))
For myByte = 4 to (LenWanted+1)
Sc.Print(cnvByte(Sc.Getbyte(myByte)))
Next
Sc.Print(Chr(13))
End If
Next
Else

' Si nous avons de la chance ... :(

Sc.Print("NOT FOUND! Try with different C1 38!")
End If

' Ici fini le script

Wx.Closeport
End Sub

```

Le script comme il est, est prêt à l'utilisation, il suffit de substituer ses propres données, là ou on initialise les variables.

Plusieurs programmes plus ou moins miraculeux, présent sur la toile, utilise un procédé analogue pour générer des instructions courtes

LEN 0x1C – procédure par tentatives

Le discours se complique pour des longueurs supérieures ; dans le cas d'une LEN = 1C, il est nécessaire de cerner une instruction courte qui donne lieu à un **d1** = 00 ou 01. Pour arrivé à ça, nous avons besoin (comme d'habitude) de l'OCTET-BUG

Sachant que pour chaque « somme MOTS modulo $0x4000$ » correspond un DECRYPT différent de l'instruction de départ, on fait varier les octets en clairs post-signature jusqu'à obtenir une C1 38 avec statut 9000 et la correspondante C1 36 avec une réponse à la longueur voulue.

Vu que nous avons 16384 combinaisons, il existe de raisonnables possibilités (mais pas une certitude) de trouver un **d1** juste.

Si on ne le trouve pas (marqué pas de chance), les données de l'instruction du départ doivent changer.

Le script précédent est aussi valide pour ce type de recherche, une fois effectué quelques petites modifications.

[illegible]

```

NLB_LB = "00 27"

' Modifié : la LEN est passée à 1D

Ins36 = "C1 36 21 91 1D "

' Modifié : ajoutée la visualisation de la réponse en clair
' pour comprendre (de la PSEUDO-NANO 83/84) si d1 = 00 ou 01

Ins36p= "C1 36 21 01 1D"

' Modifié : dans ce cas nous voulons des réponse longues 1C

LenWanted = &H1C
Wx.ClosePort
Wx.ResetMode = 1
Wx.IgnoreTimeouts = 1
Wx.OpenPort
Sc.Reset
Sc.Verbose = 0
RootFound = 0
Sc.Print("Building root-command...")
For Root = 0 to 16383

' Modifié : vu que la recherche peut prendre du temps ,
' on a insérer, par commodité, un indicateur de "STEP"

If ((Root Mod 256) = 0) Then Sc.Print("Step : "+cnvWord(Root)+chr(13))
Sc.Write(Ins38)
Sc.Read(01)

' Pour utiliser P3 = 4x, 6x laisser comme c'est
' Pour utiliser P3 = 3x, 5x, 7x il faut "cnvWordSWP" a la place de
"cnvWord"

Sc.Write(Data+cnvWord(Root)+cnvWord(&HFFFF)+Zeros+NLB_LB)
Sc.Read(02)
myByte = Sc.Getbyte(0)
If (myByte = &H90) Then
Sc.Write(Ins36)
Sc.Read(2)
LenAnsw = Sc.Getbyte(1)
If (LenAnsw = LenWanted) Then
Sc.Read(LenWanted+2)
RootFound = 1
Exit For
End If
If (LenAnsw > LenWanted) Then
Sc.Read(LenWanted+1)
Else
Sc.Read(LenWanted+2)
End If
End If
Next
If (RootFound = 1) Then

```

```

' Modifié : dans ce cas avant de démarrer avec la
' génération de la réponse, montre l'instruction en clair
' pour comprendre quel valeur on à obtenu pour d1

Sc.Print("FOUND!" + Chr(13))
Sc.Write(Ins38)
Sc.Read(01)

' Pour utiliser P3 = 4x, 6x laisser comme c'est
' Pour utiliser P3 = 3x, 5x, 7x mettez "cnvWordSWP" à la place de "cnvWord"

Sc.Write(Data+cnvWord(Root)+cnvWord(&HFFFF)+Zeros+NLB_LB)
Sc.Read(02)
Sc.Verbose = 1
Sc.Write(Ins36p)
Sc.Read(LenWanted+3)
Sc.Verbose = 0
Sc.Print("Ready to build answers (please wait)" + Chr(13))
For Param = 0 to 65535
Sc.Write(Ins38)
Sc.Read(01)

' Pour utiliser P3 = 4x, 6x laisser comme c'est
' Pour utiliser P3 = 3x, 5x, 7x mettez "cnvWordSWP" à la place de "cnvWord"

Sc.Write(Data+cnvWord(Root+Param)+cnvWord(&HFFFF-Param)+Zeros+NLB_LB)
Sc.Read(02)
Sc.Write(Ins36)
Sc.Read(LenWanted+4)
myP3 = Sc.Getbyte(2)
myP4 = Sc.Getbyte(3)
myP4c= myP4 AND &H0F

' Modifié : le filtre sur P3 est seulement "P3 >0F")

If ((myP3>&H0F) AND ((myP4c = 1) OR (myP4c = 9))) Then
Sc.Print(cnvByte(myP3))
Sc.Print(cnvByte(myP4))
For myByte = 4 to (LenWanted+1)
Sc.Print(cnvByte(Sc.Getbyte(myByte)))
Next.54
Sc.Print(Chr(13))
End If
Next
Else
Sc.Print("NOT FOUND! Try with different C1 38!")
End If
Wx.Closeport
End Sub

```

Observation : ce type de recherche n'est pas possible à partir d'une instruction CUSTOM avec P3 = 8x, comme vous pouvez le voir du tableau, on aurait une valeur fixée pour d1, qui tomberait exactement sur la NANO 82, laquelle ne subit aucun DECRYPT et/ou démasquage.

LEN 0x1F/0x24 –Procédure pour tentatives pilotées (utilisation de la signature)

Cette famille de réponses demande, en plus d'une valeur valide de **d1**, aussi des valeurs valides pour **d2 d3** :

.d1 = 03	Existence d'un EVENT PPV avec un ID supérieure ou égal à d2d3
.d1 = 04	Existence d'un enregistrement Ex avec le premier octet égal à d3
.d1 = 06	Existence d'un enregistrement avec index supérieure ou égal à d2d3

La condition se réfère aux données relatives au seul fournisseur d'accès à qui est adressé cette commande (quartet faible de P1).

Cette condition impose le choix d'une instruction CUSTOM opportune pour pouvoir générer les réponses de type long.

Observons la correspondance entre les octets de l'instruction CUSTOM et les paramètres d2d3 :

P3	.d2	d3
1y	x3	x4
2y	x4	x5
3y	x5	x6
4y	x6	x7
5y	x7	82
6y	82	s0
7y	s0	s1
8y	s1	s2

Pour les instruction CUSTOM avec P3 compris entre 0x10 et 0x5F, on note que les paramètres dépendent du dé(cryptage) de la commande, tandis que pour un P3 compris entre 0x60 et 0x8F, nous savons exactement ce qu'ils valent, parce que sont prélevés les octets de la signature qui sont en clairs.

La construction des réponse de type long sera donc constitué en deux phases distinctes:

- Recherche de l'instruction de départ de type court.
- Recherche de la réponse de type long.

LEN 0xF/0x24 – Recherche d'instruction de type court

On a dit qu'on devait se servir d'une réponse avec P3 comprise entre 0x60 et 0x8F, en réalité, nous allons devoir rétrécir le champ de recherche :

- Pour P3 = 6x, d2 est fixé à la valeur 82, donc seul la variation de d3 est possible (ne pas utiliser pour d1 = 03,06)
- Pour P3 = 8x, d1 est fixé à la valeur 82 (voir l'observation de la page précédente)

La conséquence est que seul sera utile l'INS CUSTOM avec P3 = 7x. On procédera à la recherche des réponses courtes en utilisant, par exemple, avec une petite modification au filtre de la réponse.

A la place de :

```
myP3 = Sc.Getbyte(2)
myP4 = Sc.Getbyte(3)
myP4c= myP4 AND &H0F
If ((myP3>&H0F) AND (myP3<&H90) AND ((myP4c = 1) OR (myP4c = 9))) Then
```

On substitue :

```
myP3 = Sc.Getbyte(2)
myp3c= myP3 AND &HF0
myP4 = Sc.Getbyte(3)
myP4c= myP4 AND &H0F
If ((myp3c = &H70) AND ((myP4c = 1) OR (myP4c = 9))) Then
```

De toutes les réponses obtenues, on sélectionnera la bonne. Nous avons dit que « d2d3 » = « s0s1 » ou s0 et s1 sont les deux premiers octets de la signature.

Si nous voulons par exemple une réponse longue 0x1F avec d1 = 03 et nous avons sur la carte un enregistrement Bx relatif à l'EVENT-ID 16 00, nous chercherions :

"s0" "s1" < 16 00

Si par contre nous voulons une réponse longue 0x1F avec d1 = 04 et nous avons sur la carte un enregistrement Ex du type 50 (enregistrement présent sur toutes V7 correctement activées) nous chercherions :

s0 n'importe lequel

s1 = 50

si enfin nous voulons une réponse longue 0x24 et nous avons (pour le fournisseur d'accès de P1) le dernier enregistrement positionné à l'index 00 21, nous chercherions :

"s0" "s1" < 00 21

LEN 0xF/0x24 – Recherche d'instruction de type long

Une fois trouvé la commande de type court à utiliser comme semence, on procède à la recherche des instructions longues en utilisant la procédure analogue à celles des réponses de longueur 0x1C.

Le second script des pages précédentes nous permet (avec un peu de modifications d'exécuter la recherche) :

```
' Générateur de INS longue (LEN = 0x1F / 0x24)
' Filtre P3 : > 0F (écrit aussi celles qui PARSING outre la NANO 82)
' Filtre P4 : (=x1) OR (=x9)
```

```
Private Function cnvByte(Bdata)
HEXa =
Array("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F")
TBdata = (Bdata AND 255)
cnvByte = HEXa(TBdata\16) + HEXa(TBdata AND 15) + " "
End Function
Private Function cnvWord(Wdata)
TWdata = (Wdata AND 65535)
cnvWord = cnvByte(TWdata\256) + cnvByte(TWdata)
End Function
Private Function cnvWordSWP(Wdata)
TWdata = (Wdata AND 65535)
cnvWordSWP = cnvByte(TWdata) + cnvByte(TWdata\256)
End Function
Sub Main()
```

' Se rappeler d'insérer les P1 P2 opportuns

```
Ins38 = "C1 38 P1 P2 71"
```

```
' Substituer les données de la réponse courte qui a les valeurs "s0 s1"
' voulu
```

[illegible]

```

NLB_LB = "00 27"

' Substituer les P1 e P2 qui vous intéresses et a la place de LEN
' la valeur 20 (pour les réponse 0x1F) ou 25 (pour les réponses 0x24)

Ins36 = "C1 36 P1 P2 LEN"

' Substituer les P1 qui vous intéresses et a la place de LEN
' la valeur 20 (pour les réponse 0x1F) ou 25 (pour les réponses 0x24)

Ins36p= "C1 36 P1 00 LEN"

' Insérer la longueur de la réponse voulue

LenWanted = &H1F

' Substituer &H24 si on veut celle à LEN 0x24

Wx.ClosePort
Wx.ResetMode = 1
Wx.IgnoreTimeouts = 1
Wx.OpenPort
Sc.Reset
Sc.Verbose = 0
RootFound = 0
Sc.Print("Building root-command...")
For Root = 0 to 16383
If ((Root Mod 256) = 0) Then Sc.Print("Step : "+cnvWord(Root)+chr(13))
Sc.Write(Ins38)
Sc.Read(01)

' P3 devra être = 7x

Sc.Write(Data+cnvWordSWP(Root)+cnvWordSWP(&HFFFF)+Zeros+NLB_LB)
Sc.Read(02)
myByte = Sc.Getbyte(0)
If (myByte = &H90) Then
Sc.Write(Ins36)
Sc.Read(2)
LenAnsw = Sc.Getbyte(1)
If (LenAnsw = LenWanted) Then
Sc.Read(LenWanted+2)
RootFound = 1
Exit For
End If
If (LenAnsw > LenWanted) Then
Sc.Read(LenWanted+1)
Else
Sc.Read(LenWanted+2)
End If.

```

```

End If
Next
If (RootFound = 1) Then
Sc.Print("FOUND!" + Chr(13))
Sc.Write(Ins38)
Sc.Read(01)

' P3 devra être = 7x

Sc.Write(Data+cnvWordSWP(Root)+cnvWordSWP(&HFFFF)+Zeros+NLB_LB)
Sc.Read(02)
Sc.Verbose = 1
Sc.Write(Ins36p)
Sc.Read(LenWanted+3)
Sc.Verbose = 0
Sc.Print("Ready to build answers (please wait)" + Chr(13))
For Param = 0 to 65535
Sc.Write(Ins38)
Sc.Read(01)

' P3 devra être = 7x

Sc.Write(Data+cnvWordSWP(Root+Param)+cnvWordSWP(&HFFFF-Param)+Zeros+NLB_LB)
Sc.Read(02)
Sc.Write(Ins36)
Sc.Read(LenWanted+4)
myP3 = Sc.Getbyte(2)
myP4 = Sc.Getbyte(3)
myP4c= myP4 AND &H0F
If ((myP3>&H0F) AND ((myP4c = 1) OR (myP4c = 9))) Then
Sc.Print(cnvByte(myP3))
Sc.Print(cnvByte(myP4))
For myByte = 4 to (LenWanted+1)
Sc.Print(cnvByte(Sc.Getbyte(myByte)))
Next
Sc.Print(Chr(13))
End If
Next
Else
Sc.Print("NOT FOUND! Try with different C1 38!")
End If
Wx.Closeport
End Sub

```

Phase 5C - Utilisation d'une instruction CUSTOM pour générer une ECM

Les instructions C1 3C ont la particularité d'avoir le premier quadruple mot (8 octets) qui ne subit pas de SUPER-ENCRYPTION, voulant construire une CUSTOM, on doit tenir compte de ce fait. En sachant que dans une commande, s'il y a pas de données à décrypter, la carte ne répond pas ou répond avec l'ATR, on en conclut que pour construire une 3C, il faut respecter une LEN minimum, pour laquelle en phase de PARSING, nous ayons au moins 9 octets après P3 et ses données, et avant la signature.

Il est donc impossible d'obtenir une ECM CUSTOM valide à partir d'une réponse courte (LEN = 0x13).

La méthode pour obtenir un C1 3C CUSTOM valide et celle de chercher une réponse à l'instruction C1 36 dans laquelle la LEN soit au moins 0x1C, dans laquelle P3 et P4 soit admissibles et dans laquelle le quartet faible de P2 soit supérieure ou égal à 0x0C (autrement Statut 904A).

La construction des ECM CUSTOM suit le même raisonnement de la construction des EMM, l'important est de se rappeler que le tableau des NLB_LB de la C1 3C est différent de celui des C138/40.

Dans la construction des instructions C1 3C à partir d'une réponse longue, nous avons le déclenchement du bug pour $30 \leq P3 \leq 9F$ (ou P3 doit être entre 30 compris et 9F compris).

La formule justifie le même comportement de la C1 38, mais pour des valeurs différentes de P3 à cause du premier quadruple mot (8 octets) non crypté.

A ce point on vérifie comment doit valoir la suivante affirmation :

Avec les instructions longues, si le bug apparaît, le statut à 90 00 ne perdure pas (Statut instable)

Osservazione 2:

Il bug 9600 a un'ulteriore effetto sulle C1 3C "custom" : il risultato dell'unmasking va a sostituire interamente gli 8 byte pre-signature, sovrascrivendo alcuni byte dell'ottetto in chiaro (*vedi anche pag. 79*). Ad esempio, con questa INS non si ottiene lo status 9A00, pur *sembrando* avere parsing corretto ed un nano 24 "proibito"...

Ceci signifie qu'en envoyant, en répétition, la même instruction, sans faire de RESET, et sans modification d'octets, on obtiens des statuts différents. Ceci signifie qu'à chaque envoi, le changement de quelques conditions provoque un UNMASKING différent.

Observation 1 :

L'instabilité du statut peut être constaté avec les C1 38/40, toujours en utilisant les réponses longues et avec des valeurs de P3 tels à obtenir moins de 8 octets de données à décrypter.

Observation 2

Le bug 9600 a un effet ultérieure sur les C1 3C CUSTOM, le résultat de l'UNMASKING va substituer entièrement les 8 octets pré-signature, réécrivant quelques octets du quadruple mot (8 octets) en clair. Par exemple, avec cette instruction nous n'obtiendrons jamais 9A00, même s'il semble avoir un PARSING correct et une NANO 24 interdite

```
C1 3C 21 9E 76
8F F1 36 3A D0 A9 41 FE B3 27 1A 81 31 16 80 02
24 00 80 82 EA BC E5 28 50 CA C5 3A 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 63
```

WELCOME à LAS VEGAS – probabilité d'avoir un PARSING correct.
(D'une étude adaptée faite en terre étrangère)

Après les voix insistantes de programmes miraculeux, j'ai décidé de comprendre s'il y avait d'autres bugs, non encore découverts par les gens ou si effectivement toutes les informations étaient du domaine public.

J'ai été surpris par le résultat obtenu : les conditions de PARSING correct posent une restriction très forte sur les instructions générées de manière casuelles.

Faisons l'hypothèse que les propriétés statistiques des octets en clair et celles des octets cryptés soient les mêmes (c'est à dire que la SUPER-ENCRYPTION jouisse d'assez de propriété de dispersion).

Prenant une commande longue n , il serait intéressant de découvrir toutes les combinaisons de NANOCOMMANDE SECA qui peuvent se générer dans la commande, de telle façon à avoir un PARSING correct et donc l'exécution par la carte. Toutes les possibilités peuvent être calculées en appliquant récursivement une « forme grammaticale » dans laquelle le symbole n est exprimé par $n \rightarrow (n-1) \& 0$, ou en d'autres termes, la NANO Mx peut être substituée par la NANO $(M-1)x$ plus la NANO $0w$.

En appliquant ce que nous venons de dire à une NANO quelconque « $Nx...yy...yy$ » (NANOS + données) et tenant compte de la non expansibilité de la NANO $0w$, nous pouvons construire toutes les possibilités (NOTE : dans la tractation ne seront pas considérés les NANOCOMMANDES Dx , Ex et Fx , mais avec quelques considérations additionnelles, ils peuvent rentrer dans la procédure).

On définit « TYPE » cette structure, en conséquence une instruction TYPE110 aura cette structure :

1x mm 1y nn 0z

Inversement une INS TYPE100

1x mm 0z 0y

En considérant cette dernière structure, combien d'instructions pouvons-nous construire ? Ayant 3 octets avec un quartet fixe et un octet à valeur quelconque, il résultera :

$$T = 16^m * 256^{(n-m)}$$

Où :

m est le nombre de NANOCOMMANDE de TYPE (3 dans ce cas)

n est la longueur de l'instruction

$(n-m)$ est le nombre d'octets à valeurs quelconques.

L'expression précédente ne rend pas compte de toutes les possibles permutations des NANOCOMMANDES. Cette propriété de permutation se définit multiple. Nous devons donc calculer les permutations avec répétition de k éléments (c'est à dire de NANOCOMMANDES) ayant k_i pour chaque type :

$$PERM(K, (K_1, K_2, ..., K_N)) = K! / ((k_1!) * (k_2!) * ... * (k_n!))$$

Dans le cas traité TYPE 110, nous aurons :

$$\text{PERM}(3, (2, 1)) = 3! / (2! * 1!) = 3$$

En effet les possibles résultats des permutations sont « 110, 101, 011 ». Avec tout ce que nous avons énoncé, nous pouvons maintenant calculer la probabilité sur chaque TYPE sur une instruction générée casuellement et conséquemment la probabilité d'avoir un PARSING correct. Dans le tableau suivant, on pourra observer, les probabilités par donnée de l'instruction, de longueur variable de 2 à 9 octets.

LEN = 2

TYPE	COUNT	MULTIP	TOTAL	%	% PARSING OK
1	4096	1	4096	6.25 %	94.12 %
00	256	1	256	0.39%	5.88%
TOTAL	65536		4352	6.64%	

LEN = 3

TYPE	COUNT	MULTIP	TOTAL	%	% PARSING OK
2	1048576	1	1048576	6.25%	88.89%
10	65536	2	131072	0.78%	11.11%
000	4096	1	4096	0.02%	0.35%
TOTAL	65536		1179648	7.03%	

LEN = 4

TYPE	COUNT	MULTIP	TOTAL	%	% PARSING OK
3	268435456	1	268435456	6.25%	83.37%
20	16777216	2	33554432	0.78%	10.42%
11	16777216	1	16777216	0.39%	5.21%
100	1048576	3	3145728	0.07%	0.98%
0000	65536	1	65536	0.00%	0.02%
TOTAL	4294967296		321978368	7.50%	

LEN = 5

TYPE	COUNT	MULTIP	TOTAL	%	% PARSING OK
4	68719476736	1	68719476736	6.25%	78.47%
30	4294967296	2	8589934592	0.78%	9.81%
21	4294967296	2	8589934592	0.78%	9.81%
200	268435456	3	805306368	0.07%	0.92%
110	268435456	3	805306368	0.07%	0.92%
1000	16777216	4	67108864	0.01%	0.08%
00000	1048576	1	1048576	0.00%	0.00%
TOTAL	1,09951E+12		87578116096	7.97%	

LEN = 6

TYPE	COUNT	MULTIP	TOTAL	%	% PARSING OK
5	1,75922E+13	1	1,75922E+13	6.25%	74.08%
40	1,09951E+12	2	1,75922E+12	0.78%	9.26%
31	1,09951E+12	2	1,75922E+12	0.78%	8.26%
22	1,09951E+12	1	1,09951E+12	0.39%	4.63%
300	68719476736	2	1,37439E+11	0.05%	0.58%
210	68719476736	6	4,12317E+11	0.15%	1.74%
111	68719476736	1	68719476736	0.02%	0.29%
2000	4294967296	3	12884901888	0.00%	0.05%
1100	4294967296	6	25769803776	0.01%	0.11%
10000	268435456	5	1342177280	0.00%	0.01%
000000	16777216	1	16777216	0.00%	0.00%
TOTAL	2,81475E+14		2,37482E+13	8.44%	

LEN = 7

TYPE	COUNT	MULTIP	TOTAL	%	% PARSING OK
6	4,50E+15	1	4,50E+15	6.25%	69.51%
50	2,81475E+14	2	5,63E+14	0.78%	8.69%
41	2,81475E+14	2	5,63E+14	0.78%	8.69%
32	2,81475E+14	2	5,63E+14	0.78%	8.69%
400	1,75922E+13	3	5,28E+13	0.07%	0.81%
310	1,75922E+13	6	1,06E+14	0.15%	1.63%
220	1,75922E+13	3	5,28E+13	0.07%	0.81%
211	1,75922E+13	3	5,28E+13	0.07%	0.81%
3000	1,09951E+12	4	4,40E+12	0.01%	0.07%
2100	1,09951E+12	12	1,32E+13	0.02%	0.20%
1110	1,09951E+12	4	4,40E+12	0.01%	0.07%
20000	68719476736	5	3,44E+11	0.00%	0.01%
11000	68719476736	10	6,87E+11	0.00%	0.01%
100000	4294967296	6	2,58E+10	0.00%	0.00%
0000000	268435456	1	2,68E+08	0.00%	0.00%
TOTAL	7,20576E+16		6,48E+15	8.99%	

LEN = 8

TYPE	COUNT	MULTIP	TOTAL	%	% PARSING OK
7	1,15E+18	1	1,15E+18	6.25%	65.42%
60	7,21E+16	2	1,44E+17	0.78%	8.18%
51	7,21E+16	2	1,44E+17	0.78%	8.18%
42	7,21E+16	2	1,44E+17	0.78%	8.18%
33	7,21E+16	1	7,21E+16	0.39%	4.09%
500	4,50E+15	3	1,35E+16	0.07%	0.77%
410	4,50E+15	6	2,70E+16	0.15%	1.53%
320	4,50E+15	6	2,70E+16	0.15%	1.53%
311	4,50E+15	3	1,35E+16	0.07%	0.77%
221	4,50E+15	3	1,35E+16	0.07%	0.77%
4000	2,81475E+14	4	1,13E+15	0.01%	0.06%
3100	2,81475E+14	12	3,38E+15	0.02%	0.19%
2200	2,81475E+14	6	1,69E+15	0.01%	0.10%
2110	2,81475E+14	12	3,38E+15	0.02%	0.19%
1111	2,81475E+14	1	2,81E+14	0.00%	0.02%
30000	1,75922E+13	5	8,80E+13	0.00%	0.00%
21000	1,75922E+13	10	1,76E+14	0.00%	0.01%
11100	1,75922E+13	10	1,76E+14	0.00%	0.01%
200000	1,09951E+12	5	5,50E+12	0.00%	0.00%
110000	1,09951E+12	15	1,65E+13	0.00%	0.00%
1000000	68719476736	7	4,81E+11	0.00%	0.00%
00000000	4294967296	1	4,29E+09	0.00%	0.00%
TOTAL	1,84467E+19		1,76E+18	9.55%	

LEN = 9	TYPE	COUNT	MULTIP	TOTAL	%	% PARSING OK
	8	2,95E+20	1	2,95E+20	6.25%	51.58%
	70	1,84E+19	2	3,69E+19	0.78%	7.70%
	61	1,84E+19	2	3,69E+19	0.78%	7.70%
	52	1,84E+19	2	3,69E+19	0.78%	7.70%
	43	1,84E+19	2	3,69E+19	0.78%	7.70%
	600	1,15E+18	3	3,46E+18	0.07%	0.72%
	510	1,15E+18	6	6,92E+18	0.15%	1.44%
	420	1,15E+18	6	6,92E+18	0.15%	1.44%
	330	1,15E+18	3	3,46E+18	0.07%	0.72%
	411	1,15E+18	3	3,46E+18	0.07%	0.72%
	321	1,15E+18	6	6,92E+18	0.15%	1.44%
	222	1,15E+18	1	1,15E+18	0.02%	0.24%
	5000	7,21E+16	4	2,88E+17	0.01%	0.06%
	4100	7,21E+16	12	8,65E+17	0.02%	0.18%
	3200	7,21E+16	12	8,65E+17	0.02%	0.18%
	3110	7,21E+16	12	8,65E+17	0.02%	0.18%
	2210	7,21E+16	12	8,65E+17	0.02%	0.18%
	2111	7,21E+16	4	2,88E+17	0.01%	0.06%
	40000	4,50E+15	5	2,25E+16	0.00%	0.00%
	31000	4,50E+15	10	4,50E+16	0.00%	0.01%
	22000	4,50E+15	10	4,50E+16	0.00%	0.01%
	21100	4,50E+15	30	1,35E+17	0.00%	0.03%
	11110	4,50E+15	5	2,25E+16	0.00%	0.00%
	300000	2,81475E+14	6	1,69E+15	0.00%	0.00%
	210000	2,81475E+14	30	8,44E+15	0.00%	0.00%
	111000	2,81475E+14	20	5,63E+15	0.00%	0.00%
	2000000	1,75922E+13	7	1,23E+14	0.00%	0.00%
	1100000	1,75922E+13	21	3,69E+14	0.00%	0.00%
	10000000	1,09951E+12	8	8,80E+12	0.00%	0.00%
	000000000	68719476736	1	6,87E+10	0.00%	0.00%
	TOTAL	4,72237E+21		4,79E+20	10.15%	

Au vu des chiffres de ces tableaux, nous pouvons extrapoler différentes choses :
 premièrement nous voyons que la probabilité d'avoir un PARSING correct varie de 6.46 % pour une instruction avec une longueur de 2 octets à 10.15% pour une instruction avec une longueur de 10 octets, une donnée rencontrée aussi par expérimentation. Une autre propriété importante que nous en retirons, est la suivante :

Pour une instruction de longueur (N+1), la probabilité d'avoir la NANO Nx est beaucoup plus grande que n'importe quelle configuration à PARSING correct.

En conclusion, si nous assumons que l'instruction avec TYPE, qui inclus « 1, 0/1, 0y » sont les plus dangereuses et en excluant de générer des effacements multiples de clés et/ou la déplacement d'un PROVIDER (NANO 25, cette dernière étant admissible uniquement pour SECA), il est possible d'estimer ma proportion entre la probabilité de produire des instructions aux effets **négatifs** et la probabilité de produire des effets **positifs**.

Enfin le mystère de la génération des NANOS est révélé et cela est un grand pas en avant, parce que nous pouvons maintenant utiliser d'autres NANOCOMMANDES (différents des habituelles 80, 41 ou 21) qui pourront nous aider à mieux comprendre le fonctionnement de la carte. La situation est analogue à un modèle relatif à un modèle plus contrôlable disons classique, ou le déterminisme nous permet de prévoir le résultat de nos actions.

SUPER-ENCRYPTION EN SECA2

On étés définies de nombreux modèle en regard du fonctionnement de la SUPERECRYPTION, au croisement et à la fusion de ces théories, naît un modèle que nous exposerons ensuite, mais avant il est opportun de démarrer et d'analyser avec une série d'observations expérimentales.

A ce propos on prend en considération les réponses longues 0x24 ; associations un *nom* à chaque octet crypté.

C1 36 P1 P2 LEN

36

24 c00 c01 c02 c03 c04 c05

c06 c07 c08 c09 c0A c0B c0C

c0D c0E c0F c10 c11 c12 c13

c14 c15 c16 c17 c18 c19 c1A

82 + signature

Et les octets correspondants, en clairs

p00 p01 p02 p03 p04 p05

p06 p07 p08 p09 p0A p0B p0C

p0D p0E p0F p10 p11 p12 p13

p14 p15 p16 p17 p18 p19 p1A

Construisons un tableau “ octets en clairs” VS “octets cryptés”, ou reporter la variation des octets cryptés en fonction des octets en clairs (■ = varié , ■ = pas varié).

[illegible]

De l'analyse par ligne on note une organisation des octets en clairs par quadruple mot

1. p00---p07 : le contenu de ces 8 octets conditionne c00---c07 e c13---c1A
2. p08---p0F : le contenu de ces 8 octets conditionne c00---c0F e c13---c1A
3. p10---p17 : le contenu de ces 8 octets conditionne c00---c12 e c13---c1A

Avec les commandes longues 0x24, on ne réussi pas à faire varier les derniers octets, mais en faisant les mêmes essais sur des commandes de longueurs différentes on peut conclure que :

4. p18---p1A : le contenu de ces trois octets conditionne c00---c12 e c13---c1A

Première observation : les derniers 8 octets changent toujours, peu importe l'octet que nous faisons varier.

Tenons à part cette considération, qui nous sera bien utile plus tard, et considérons les points 1 et 2.

		C C C C C C C C	C C C C C C C C	C C C	C C C C C C C C
		0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 1 1	1 1 1 1 1 1 1 1
		0 1 2 3 4 5 6 7	8 9 A B C D E F	0 1 2	3 4 5 6 7 8 9 A
-----+					
(...)					
p07		* * * * * * * *	* * * * * * * *
p08		* * * * * * * *	* * * * * * * *	. . .	* * * * * * * *
(...)					

En analysant par colonne, il semblerait qu'il y ai une subdivision en quadruple mot (8 octets) aussi pour les données cryptées. Nous pouvons faire l'hypothèse d'une séquence logique d'opération, vu que le contenu de p08---p0F influence c00---c0F, tandis que le contenu de p00---p07 influence c00---c07 mais pas c08---c0F, nous pouvons donc dire que :

- Le CRYPT de (p08---p0F) advient **avant** le CRYPT de (p00---p07)
- Le CRYPT de (p00---p07) dépend du CRYPT (p08---p10)

NB : pour l'instant on ne fait aucune sur quels sont les opérants du CRYPT. L'écriture CRYPT() est à interpréter comme une générique « fonction de chiffrement de données »

Observons le point 3 : dans ce cas tous les octets changent. Du processus décrit précédemment on pourrait par induction affirmer :

- Le CRYPT de (p10---p17) advient avant le CRYPT de (p08---p0F)
- Le CRYPT de (p08---p0F) dépend du CRYPT (p10---p17)

Mais entre le point 2 et 3, il n'y a pas un quadruple mot de différence mais seulement trois octets

	C C C C C C C C	C C C C C C C C	C C C	C C C C C C C C
	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 1 1	1 1 1 1 1 1 1 1
	0 1 2 3 4 5 6 7	8 9 A B C D E F	0 1 2	3 4 5 6 7 8 9 A
(...)				
p0F	* * * * * * * *	* * * * * * * *	. . .	* * * * * * * *
p10	* * * * * * * *	* * * * * * * *	* * *	* * * * * * * *
(...)				

Si vous observez le quadruple mot p10---p17 il est partiellement superposé au quadruple mot p13---p1A (les 8 octets de la pré-signature) : ces 8 derniers octets sont variables à chaque nouvelle condition (voir tableau). Nous avons une donnée de fait : c10---c12 bien que ne faisant pas partie d'un quadruple mot, ne sont pas restés en clairs, signe que le processus de CRYPT s'applique aussi d'eux, mais de quel façon ? .

Formulons trois hypothèses :

- Hypothèse 1 : Cela pourrait être CRYPT(p10 p11 p12 00 00 00 00 00)
- Hypothèse 2 : Cela pourrait être CRYPT (p10 p11 p12 p13 p14 p15 p16 p17)
- Hypothèse 3 : Cela pourrait être CRYPT (p10 p11 p12 x13 x14 x15 x16 x17)

La dernière hypothèse signifie que le CRYPT utilise pour les octets 13-17 , ceux résultants du CRYPT (p13---p1A)

Nous avons un indice, c'est que c10---c12 varient si on fait varier n'importe quel octet de p10---p17 et nous pouvons donc éliminer la première hypothèse. Nous pouvons aussi éliminer la troisième hypothèse, car avec cette méthode les octets c10---c12 varieraient en modifiant un quelconque octet en clair.

Donc ce que nous avons affirmé précédemment devrait être vrai :

- Le CRYPT de (p10---p17) advient **avant** le CRYPT de (p08---p0F)
- Le CRYPT de (p08---p0F) dépend du CRYPT (p10---p17)

De plus on peut observer que :

- Le CRYPT des 8 octets pré-signature advient **APRES** le CRYPT(p10---p17)

Il est raisonnable de penser que c'est le dernier du processus. Mais en analysant le point 4, on en déduit que les octets p18---p1A doivent entrer en jeu dans la phase initiale du processus.

- Hypothèse A : les octets restent initialement en clairs
- Hypothèse B : les octets subissent un CRYPT initiale de (p18 p19 p1A p00 p01 p02 p03 p04)
- Hypothèse C : Les octets subissent un CRYPT initiale de (p18 p19 p1A 00 00 00 00 00)

Observation : quand on parle de CRYPT de quadruple mot incomplet , on suppose le rajout des octets manquants par des valeurs préétablies.

Ce processus se nomme PADDING, habituellement on adopte le PADDING avec des valeurs zéro.

Il faut dire que ce n'est pas le seul type de PADDING dans la théorie des algorithmes de chiffrement.

Par exemple dans l'algorithme IRDETO 1 on peut lire ceci :

IRDETO (citation)

La préparation du paquet à dé(crypter) s'obtient par la subdivision en blocs de 8 octets chaque un.

Bloc 1 : octet 0...7

Bloc 2 : octet 8...15

Et ainsi de suite jusqu'au dernier blocs qui bien sur ne devra pas obligatoirement se terminer avec la redondance de 8 octets (en clair il peut être plus petit que 8 octets) et dans le cas ou cela arriverait, il faut ajouter les octets hexadécimaux 0x61, 0x62 , 0x63, 0x64, 0x65, 0x66, 0x67, autant que nécessaire pour arriver au nombre de 8 octets aussi dans le dernier bloc.

Il est possible de démontrer (et nous le ferons après avoir décrit le modèle) la validité des informations qui suivent :

- p18 p19 p1A restent initialement en clair
- le CRYPT (p10---p17) dépend de (p18 p19 p1A) en clair

Résumant le tout nous avons :

- p18 p19 p1A restent en clair
- Le CRYPT(p10---p17) dépend de (p18 p19 p1A)
- Le CRYPT (p10---p17) advient **avant** le CRYPT de (p08---p0F)
- Le CRYPT (p08---p0F) dépend du CRYPT (p10---p17)
- Le CRYPT (p08---p0F) advient **avant** le CRYPT de (p00---p07)
- Le CRYPT (p00---p07) dépend du CRYPT de (p08---p10)
- il processus se conclut avec CRYPT (c13---c17 p13---p1A)

Ces observations ne sont pas suffisantes pour décrire le comportement de la SUPER-ENCRYPTION, il faut y ajouter d'autres considérations :

SUPER-ENCRYPTION (citation)

On observe par expérimentation que certaines combinaisons d'octets en clairs donnent le premier quadruple mot (8 octets) changé, mais pas les huit octets pré-signature.

Exemple 1 (LEN = 0x1C) :

D5 08 ED F1 75 DC 6B AD 61 AB 15 00 1B CE D7 71 B5 B8 60 82 + signature
0E AE 07 2C 59 FE F4 AA 61 AB 15 00 1B CE D7 71 B5 B8 60 82 + signature

Exemple 2 (LEN = 0x1C) :

B8 1F 69 B8 25 0C 90 E0 61 AB 15 01 FA 65 31 45 4B 0C CC 82 + signature
FF 95 4A CE D8 DD 74 83 61 AB 15 01 FA 65 31 45 4B 0C CC 82 + signature

Exemple 3 (LEN = 0x1C)

3E 91 CE 26 D4 E5 54 AA 61 AB 15 02 01 BA D9 28 F9 BB F4 82 + signature
C4 F7 1F B8 03 8B 0E 0C 61 AB 15 02 01 BA D9 28 F9 BB F4 82 + signature

SUPERENCRYPTION (citation)

En numérotant les quadruple mots (rappel 8 octets par quadruple mot), de 1 à N, on considère externes les octets contenus à l'intérieur des quadruple mots de 1 à N -1.

(...)

Quand on fait varier uniquement les octets externes, on a que le nombre des résultats du DECRYPT du dernier quadruple mot, est limité à seulement 16384 combinaison.

La dernière citation est illuminante : 16384 (0x4000 en hexadécimal) est justement la valeur utilisée lorsqu'on va exécuter le modulo à l'intérieur de la « somme des mots », avec une vérification rapide, on trouve le lien pour chaque couple d'exemple exposées avant

(D5 08 + ED F1 + 75 DC + 6B AD) MOD 4000 = 2482

(0E AE + 07 2C + 59 FE + F4 AA) MOD 4000 = 2482

(B8 1F + 69 B8 + 25 0C + 90 E0) MOD 4000 = 17C3

(FF 95 + 4A CE + D8 DD + 74 83) MOD 4000 = 17C3

(3E 91 + CE 26 + D4 E5 + 54 AA) MOD 4000 = 3646

(C4 F7 + 1F B8 + 03 8B + 0E 0C) MOD 4000 = 3646

Le CRYPT du dernier quadruple mot semble dépendre de la « somme mots modulo 0x4000 »
du premier quadruple mot

En utilisant des réponses plus longues (LEN = 0x24), nous pouvons ajouter d'autres considérations :

E7 1F 73 BA 5A F5 AD 81 E1 C5 7A DE CA 5C 3B 77

9F CC F0 00 C7 3D 11 A8 7F 73 2C 82 + signature

B5 F1 65 F5 E0 B8 66 B1 C4 C0 95 78 F0 1D 98 21

9F CC F0 00 C7 3D 11 A8 7F 73 2C 82 + signature

Ces deux réponses ont le même quadruple mot pré-signature

$(E71F + 73BA + 5AF5 + AD81 + E1C5 + 7ADE + CA5C + 3B77) \text{ MOD } 4000 = 05C5$

$(B5F1 + 65F5 + E0B8 + 66B1 + C4C0 + 9578 + F01D + 9821) \text{ MOD } 4000 = 05C5$

et la même somme « mots » étendue à tout le corps de l'instruction (à l'exclusion des derniers 8 octets bien sur)

Le CRYPT du quadruple mot pré-signature dépend de la « somme mots modulo 0x400 » des données restantes.

On obtiens un résultat analogue en focalisant son attention sur le premier et le second quadruple mot d'une réponse longue :

4A 23 B4 AE EE 88 53 8A 47 55 18 7C 8F 14 19 47
1E 21 A4 62 18 28 29 44 E4 DA D1 82 + SIGNATURE

4A 23 B4 AE EE 88 53 8A DE 8E F0 7F CC E7 6C 38
1E 21 A4 62 18 28 29 44 E4 DA D1 82 + SIGNATURE

$(4755+187C+8F14+1947) \text{ MOD } 4000 = 082C$
 $1082C \text{ MOD } 4000 = 082C$

$(DE8E+F07F+CCE7+6C38) \text{ MOD } 4000 = 082C$
 $3082C \text{ MOD } 4000 = 082C$

Le CRYPT du premier quadruple mot dépend de la «somme mots MOD 4000 » du second quadruple mot.

Le règle peut être étendue à n'importe quel quadruple mot de la commande cryptée.

Le CRYPT d'un quadruple mot dépend de la « somme mots modulo 0x400 » qu quadruple mot se trouvant à sa droite.

Il faut noter que le premier quadruple mot à crypter / décrypter, n'ayant pas de quadruple mot à sa droite utilisera une « somme mot » particulière, générée en phase d'initialisation de l'algorithme.

Maintenant il est possible de formuler une théorie possible de fonctionnement, mais avant nous devons définir les termes suivants : CYPHER et MASKING.

CYPHER : C'est le cryptage classique SECA1, les paramètres à passer en entrée sont, le quadruple mot à crypter et la clé utilisé (indiqué par le quartet faible de P2 et le bit 4 de P1. Il est de plus, possible de sélectionner les tables HARSH pour le CYPHER (par les bits 6 et 5 de P1).

MASKING : Est un procédé de cryptage à algorithme inconnu (certains tests laisseraient penser à une opération XOR, avec des valeurs de tables, associées à une permutation, mais nous n'avons aucunes certitudes), les paramètres à passer en entrée sont le quadruple mot à masquer et la clé utilisée. Cette dernière est formée de 14 bits et est construite à partir d'un quadruple mot de données différentes (tout au moins dans l'implémentation théorique... puis il y a les BUGS) de celui à masquer. [La clé se construit en faisant la « somme mots modulo 0x4000 ».](#)

Note : pour des raisons historiques nous avons maintenus le terme MASKING, bien qu'il serait plus juste de parler de CHAINING à la lumière de dernières mise à jour su modèle SUPER-ENCRYPTION

SUPER-ENCRYPTION – Un modèle possible (dernière mise à jour 21/05/2003).

Le processus peut être divisé en trois parties :

- Initialisation
- En-cryptage du corps de l'instruction
- En-cryptage de la queue (Quadruple mot pré-signature)

On prend le corps de l'instruction jusqu'à la NANO 82 exclue et on le divise en blocs de huit octets en partant de la gauche vers la droite. Si nous avons un quadruple mot qui n'a pas les huit octets, ceux-ci seront défini comme octets restants.

Une fois effectuée la division en blocs de 8 octets, on numérote chaque bloc en partant de celui qui est le plus à droite en continuant vers la gauche (on numérote à partir de 0) définissons quelques termes par commodité :

(Ai) : Quadruple mot énième en clair
(Mi) : Résultat du MASKING du quadruple mot énième
(Si) : Somme MOTS modulo 0x4000 à utiliser pour le MASKING
(Ci) : Résultat du CYPHER du quadruple mot énième
'key' : La clé utilisée pour le CYPHER (la clé normale du SECA)
(LB) : Les huit octets pré-signature
(BS) : Les octets restants
(A_init) : Quadruple mot généré en phase d'initialisation
(M_init) : Résultat du MASKING de (A_init)
(C_init) : Résultat du CYPHER de (M_init)
(MF) : Résultat du MASKING du (LB)
(CF) : Résultat du CYPHER du (MF)

Phase 1 : initialisation

On vérifié la présence d'octets restants :

- Si oui

```
A_init = padding(BS)
```

```
M_init = mask(A_init, 0000)
```

```
C_init = cypher(M_init, key)
```

- Si non

```
C_init = 00 00 00 00 00 00 00 00
```

On fait l'hypothèse d'un PADDING composé de zéro. Le quadruple mot (**C_init**) sera utilisé pour le **calcul de (S0)**, mais ne substitue aucune partie du texte à crypter : les octets restants éventuellement présent restent en clairs

Phase 2 : ENCRYPTION du corps de l'instruction

Cette phase est applicable des quadruples mots entiers (donc qui se compose de 8 octets) et est constituée du masquage et du cryptage avec l'algorithme SECA1 (MASK&CYPHER). Le masquage dépend du quadruple mot adjacent (à sa droite), à travers le désormais connu « somme mots modulo 0x4000 ». On réalise ainsi un mécanisme de chiffrement à chaîne (CYPHER BLOCK CHAINING).

```
Si = sum(Mi-1) Pour i=0 vaut la somme faite sur le quadruple mot(C_init)
```

```
Mi = mask(Ai, Si)
```

```
Ci = cypher(Mi, key)
```

Pour actualiser ce processus il est nécessaire d'exécuter « d'abord le MASKING puis le CYPHER » quadruple mot par quadruple mot, c'est à dire qu'on ne peut séparer les deux procédés, comme dans la théorie de d'abord CYPHER puis MASKING proposé dans le passé.

Phase 3 : ENCRYPTION de la queue

```
SF = sum(CYPHERTEXT-8)
```

```
MF = mask(LB, SF)
```

```
CF = cypher(MF, key)
```

La somme SF utilise tous les octets du corps de l'instruction (qui sont maintenant cryptés) à l'exclusion des huit derniers.

L'implémentation dans la carte de ce processus est affectée de bug (on peut avoir un **pointer underflow**).

Le modèle est applicable avec succès sur n'importe quel nombre d'octets supérieur ou égal à 8 (et donc aussi sur de simple quadruple mot). Si les données à dé(crypter) sont exactement 8, la phase 2 n'est pas exécutée et la somme SF vaut 0000.

SUPERENCRYPTION – vérification expérimentales

Le modèle étant identifié, il est possible d'analyser certains test pour le valider (revue de post présents sur des forums)

Preuve 1 – élaboration de (LB)

On construit une C1 38 (à partir d'une réponse courte) capable de générer une réponse longue 0x24 par exemple la suivante :

C1 38 21 91 76

```
75 91 1F 6F 72 74 34 77 C9 1A 82 00 1C 90 07 23
B4 E0 5C 00 00 00 00 00 00 00 00 yy 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 09
```

A travers elle, on tire 256 réponses longues 0x24 et 256 réponses courtes : il s'agit de faire une variation de l'octet yy et d'envoyer deux couples de C1 38/36 avec LEN de la réponse différente

```
C1 38 21 91 76
75 91 1F 6F 72 74 34 77 C9 1A 82 00 1C 90 07 23
B4 E0 5C 00 00 00 00 00 00 00 yy ...
```

C1 36 21 91 14
... et ...

```
C1 38 21 91 76
75 91 1F 6F 72 74 34 77 C9 1A 82 00 1C 90 07 23
B4 E0 5C 00 00 00 00 00 00 00 yy ...
```

C1 36 21 91 25

La forme en clair des réponses est celle-ci (la première c'est la courte et la seconde la longue)

86 E0 5C 00 00 00 00 00

yy 03 82 + SIGNATURE

86 E0 5C 00 00 00 00 00

```
yy D2 00 1F 5D FF FF FF
FF FF FF FF FF 89 00 81
00 00 03 82 + SIGNATURE
```

Bien sûr que des cartes différentes peuvent donner des réponses différentes, selon la configuration des enregistrements, mais à la fin du résultat, il n'y a plus de différences.

Observation : de la façon dont ont été construite les réponses, le premier quadruple mot en clair, sera le même pour toutes.

Des 256 + 256 réponses générées, on extrait un couple long + courte de tel façon d'avoir les premiers deux octets identiques, en général les deux réponses auront une valeur différente pour yy

Réponses Cryptée	Réponse en Clairs
23 1E EA 08 17 F0 61 10 ED 4E 82 E7 91 26 7E 62 B0 70 F8	86 E0 5C 00 00 00 00 00 67 03 82 + SIGNATURE
23 1E E8 D5 5E E5 A2 3F E1 C5 7A DE CA 5C 3B 77 9F CC F0 10 45 0E A3 0F 48 84 88 82 41 28 39 1C 1B 66 D1 06	86 E0 5C 00 00 00 00 00 7B D2 00 1F 5D FF FF FF FF FF FF FF FF 89 00 81 00 00 03 82 + SIGNATURE

Dans ce cas nous avons yy = 7B pour la longue et yy = 67. il est maintenant nécessaire d'expliquer ce qu'implique le fait d'avoir les deux premiers octets identiques.

Si vous pensez à la subdivision en quadruple mot de l'instruction et à la façon d'opérer de la SUPERENCRYPTION : les deux réponses ont le premier octet en clair, mais (en général) CRYPT différent par effet du MASKING.

Dans la réponse courte, les six derniers octets du premier quadruple mot font partie des octets pré-signature élaborés pendant la phase 3 de la SUPERENCRYPTION, seul les deux premiers octets restent en dehors, c'est justement eux que nous observons : dans le cas où le MASKING appliqué au premier quadruple mot donnerait le même résultat pour les deux commande, alors se serait justement les deux premiers octets qui serait identique (à parité de CYPHER).

Pour avoir le même MASKING, la « somme mot » utilisé pour le masquage, devront être identiques.

Maintenant, si nous faisons varié le premier quadruple mot, nous obtiendrions des CRYPT différent des précédent, mais toujours ayant en commun les deux premiers octets. Cette propriété nous servira plus tard.

On construis maintenant une réponse longue qui ai le troisième octet égal à 86 (on l'obtiens en faisant une variation des octets compris entre la signature et yy, avec yy = 7B).

On génère en plus une réponse courte avec yy = 67 et même quadruple mot de la longue.

Réponses Cryptée	Réponse en Clairs
70 56 05 B6 20 15 DF 76 FA B4 82 8E 46 A2 5F F5 C0 00 15	86 E0 5C 00 45 6B 00 00 67 03 82 + SIGNATURE
70 56 86 74 79 F9 D6 DF 46 FB CD 8A 11 36 25 A9 2C 49 D5 F3 76 3B BA 4B 6C 40 6E 82 30 8E C5 91 16 28 1A 4B	86 E0 5C 00 45 6B 00 00 7B D2 00 1F 5D FF FF FF FF FF FF FF FF 89 00 81 00 00 03 82 + SIGNATURE

On peut noter l'égalité de valeur des deux premiers octets, en accord avec la propriété décrite précédemment.

A quoi sert l'octet fixé à la valeur 86 ?

Analysons le processus de la SE appliqué à la réponse courte.
Nous aurons les étapes suivantes :

- MASKING (86 E0 5C 00 45 6B 00 00) et CYPHER du résultat
- Elaboration du quadruple mot pré-signature que nous appellerons (LB) par commodité

Ce que nous voulons démontrer, c'est que l'élaboration du (LB) consiste en un MASK & CYPHER.

On fait la démonstration en analysant la réponse longue, et en particulier le premier quadruple mot, duquel nous connaissons la valeur en clair, ainsi que sa valeur sous MASK & CYPHER :

MASK & CYPHER (86 E0 5C 00 45 6B 00 00) = 70 56 86 74 79 F9 D6 DF

"70 56" correspond (par construction) avec les deux premiers octets de la réponse courte.
Les autres sont en général différents par effet de l'élaboration du (LB).

On suppose que le processus soit le suivant :

Données en clairs

86 E0 5C 00 00 00 00 00
67 03 82 + SIGNATURE

(86 E0 5C 00 45 6B 00 00) 67 03 82 + SIG. -> en clair

(70 56 86 74 79 F9 D6 DF) 67 03 82 + SIG. -> après le premier MASK&CYPHER (hypothèse)

(70 56 05 B6 20 15 DF 76) FA B4 82 + SIG. -> après l'élaboration du quadruple mot pré-signature

Hypothèse : vu l'égalité des deux premiers octets avec ceux de la réponse longue, on peut raisonnablement supposer que le résultat du MASK&CYPHER de : 86 E0 5C 00 45 6B 00 00 donne le même résultat de la réponse longue

Données cryptées :

70 56 05 B6 20 15 DF 76
FA B4 82 8E 46 A2 5F F5
C0 00 15

Dans la réponse longue le MASKING a utilisé la « somme mots » suivant :

$$(46 \text{ FB} + \text{CD } 8\text{A} + 11 \text{ } 36 + 25 \text{ A9}) \bmod 4000 = 0\text{B}64$$

Cette valeur est différente de la somme des octets restants de la réponse courte :

$$6703 \bmod 4000 = 2703$$

Nous avons la confirmation de la non utilisation par le MASKING du premier quadruple mot, des octets restants en clair.

Reprenons le processus SE appliqué à la réponse courte, cette fois en mettant en valeur le quadruple mot pré-signature :

86 E0 (5C 00 45 6B 00 00 67 03) 82 + SIG. -> en clair

70 56 (86 74 79 F9 D6 DF 67 03) 82 + SIG. -> après le premier MASK&CYPHER (*hypothèse*)

70 56 (05 B6 20 15 DF 76 FA B4) 82 + SIG. -> après l'élaboration du quadruple mot pré-signature

De l'élaboration de 86 74 79 F9 D6 DF 67 03 on arrive au quadruple mot 05 B6 20 15 DF 76 FA B4

Voilà où nous sert la présence de l'octet 86 : il est possible de construire une réponse dont le premier quadruple mot en clair soit égal à 86 74 79 F9 D6 DF 67 03 (le premier octet correspondrait à la pseudo-nano de la réponse à la C1 36):

C1 38 21 90 LL 71 D1 D6 56 6A F2 72
5B DA F3 82 A4 A2 23 7F 48
A0 74 79 F9 D6 DF 67 03.... etc

Les octets mis en évidences suivent la signature et précèdent les 90 octets SSE/enveloppe.

C1 38 21 90 LL

Réponse longue avec d1=03, deux PPV record

Nous sommes presque arrivé à la fin de la vérification, en faisant varier les octets des enregistrements dumpés on arrive à une réponse de ce type :

05 B6 20 15 DF 76 FA B4
0F FE 2E 97 92 0B 9F B6
D7 4D 0F DF 47 1A C8 19
EA 56 EF 56 1E 5F C5 A1
A2 DE 82 + SIGNATURE

- Le premier quadruple mot est le MASK&CYPHER de 86 74 79 F9 D6 DF 67 03
- Tandis que dans la version courte, elle était le résultat de l'élaboration du (LB), constitué des mêmes octets du point précédent.

Résultat : l'élaboration du (LB) est un MASK&CYPHER

Nous pouvons aussi obtenir la « somme mot » associé au masquage :

$$(0F\ FE + 2E\ 97 + 92\ 0B + 9F\ B6) \bmod 4000 = 7056 \bmod 4000$$

Considérons l'instruction courte

70 56 (05 B6 20 15 DF 76 FA B4) 82 ...

Les deux premiers octets sont justement « 70 56 » et ce sont les seuls, qui dans ce cas prennent part à la « somme mot »

Preuve numéro 2 – Possible implémentation du DECRYPT-SE

En générant la « succession des delta » pour les instructions suivantes, nous obtenons des résultats identiques (à part le désalignement, voir page 43 pour la signification des tables des delta) :

C1 38 21 B0 LL

89 69 4B 88 AD B1 EA FB 98 D8 82 55 6C E6 D4
AC B2 3C 50 x1 x2 + byte in SSE/envelope

C1 38 21 9C LL

89 69 D7 37 F9 C8 14 17 3B 56 82 6A 8F 4E 36
F3 4D AB 92 y1 y2 + byte in SSE/envelope

En analysant les tables on réussis à identifier le degré de désalignement (voir page 44) et on trouve, par exemple, la correspondance suivante : $(y1\ y2 - x1\ x2) \bmod 4000 = 0F9E$

La valeur "0F 9E" n'est pas casuelle, mais coïncide avec la différence entre les « somme mots » des octets suivants :

D8 82 55 6C E6 D4 AC B2 3C 50 première INS

56 82 6A 8F 4E 36 F3 4D AB 92 seconde INS

"D8" e "56" sont des octets en SUPERENCRYPTION : la « somme mots » considère les octets avant leur élaboration, alors :

Indépendamment de l'ordre dans lequel est exécuté le (dé)CYPHER et le (UN)MASK d'un quadruple mot, la « somme mots » est exécutée avant les deux processus.

SUPERENCRIPTION – observation sur les temps d'exécution et sur les bugs

Note :

Si les données à dé(crypter) sont exactement 8 , la phase 2 n'est pas exécutée et la somme SF vaut 0000

Voici les résultats avec les C1 38 « CUSTOM » (temps moyens sur des échantillons significatifs et de longueurs égales)

```
(0x1A octet en SE -> 3 quadruple mots + 2 octets) : 534878 cc
(0x19 octet en SE -> 3 quadruple mots + 1 octet) : 532580 cc
(0x18 octet en SE -> 3 quadruple mots + 0 octet) : 487071 cc
(0x17 octet en SE -> 2 quadruple mots + 7 octet) : 489213 cc
(0x11 octet en SE -> 2 quadruple mots + 1 octet) : 474492 cc
(0x10 octet en SE -> 2 quadruple mots + 0 octet) : 424974 cc
(0x0F octet en SE -> 1 quadruple mot + 7 octets) : 429111 cc
(0x0E octet en SE -> 1 quadruple mot + 6 octets) : 431064 cc
(0x0D octet en SE -> 1 quadruple mot + 5 octets) : 428460 cc
(0x08 octet en SE -> 1 quadruple mot + 0 octet) : 340008 cc
(0x07 octet en SE -> exécution BUGGEE) : 341730 cc
(0x06 octet en SE -> exécution BUGGEE) : 341668 cc
(0x03 octet en SE -> exécution BUGGEE) : 343557 cc
```

La variation du nombre d'octet en SE et possible à travers l'envoi d'instruction de longueur équivalentes, mais avec un paramètre P3 différent.

Beaucoup de facteurs entre en jeu dans l'exécution de chaque commande.

La tendance est d'augmenter le temps nécessaire à la réponse, en fonction de la longueur de la commande.

Essayons de scinder les temps en plusieurs parties :

- A - Une partie constante
- B - Une partie qui augmente avec la croissance du nombre d'octets utilisés
- C - Une partie qui augmente avec la croissance du nombre de quadruple mots utilisés
- D - Une partie RANDOM (dans laquelle on peut inclure, le pré-parsing, si on a effectué des tests égaux avec un nombre suffisant d'instructions différentes)

Note : les essais sont fait à avec des clés et des tables HASH identiques, donc les temps d'accès au données sont maintenues constantes (clés différentes → position différentes en EEPROM → temps différents pour les retrouver)

Dans la partie « C » sont inclus le DECYPHER&UNMASK, on peut facilement séparer le DECYPHER du reste (y compris de l'UNMASKING), obtenant un temps d'environ 39000 cycle d'horloge pour chaque quadruple mot chiffré.

Des cas pris en considération on peut extrapoler les résultats suivants :

Octets en SE	Nombre de (de) CYPHER
0x19 0x1A ...	5
0x11 ... 0x18	4
0x09 ... 0x10	3
0x01 ... 0x08	1

En effet dans le modèle SE, étaient présents :

- Un CYPHER en phase d'initialisation
- « N » CYPHER dans la seconde phase, avec « N » égal au nombre de quadruple mot entier.
- Un CYPHER dans le processus de l'élaboration du quadruple mot pré-signature

L'évaluation du comportement en présence de seulement 8 octets à crypter/décrypter, est importante ; le processus prend un chemin légèrement différent en exécutant UN seul CYPHER/DECYPHER : la seule phase exécutée est la troisième.

Dans ce cas particulier la somme :

$$SF = \text{sum}(\text{CYPHERTEXT}-8)$$

Ne sera appliqué à aucun octets, assumant la valeur 0000. On justifierait aussi le comportement en présence d'un nombre d'octets à décrypter inférieure à 8, il est probable que la routine a été implémentée, en supposant implicitement, que les données à mettre en clair, devaient être au moins de 8, ne prévoyant pas que le pointeur puisse assumer des valeurs négatives (un peu comme le ZZBUG). Dans ces conditions la « somme mots » prendrait en considération un certain nombre d'autres octets, qui seraient autre les données à décrypter.

Ce n'est pas par hasard que dans l'**OCTET-BUG**, on a put observer, de quel façon la « somme mots » dépend de la signature, des octets en clair et des octets sous enveloppe. Le **RESET-BUG** peut se justifier par le fait que la « somme mots » peut continuer dans la partie RAM qui suit le COMMAND BUFFER, allant ainsi prélever des données dans une zone au contenu non prédictible (pensez à la ROM de la 6.0 : pendant la phase de démarrage (STARTUP) il y a des page et RAM remplies avec des valeurs fortuites, pour vérifier le fonctionnement du générateur aléatoire (RANDOM GENERATOR).

Même l' **instabilité du statut, pour les instructions longues** est explicable : la « somme mots » continue jusqu'à atteindre pour son index une valeur précise (zéro ?) : la RAM lue à une quantité fixe, mais la position par laquelle la lecture commence, est dictée par la position du premier octet à décrypter.

Dans les instructions longues, le premier octet est dans une position plus avancée et donc la somme ira prélever le contenu d'adresses mémoire positionné plus en avant dans la RAM.

En opérant ainsi, si on impliquait une région affectée à des registres et/ou des mémoire tampons, on aurait une possible modification de comportement à chaque envois d'instructions.

On peut facilement obtenir de combien d'octet dépasse la « somme mots » en fonction de la longueur (LEN) de l'instruction et de la valeur de P3. Il reste à expliquer le phénomène que nous obtenons en présence de zéro octet à décrypter, c'est à dire le blocage de la carte ou encore l'envoi de l'ATR.

Il est possible que nous dépassions la RAM, nous retrouvant dans une aire protégée ou non formatée, provoquant le déclenchement d'un dispositif de protection de la carte (MEMORY MANAGEMENT UNIT) qui était présent dans les version 3.1 et 4.0, ou encore on provoque la modification de partie de RAM destinées à un autre usage (STACK ?) ou encore on altère l'exécution du logiciel de gestion qui prend un chemin inattendu, provoquant un phénomène analogue au ZZBUG.

Il existe une multitude de chemin à parcourir pour mieux comprendre, surtout pour prévoir quel sera le résultat du décryptage affecté par le bug.

Observez par exemple ceci :

C1 3C 21 9E 76

8F F1 36 3A D0 A9 41 FE B3 27 1A 81 31 16 80 02 24

00 80 82 + signature + 90 octets sous enveloppe

Puisque nous avons affaire à une C1 3C, les premiers 8 octets (en vert) resteront en clair il ne reste donc que les octets en bleu clair, qui étant moins de 8 seront donc traités de façon buggée

decypher & unmask (00 80 xx xx xx xx xx xx) = X7 X6 X5 X4 X3 X2 X1 X0

Le résultat est qu' il va réécrire (autre conséquence du bug) les 8 octets pré-signature, en modifiant aussi les octets qui précèdent ceux à mettre en clair :

C1 3C 21 9E 76

8F F1 36 3A D0 A9 41 FE B3 27 1A X7 X6 X5 X4 X3

X2 X1 X0 82 + signature + 90 octets sous enveloppe

Résultat : le premier quadruple mot, qui dans la C1 3C devrait rester intact, est maintenant partiellement modifié, avec des conséquences sur le PARSING des NANO.

Dans le cas d'une instruction 38/40, le résultat irait là, influencer les données de P3, dont la sur écriture ne provoque aucun effet particulier.

C1 40 21 B0 76

65 31 D3 B2 97 A5 D2 86 81 DE 82 + signature + 90 octets sous enveloppe

C1 40 21 B0 76

65 31 X7 X6 X5 X4 X3 X2 X1 X0 82 + signature + 90 octets sous enveloppe

Ce phénomène mériterait une étude ultérieure

SUPERENCRYPTION – la phase du MASKING

La première tentative d'interprétation du processus de masquage a utiliser la chaîne d'instruction **3C/3A/04/02** :

- On construit une C1 3C/NANO D1 avec des CW connues
- On lit les donnée en clair DW (par une C1 3A)
- On prend les DW séparément et on les cryptés avec l'instruction 04
- On lit le résultat avec l'instruction 02

Le but est d'arriver à démontrer que, dans l'instruction 3C, le décryptage des CW, se fait à travers la suite UNMASKING suivit du DECYPHER. De plus on vérifie la différence du DECRYPT-SE

Première question : INS 3C/NANO D1 utilise t'il l'UNMASKING ?

Seconde question : Si l'instruction 3C fait aussi l'UNMASK, traite t'il les quadruple mots séparément ou traite t'il les CW comme si c'étaient un test SE long 16 octets ? (si vous pensez que c'est la même chose, il suffit de penser comment agirait l'UNMASK et le DECYPHER dans les deux cas)

Troisième question : l'instruction 04 utilise t'elle le MASKING ?

On démarre avec quelques données expérimentales :

```
CW "1": FF CA 00 5A 24 F3 67 E3 00 00 D2 00 24 A6 FF FF
C1 3A : B2 35 D4 6D C6 86 40 93 E6 25 2B E4 0A 38 78 37
C1 04 : DD F8 3F 4E 04 C6 04 D5 37 CA 89 B8 E0 C6 97 A6
```

```
CW "2": FF CA 00 B8 0D 1B 93 E3 00 00 D2 00 24 A6 FF FF
C1 3A : B6 88 FF D3 AA CA 06 EF E6 25 2B E4 0A 38 78 37
C1 04 : DD F8 3F 67 E6 C6 EC 21 37 CA 89 B8 E0 C6 97 A6
```

```
CW "3": FF CA 00 B8 0D 1B 93 E3 00 00 D2 00 24 A6 FF FF (utilise les mêmes
C1 3A : C0 3C 43 BB CD 56 D7 E3 52 6D 95 05 70 4C 09 AF données que la 2 mais
C1 04 : DD F8 3F 67 E6 C6 EC 21 37 CA 89 B8 E0 C6 97 A6 des clés différentes)
```

```
CW "4": FF CA 00 5A 24 F3 67 E3 00 00 D2 00 25 00 4E 08
C1 3A : B2 35 D4 6D C6 86 40 93 4A 9A CD BE 8A EC 75 7B
C1 04 : DD F8 3F 4E 04 C6 04 D5 86 CA 2F B8 17 C7 97 A6
```

En examinant les CW (INS 3C) et les DW (INS 3A) on observe qu'en changeant les valeurs d'un quadruple mot crypté, les valeurs du DECRYPT de l'autre quadruple mot ne changent pas.

Ceci exclut que les 16 octets soient traités, comme si c'était une seule entité de 16 octets. Nous venons de répondre la [seconde question](#) et du coup renforcé la question légitime sur les tests avec les INS 02/04 appliqués aux DW séparément.

De plus on observe qu'il ne semble pas avoir de corrélation entre le CW et le DW homologues, c'est justement le résultat attendu et qui nous prouve le DE - CYPHER.

Nous ne pouvons encore tirer de conclusion sur la présence ou pas de l'UNMASK dans le processus de la 3C.

Nous avons donc pour la [première question](#), trois réponses possibles.

Chaque quadruple mot :

- X1 – pourrait subir seulement le DE - CYPHER
- X2 – Pourrait subir le UNMASK puis le DE - CYPHER
- X3 - Pourrait subir le DE – CYPHER puis le UNMASK

En ce qui concerne la [troisième question](#), on peut par symétrie faire le même discours en examinant les DW (INS 3A) et leurs CRYPT avec les INS 04/02.

Les réponses possible seront :

- Z1 – pourrait subir seulement le CYPHER
- Z2 – Pourrait subir le MASK puis le CYPHER
- Z3 - Pourrait subir le CYPHER puis le MASK

En examinant une CW (INS 3C) et son cryptage avec INS 04/02 on observe clairement une relation entre les deux quadruple mots, bien qu'il soient différents. Ceci signifie qu'entre l'une et l'autre il y a entre, une élaboration.

[La question est : qui fait usage de cette élaboration ? la 3C ou la 04 ?](#)

On peut imaginer deux possibilités :

- a- INS 3C fait UNMASK & INS 04 ne fait pas le MASK
- b- INS 04 fait le MASK & INS 3C ne fait pas le UNMASK

En se souvenant des hypothèses précédentes (X1...X3 et Z1...Z3) liées à cette dernière considération, on conclut que toutes les configuration "**Xi**" - "**Yj**" ne sont pas admissibles :

- si X1 est valide alors on exclut X2, X3, Z1
- si Z1 est valide alors on exclut Z2, Z3, X1.82

Examinons maintenant les CW/DW « 1 » et « 3 » : les résultats des INS 04/02 sont identiques en partant de DW différents avec des clés différentes.

Raisonnons par l'absurde : si le couple "X3" - "Z1" étaient vrais, on aurait le processus suivant : chaîne 3c/3a/04/02 .

CW --> de-cypher --> unmask --> DW --> cypher --> réponse INS 02/04

Avec l'UNMASKING à cheval entre on ne pourrait pas arriver au même résultat final.

En conséquence :

- si Z1 est valide alors on exclut Z2, Z3, X1, X3

Le raisonnement est identique si on prend en considération le couple "X1" - "Z2" :

CW --> de-cypher --> DW --> mask --> cypher --> réponse INS 02/04

Avec le MASKING à cheval entre le DE – CYPHER et le CYPHER on ne pourrait pas arriver au même résultat final.

En conséquence :

si X1 est valide alors on exclut X2, X3, Z1, Z2

Il nous reste deux options possibles :

- "X1" - "Z3" : INS 3C seulement DE – CYPHER / INS 04 CYPHER & MASKING
- "X2" - "Z1" : INS 3C UNMASKING & DE- CYPHER / INS 04 seulement CYPHER

Pour déterminer quelle est la bonne, il est nécessaire d'examiner, plus en détail, les résultats relatifs à la chaîne 3C/3A/04/02.

On considère une instruction 3C sous sa forme générale :

C1 3C P1 P2 ... 04 D1 + CW1 + CW2 ...

L'instruction suivante (3A) rend les DW qui sont re cryptés séparément à travers les instructions 04/02.

Essayons de séparer les résultats relatifs à la première CW de ceux relatifs à la deuxième CW :

CW1	Réponse Instruction 02
A2 B3 00 00 D2 00 9F EA	A4 A5 3F B8 5E CF F7 2D
A2 B3 00 00 D2 00 7C E5	A4 A5 3F B8 5E C0 F7 CE
A2 B3 00 00 D2 00 16 20	A4 A5 3F B8 5E 05 F7 A4
A2 B3 00 00 D2 00 B7 7D	A4 A5 3F B8 5E 58 F7 05
47 B3 00 00 D2 00 7E D5	A4 40 3F B8 5E F0 F7 CC
47 B3 00 00 D2 00 F5 10	A4 40 3F B8 5E 35 F7 47
47 B3 00 00 D2 00 BB 74	A4 40 3F B8 5E 51 F7 09
47 B3 00 00 D2 00 53 F4	A4 40 3F B8 5E D1 F7 E1

CW2	Réponse Instruction 02
69 FF 12 00 00 FF FF 0F	37 CA D0 78 10 E2 68 CF
69 FF 12 00 00 FF FF 22	37 CA D0 78 3D E2 68 CF
69 FF 12 00 00 FF FF 59	37 CA D0 78 46 E2 68 CF
69 FF 12 00 00 FF FF 6F	37 CA D0 78 70 E2 68 CF
69 FF 12 00 00 FF FF 92	37 CA D0 78 ha E2 68 CF
69 FF 12 00 00 FF FF A1	37 CA D0 78 BE E2 68 CF
69 FF 12 00 00 FF FF C5	37 CA D0 78 DA E2 68 CF
69 FF 12 00 00 FF FF F4	37 CA D0 78 EB E2 68 CF

On découvre immédiatement la présence d'une permutation d'octets et d'une ultérieure élaboration. Cette dernière n'est en fait qu'un simple XOR, en effet en appliquant le XOR entre un octet du CW1 et l'homologue présent dans la réponse à l'instruction 04/02, on obtient toujours le même résultat.

On réussit à déterminer un quadruple mot à valeurs connues à utiliser dans le XOR.

On note un résultat différent entre le CW1 et le CW2 (permutations différentes et quadruple mot connu différent), mais le processus est exactement le même. En faisant de test avec diverses tables HASH ou encore sur des fournisseurs d'accès différents, on obtient des résultats identiques, il n'y a donc d'influence de tel paramètres sur le processus.

Unmasking di CW1	Unmasking di CW2
CW1 masquée:	CW2 masquée :
M0 M1 M2 M3 M4 M5 M6 M7	M0 M1 M2 M3 M4 M5 M6 M7
M0 M1 M2 M3 M4 M5 M6 M7 XOR 07 17 3F 5E 6A F7 B2 25 =	M0 M1 M2 M3 M4 M5 M6 M7 XOR A6 97 6A CA E2 2F C8 1F =
N0 N1 N2 N3 N4 N5 N6 N7	N0 N1 N2 N3 N4 N5 N6 N7
CW1 non masquée:	CW2 non masquée:
N1 N0 N2 N4 N3 N6 N7 N5	N7 N6 N3 N1 N5 N2 N0 N4
(Noter la permutation!)	(Noter la permutation!)

Le processus de **MASKING CW** prévoyait d'abord la permutation, puis l'opération de XOR avec le quadruple mot connu.

La possibilité d'intervertir le processus de masquage et de XOR est évident, à condition de permuter aussi les octets du quadruple mot connu. Bien que cela ne fasse pas de différence, il pourrait être utile pour mieux comprendre le fonctionnement du MASKING.

Pour l'instant nous pouvons tirer deux conclusion :

Vu la différence de traitement entre le quadruple mot 1 t le quadruple mot 2 de la C1 3C, nous pouvons affirmer, sans aucun doute, que l'UNMASKING est inclus dans le processus de mise en clair des CW, ce processus change selon le quadruple mot considéré.

Le couple 04/02 pourrait ne pas exécuter le MASKING ou bien l'exécuter avec permutation et XOR divers.

Qui utilise le MASKING ? (Hypothèse à vérifier)

- Le CRYPT d'un quadruple mot à travers INS 04, ne devrait pas utiliser le MASKING
- Le DECRYPT des clés (INS 40 – NANO 90/91) devrait utiliser l'UNMASKING.

Malheureusement, ce processus de masquage des quadruples mots ne semble pas identique à celui subit en SUPERENCRIPTION, en effet la réponse suivante à une C1 36 avec une clé 0F :

Réponse cryptée	Réponse en Clair
23 1E E8 D5 5E E5 A2 3F	86 E0 5C 00 00 00 00 00
E1 C5 7A DE CA 5C 3B 77	7B D2 00 1F 5D FF FF FF
9F CC F0 10 45 0E A3 0F	FF FF FF FF FF 89 00 81
48 84 88 82 41 28 39 1C	00 00 03 82 + SIGNATURE
1B 66 D1 06	

On découvre qu'il n'est pas possible de reconstruire les tables de permutation/XOR en fonction de la « somme mot » en suivant la procédure suivante :

1. On annote le premier quadruple mot crypté
2. On fait varier le premier quadruple mot en gardant fixe les octets restants (c'est à dire fixe la « somme mot » du second quadruple mot)
3. On annote la « somme mot modulo 4000 » du second quadruple mot
4. On annote le premier quadruple mot en clair
5. On crypté avec l'INS 04/02 (voici la raison de la clé 0F), le premier quadruple mot en clair
6. On analyse les résultats a la recherche des permutation et des XOR
7. On itère le processus en variant le second quadruple mot (conséquence variation de la « somme mot »)
8. On analyse les résultats a la recherches des relations entre les permutation, XOR et « somme mot modulo 4000

La procédure aurait fonctionné si dans la SUPERENCRYPTION nous aurions eu d'abord le CYPHER, puis le MASKING, mais nous n'arrivons pas à mettre en évidence aucun test expérimental analogue à celui des CW/DW donc :

Sur chaque quadruple mot en SUPERENCRYPTION agit d'abord le MASKING puis le CYPHER, de manière inverse à ce qui se passe dans le processus de décodification des CW.

La méthode pour la construction de la table doit donc être modifiée de la façon suivante :

1. On annote le premier quadruple mot crypté
2. On fait varier le premier quadruple mot en clair en maintenant fixe les octets restants (c'est à dire fixe la « somme mot » du second quadruple mot)
3. On annote la « somme mot modulo 4000 » du second quadruple mot
4. On annote le premier quadruple mot en clair
5. On itère le processus variant le second quadruple mot (conséquence variation de la « somme mot »)
6. On construit une INC 3C avec 04 D1 A1 A2 A3 A4 A5 A6 A7 A8, etc.. ou A1...A8 est la donnée du MASKING (selon l'INS 3C-NANO D1) appliquée au premier quadruple mot de la réponse cryptée de la C1 36
7. On envoie C1 3C et C1 3A
8. On annote le premier DW (qui correspond au résultat du DECYPHER du premier quadruple mot de la C1 36 sans que soit appliqué l'INMASKING)
9. On compare le DW avec le quadruple mot en clair.
10. On analyse les résultats, à la recherche de la relation entre permutation, XOR et « somme mot modulo 0x4000 »

Il faudra tenir compte du fait que chaque quadruple mot devrait être permuté, toujours d'une certaine façon, déterminé de sa position dans l'instruction.

Hypothèse sur la « **somme mot** » (citation)

3F FF qu'il type d'information peut t'il nous donner ?.

FF est un beau pointeur pour une table de 256 octets, avec laquelle on pourrait résoudre le problème de la pseudo-clé.

Reste le problème de la permutation, mais **3F** correspond, en comptant le 0, à un superbe 64 décimal qui est aussi le carré de 8.

Cela pourrait être les permutations, mais certainement pas toutes les permutations possibles, par coïncidence c'est aussi un nombre qui, avec une autre table de 256 octets, à travers la configuration des bits, permet de donner les informations correctes pour la permutation.

Dernière considération sur le MASKING : on peut exclure que tel processus utilise le même algorithme que l'enveloppe, la première raison vient de l'analyse des données (les propriétés statiques particulières des NLB et LB dans les ECM/EMM).

La seconde, plus importante nous vient de la POWER ANALYSIS, vu que l'enveloppe utilise un présumé CRYPTO-ENGINE (ou crypto-processeur), tandis que les opérations de (DE) CRYPTÉ et (UN)MASK ne l'utilisent pas.

Le calcul de la signature en SECA 2

Prenons comme point de départ le calcul de la signature en SECA 1 :

Signature en SECA 1 (citation)

Cette procédure utilise l'algorithme de cryptage SECA.

Les données utilisées, pour le calcul, sont les octets du corps de l'instruction jusqu'à la NANO82. On divise les données en blocs de 8 octets, le derniers blocs pouvant, bien sur, avoir moins que 8 octets. Dans ce cas on ajoute autant de « 00 » que nécessaire pour arriver à former un bloc de 8 octets. La procédure complète est la suivante :

- 1) On initialise l'HASHBUFFER (qui contient 8 octets)
- 2) XOR entre le premier bloc de 8 octets et le contenu du BUFFER
- 3) Exécution de l'algorithme de cryptage SECA (on met le résultat dans le BUFFER)
- 4) On recommence avec le bloc suivant et on retourne au point 2

A la fin de la procédure nous avons dans le BUFFER, la signature calculée.

L'initialisation de l'HASHBUFFER varie selon la valeur prise par les 3 bits les plus significatif de P1 :

a) x01xxxxx HASH-BUFFER initialisé avec : **UA UA UA UA UA UA 00 00**
(Unique Address – le numéro de série de la carta)

b) x10xxxxx HASH-BUFFER initialisé avec : **PP PP PP PP 00 00 00 00**
(La PPUA du fournisseur d'accès à qui est envoyée la commande)

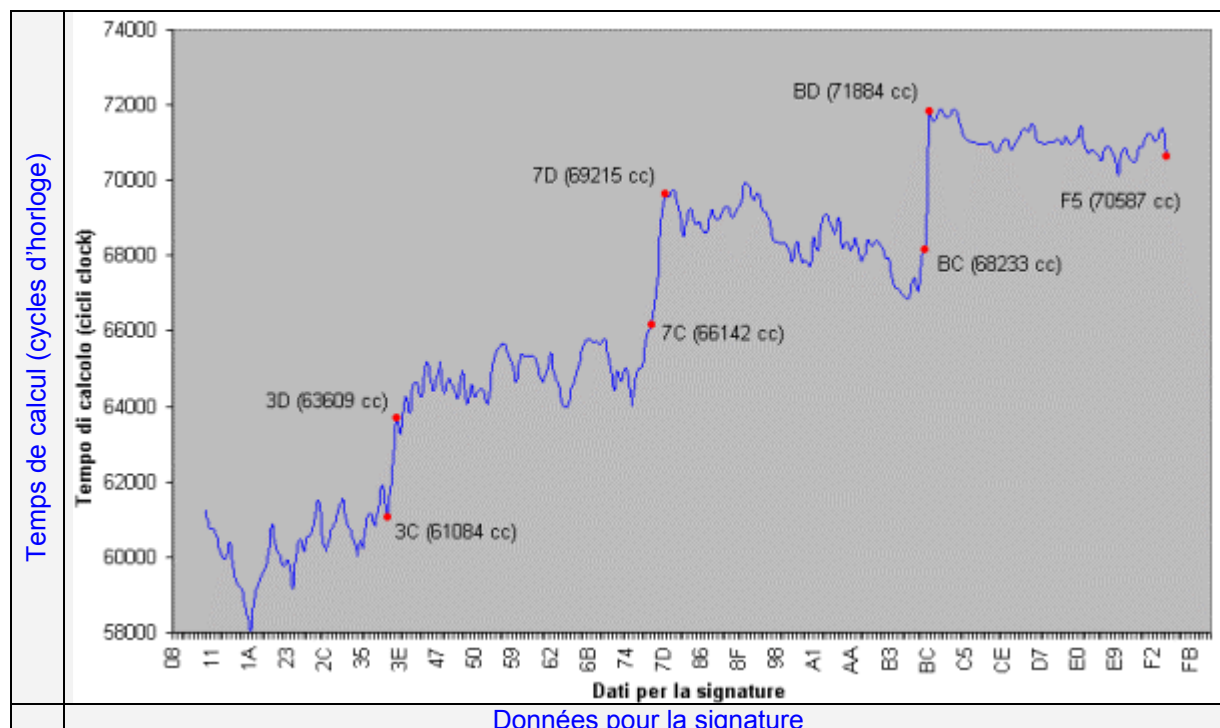
c) x11xxxxx HASH-BUFFER initialisé avec : **SA SA SA 00 00 00 00 00**
(La Shared Address – Premier 3 octets de la PPUA)

d) x00xxxxx HASH-BUFFER initialisé avec : **00 00 00 00 00 00 00 00.**

En SECA 1, donc, on exécutait un certain nombre de CRYPT pour arriver à la signature, le nombre dépendait de la longueur de la commande (en particulier sur le nombre de quadruple mots, résultant de la division du corps de l'instruction).

Pour essayé de comprendre le calcul de la signature en SEKA2, on peut tenter une analyse des temps de réponses et obtenir des résultats intéressant.

Les données des tests font référence au temps utilisé par une instruction C1 38 pour la vérification de la signature.



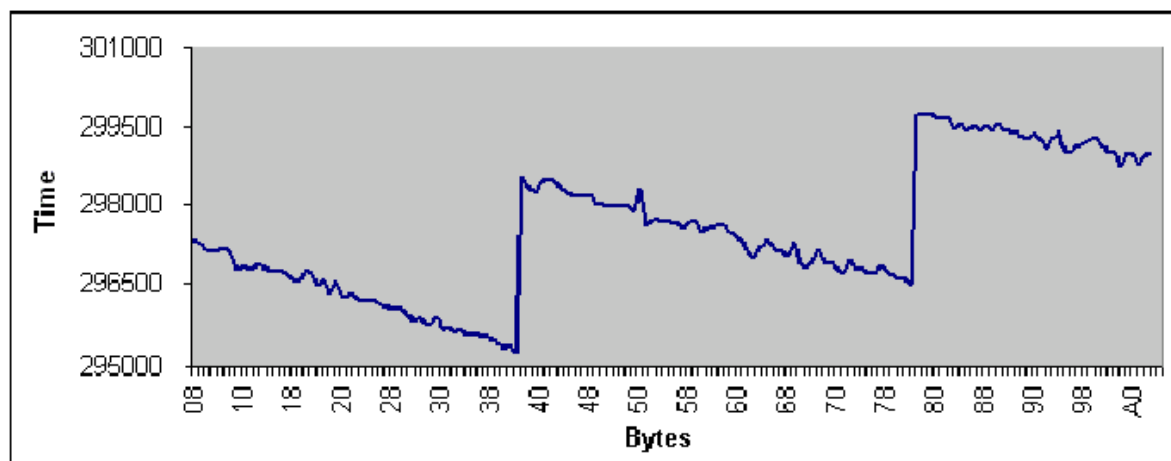
Contrairement à ce qu'on croyais, le calcul de la signature, n'est pas indépendant de la longueur de la commande : on observe une sensible augmentation du temps nécessaire à la vérification de la signature dans les passage suivants :

3C -> 3D octet 7C -> 7D octet BC -> BD octet

Il est presque naturel de supposer que l'algorithme de calcul travaille par bloc de 64 octets, les résultats de l'élaboration de chaque bloc sont concaténés les uns aux autres et le résultat est un quadruple mot. Il existe beaucoup d'algorithmes qui utilisent 64 octets de données, mais rendent un nombre d'octets supérieur à 8. Il se pourrait donc qu'il y ait un processus de coupure du résultat (comme dans le calcul de la pré-signature de IRDET01).

On pourrait se poser la question sur le premier saut de 3C, c'est à dire quand la longueur des données vaut 64 (40 en hexadécimal). Il est possible qu'on ajoute quelques octets en phase de préparation du calcul (cela arrive avec certains algorithmes connus). Les quatre octets manquants pourraient être l'équivalent SECA2 de l'initialisation de l'HASHBUFFER du SECA1.

Du graphique nous ne pouvons avoir d'autres observations, si ce n'est par un autre graphique qui montre une rampe à gradins descendante. Elle ne dépend pas vraiment de la signature, mais apparaît dans les premières phases de l'exécution de l'instruction.



Signature SECA 2 – le TIMMING de 9001-A

En considérant le processus d'exécution de l'instruction, le test sur la présence de la NANO 82 arrive avant la plupart des contrôles, mais le temps de restitution de ce statut, est plus long que la plupart des temps de réponses des contrôles ultérieurs. Cela signifie que d'autres opérations sont exécutées avant la restitution de ce statut.

On observe qu'entre les statuts 9002_A et 9002_B, il y a une différence égale au temps nécessaire au prélèvement de la clé en EEPROM et de sa mise en clair.

Ceci nous laisse penser que de toute façon le calcul de la signature a lieu même en absence de la NANO 82. L'analyse des temps de réponse accrédite cette hypothèse :



Le temps de réponse a une progression en escalier comme le graphique relatif au calcul de la signature.

Retenez que le statut 9034, produit par faillite du contrôle immédiatement précédent la vérification de la signature (9002_B), ne présente pas la même progression.

FANTASECA (citation)

S'il en était ainsi, la carte ne trouvant pas la NANO 82, sauterait directement à la vérification de la signature, la clé n'étant pas chargée, il utiliserait la KEY-BUFFER comme elle est (qui pourrait contenir le résultat de la dernière KEY OPERATION), et le contrôle rate, mais si dans la KEY-BUFFER, il tombe sur une clé qui valide la signature, alors il passe directement au processus du DECRYPT SE etc..

Et vu que nous parlons de FANTASECA, alors on peut admettre de forcer une clé connue dans son BUFFER.....

Signature SECA 1 – Routine alternative (tirée d'un post de mars 2001)

Pour être complet, on rapporte l'algorithme pour la signature en SECA1, en utilisant quelques instructions particulières (différentes de l'algorithme standard utilisé par les ECM/EMM).

La syntaxe est la suivante :

C1 INS P1 P2 LEN + DATA PACKET

P1

Bit 0...3 : Index du fournisseur d'accès à qui on envoie la commande
Bit 4 : Utilisation de la clé primaire ou de la clé primaire + secondaire
Bit 5,6,7 : VOIR APRES!!!

P2

Bit 0...3 : Index de la clé à utiliser pour (de)CRYPT SUPERENCRYPTION
Bit 4,5,6 : VOIR AORES!!!
Bit 7 : Indicateur de la SUPERENCRYPTION

LEN

C'est la longueur du paquet de données

A quoi servent les bits 7,6,5 de P1 et les bits 6,5,4 de P2 ?

Ils servent à l'initialisation des clés utilisées dans l'algorithme de CRYPT/DECRYPT et le calcul de la signature.

Supposons que nous avons une instruction adressée au fournisseur d'accès 01, avec un index de clé égal à 06

C1 INS 41 36 LEN 10 01 02 03 04 05 06 82 + SIGNATURE

P1 = 0x41 -> 01000001 -> bit 7,6,5 = 010 -> X = 010 = 0x02

P2 = 0x36 -> 00110110 -> bit 6,5,4 = 011 -> Y = 011 = 0x03

La clé utilisée pour le calcul de la signature n'est pas la clé 06, indiquée par le quartet faible de P2.

Cette clé est utilisée pour crypter un quadruple mot, qui est généré de cette façon :

- On part d'un quadruple mot nul : **00 00 00 00 00 00 00 00**

- à partir du (X+1)-ième octet de donnée pour une longueur Y, on réécrit ce quadruple mot .

Exemple :

```
X=00; Y =00; 00 00 00 00 00 00 00 00 00 00 -> 00 00 00 00 00 00 00 00
X=02; Y =03; 00 00 00 00 00 00 00 00 00 00 -> 02 03 04 00 00 00 00 00
X=01; Y =04; 00 00 00 00 00 00 00 00 00 00 -> 01 02 03 04 00 00 00 00
```

L cas que examinons est le second donc :

```
- Quadruple mot généré :      02 03 04 00 00 00 00 00
- Clé primaire 06 :          49 63 6F 6E 43 6F 69 6C
- Quadruple mot crypté :     03 B1 FF 19 98 6D 2F 82
```

Cette dernière représente la clé avec laquelle on calculera la signature, mais c'est pas fini, il y a deux autres chose à mettre en place :

1) Dans un calcul normal, de la signature on doit initialiser l'HASHBUFFER, selon les différentes cas : UA/SA etc.. Ici aussi il est nécessaire de passer par ce chemin, cet quadruple mot sera tout à zéro au départ, on réécrira ensuite à partir du premier octet de données pour une longueur X.

Exemple :

```
X=00 ; 00 00 00 00 00 00 00 00 00 00 -> 00 00 00 00 00 00 00 00
X=02 ; 00 00 00 00 00 00 00 00 00 00 -> 10 01 00 00 00 00 00 00
X=01 ; 00 00 00 00 00 00 00 00 00 00 -> 10 00 00 00 00 00 00 00
```

2) Dans le calcul normal de la signature il y a un index qui nous donne le point de départ habituellement cet index vaut zéro, ceci signifie que la signature est calculée à partir du premier octet des données. Dans ce cas particulier le pointeur assumera la valeur de :

$$\text{Offset} = X + Y$$

Des exemples précédent, il en résulte que le quadruple mot de départ pour le calcul de la signature est :

```
Offset = 00 --> 10 01 02 03 04 05 06 82 (N'oubliez pas que
Offset = 05 --> 05 06 82 00 00 00 00 00 c'est une routine
Offset = 05 --> 05 06 82 00 00 00 00 00 du SECA 1)
```

En pratique on écarte les premiers X+Y octets dans le calcul

- HASH-BUFFER : 10 01 00 00 00 00 00 00
- 1° Quadruple mot : 05 06 82 00 00 00 00 00
- XOR entre les deux : 15 07 82 00 00 00 00 00

On crypte le résultat avec la clé construite précédemment
(c'est à dire 03 B1 FF 19 98 6D 2F 82)
obtenant la signature : 54 EC C2 B8 4C A7 8A A5 .

Signature SECA 2 – Les EMM POST-BLACK MONDAY

Il est de connaissance notoire que les instructions à travers le bug 36/38 sont adressables uniquement à la carte avec laquelle ont à réaliser la commande.

Ceci est dû à la valeur assumé de P1 (voir page 88 – signature SECA 1) qui impose l'initialisation du HASHBUFFER avec le numéro de série de la carte, qui est différent pour chaque carte.

Malheureusement les opérations utilisées pour l'initialisation sont différentes du SECA 1 :

Preuve sur la signature (citation)

Une C1 36 relative à une certaine carte, n'est pas bonne pour une autre, même si la clé utilisée est la même : la C1 36 crypte et signe la réponse en initialisant l'HASHBUFFER avec le numéro de série de la carte (les derniers 6 octets de la C1 0E), le corps ne peut donc être renvoyée avec une C1 38 seulement par la même carte qui l'a crée (...)

J'ai fais l'expérience, j'ai pris ma vieille 4.0 et j'ai crée une MK = 00 00 00 00 00 00 00 00 puis j'ai envoyé, après avoir calculé la signature l'instruction suivante :

```
C1 40 01 02 12 80 FF FF FF FF FF FF FF 82 B4 60 A0 EB 43 A4 92 B2 [90 00]
```

Supposons maintenant que la série de ma vieille 4.0 soit : 00 00 01 02 03 04.

J'ai réinventé :

```
C1 40 21 02 12 80 FF FE FD FC FB FF FF FF 82 B4 60 A0 EB 43 A4 92 B2 [90 00]
```

En initialisant le BUFFER avec le SERNUM (SERIAL NUMBER) et exécutant le XOR des premiers octets de la chaîne avec celui-ci, j'ai eu un résultat identique.

Je n'ai pas pus résister à la tentation : j'ai pris une bonne C1 38, j'ai exécuté un XOR des premiers 6 octets, selon la routine reprise au dessus et je l'ai envoyé à ma V7 -> C1 38 01 Etc... (...)

Réponse ... l'HASH n'est plu initialisé comme avant (...)

Il est certain que si on arrivait à trouver une C1 38 du type C1 38 01 xE 13 ... ou x = 9 ,B, F le bug se déclencherait sur toutes les cartes ...

Nous n'avons pas encore la trouvé une façon de nous démarquer de la dépendance que nous avons vers le numéro de série de la carte et construire une commande universelle pour déclencher le bug sur toute les V7.0

Nous sommes à la recherche, de procédures avec les quelles nous pourrions construire un EMM modifié et BUGABLE à partir d'une LEN = 63 (avec cette NANO qui empêche l'utilisation du bug).

Dans l'état actuel des recherches, il y a des méthodes pour y parvenir, mais elles ne garantissent pas systématiquement la réussite, elles ne sont donc pas reprises dans cette DOC.

Exécution partielle des instructions : la méthode SEND-WAIT-RESET

Pour cet exercice nous avons besoin :

SMARTTIMER 1.82x (BETA)

Dans une des dernière « BETA » du programme, vous trouverez une option inédite « **sw-Unloop** » dont je vous donne une explication :

Envoyer la première commande, puis attendre un laps d temps en milliseconde égal à celle indiquée dans le cadre à droite du bouton « MULTISEND », puis faire un « RESET » de la carte. Très utile pour faire exécuter une partie d l'instruction.

Cela signifie que la carte est réinitialisée avant l'exécution totale de l'instruction, en fait nous avons une exécution partielle. Ceci peut être utile pour exécuter un certain nombre de NANO d'un EMM/ECM.

Considérons par exemple un EMM d'activation et supposons qu'on veuille réécrire les clés sans modifier la PPUA, il suffit de réinitialiser la carte avant l'exécution de la dernière NANO (NANO 41 – écriture de la PPUA).

Pour déterminer l'instant précis de la réinitialisation, il convient de procéder par tentatives en débutant avec une petite valeur de temps et en incrémentant jusqu'à obtenir l'exécution des NANO désirés.

Il faut se rappeler que dans la carte sont présents des routines qui introduisent des retards « RANDOM », et il est possible qu ' en envoyant, plusieurs fois, la même commande, avec le même temps d'attente en milliseconde, on obtiennes des résultats différents (le plus typique étant le plus ou moins de NANO exécutées).

Donnons un autre exemple a travers un ECM, il est possible par exemple de vouloir effacer un enregistrement PREVIEW sans pour autant en écrire une autre, il suffit de construire une C1 3C – NANO 27 avec donnée haute (= supérieure des données de l'enregistrement PREVIEW présent, inférieure à la date de fin d'abonnement) et faire un RESET (réinitialisation) avant la création d'un enregistrement Ax (qui ne se fait qu'après l'effacement des anciens enregistrements).

La méthode SEND-WAIT-RESET permet de découvrir une autre caractéristique de la V7.0, la présence d'une routine de contrôle d'écriture en EEPROM, aussi présente sur les version 6.0 et absente des versions précédentes.

Supposons que nous voulions effacer une clé (instruction 40 – NANO 10), l'effacement prévoit l'annulation de tous les octets de l'enregistrement. En arrêtant l'exécution de la NANO avant l'effacement du RECORD DESCRIPTOR (le dernier octet), on aurait théoriquement une KEY-RECORD partiellement altérée (et presque sûrement inutile à cause de la KEY-CHECKSUM non valide). Ce processus fonctionne par exemple sur une V4.1 OKI, tandis qu'en faisant l'expérience sur une V6.0, nous n'avons aucun effet si une écriture 'est pas arrivée à son terme, à la remise en service de la carte (par exemple avec un RESET) c'est la configuration précédent l'écriture qui est remise.

Vu que sur le V7.0, l'essai d'effacement partielle d'une KEY-RECORD, rate, on peut faire l'hypothèse quelle aussi utilise une routine de vérification d'écriture en EEPROM.

Autre point de vue : POWER ANALYSIS

Analyse de la V7 orientée vers le HARDWARE – tiré d'un SAT-FORUM

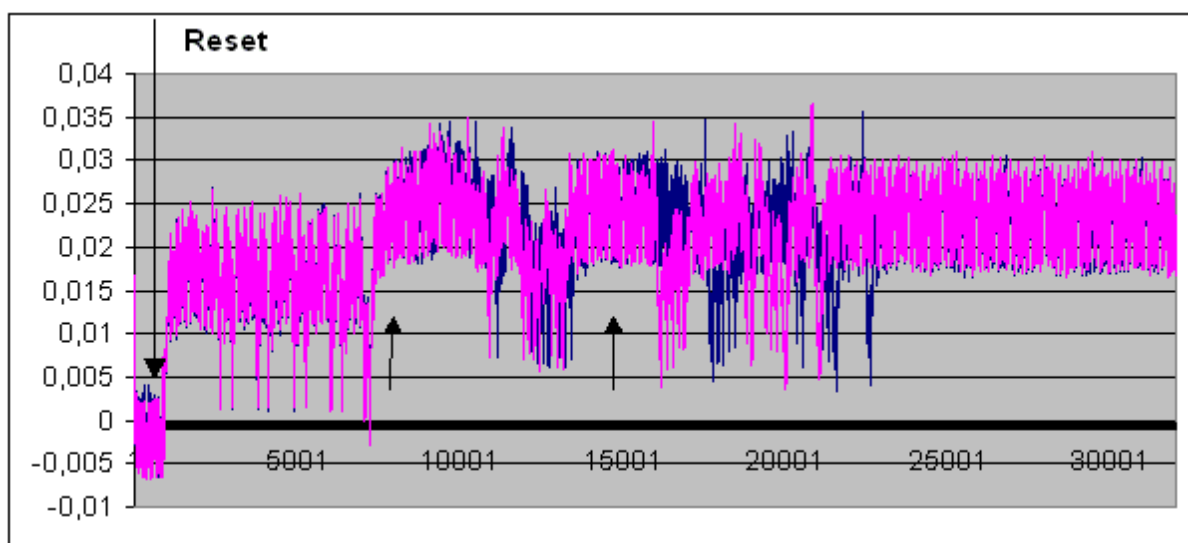
Premièrement on examine visuellement la superficie des contacts du chip, ceci nous dirigerait vers le producteur GEMPLUS (les avis vont aussi dans ce sens dans les divers forums public).

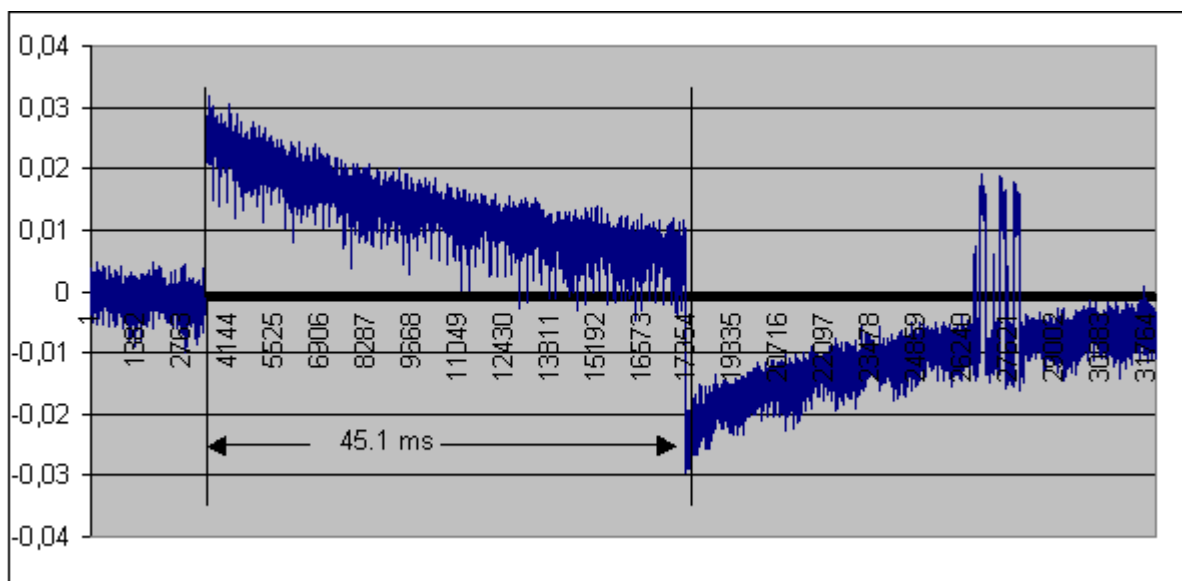
En recoupant la date de production/fabrication, aux caractéristique électriques et aux partenaires habituels de GEMPLUS, on pourrait faire l'hypothèse que la carte utilisée soit une PHILIPS **P8WE5xx** (avec crypto-processeur).

Un autre candidate pourrait être une variante de la série **AT90SCxxx** (ATMEL), elle aussi reprise dans les forums français. Un autre possible, la SIEMENS/INFINEON ont des modèles avec crypto par exemple la **66CXnnn**, il est vrai qu'une grande partie des hardware précédent ont subi avec succès des attaques soft dirigé DUMP, et donc il est possible qu'il ai été nécessaire de changer de fournisseur. Il se pourrait que ce soit carrément le même micro que les cartes DSS-HU (**TMS370C-P3**), mais tout ceci ne sont que des hypothèses.

En ce qui concerne les caractéristiques techniques, la carte fonctionne en deux intervalles de tension différentes : 2.7-3.3V et 4.5-5.5V (tensions typique, respectivement de la famille logique LOW-VOLTAGE et standard) avec relèvement des niveaux hors spécifique et parfois conséquent RESET. La fourchette de fréquences acceptées s'étend de 0.9 MHZ à environs 6 MHZ, avec un relèvement des valeurs hors bande et RESET conséquent. Il semblerait que la dispersion des valeurs assumés de cette valeur supérieure soit plutôt haute. La carte ne semble pas complètement protégée des GLITCH (bug de hardware causant des problèmes), certains GLITCH provoquent des modifications à l'exécution du logiciel de gestion ou FIRMWARE, sans qu'il y ai de RESET, mais on ne trouve pas systématiquement une utilisation utile (même si nous avons pas ce biais, ouvert un chemin d'expérimentation). Dans la V7 , nous n'avons pas une utilisation massive de la RANDOM DELAY ROUTINE, mais le Jitter de la CLOCK, présent sur la V6 a disparut. Le Jitter rend très difficile (pour ne pas dire impossible) l'utilisation de l'UNLOOPER.

En utilisant l'oscilloscope digital, on peut ouvrir de nouvelles routes dans la compréhension des phénomènes qui se passe à l'intérieur de la carte. La méthode consiste à analyser l'absorption des courants (SIMPLE POWER ANALYSIS), en interposant par exemple une résistance entre la PIN du GND de la carte et la masse. Si le carte ne contient pas de protection contre la SIMPLE POWER ANALYSIS ou contre la DIFFERENTIAL POWER ANALYSIS, la consommation du microcontrôleur révèle des informations sur l'état des registre et/ou sur le type de OPCODE en exécution à ce moment là.





Dans le premier graphique, nous observons le comportement pendant les cycles d'horloge suivant immédiatement le RESET, en deux mesures distinctes (bleu et magenta). Nous voyons clairement les cycles d'horloge dans laquelle vient appelée la « RANDOM DELAY ROUTINE » (indiquée par les flèches). Dans le second graphique (avec une échelle temporelle différente), on observe une consommation pendant l'exécution d'une C1 38.

On note l'incréméntation de consommation due presque avec certitude à la présence d'un crypto-processeur. La chose la plus surprenante, est la durée de l'impulsion à forte consommation, vu qu'il est constant indépendamment de l'horloge utilisée, ceci pourrait suggérer la présence d'un PLL interne

(...)

Les trois pics de durées brèves et de forte consommation devraient se référer à la vérification de la signature.

(...)

Le problème principal est de comprendre, si l'analyse du courant absorbé, est une bonne méthode pour obtenir des informations sur les microcontrôleurs modernes (**AT90SC, P8WE, DS5002FP...**), la réponse est sans aucun doute affirmative car (et nous le verrons plus tard), on ne note pas de protection contre la SINGLE POWER ANALYSIS, n'y dans le CORE, n'y dans le crypto-processeur, la seule contre mesure implantée semblant être l'introduction du RANDOM DELAY ROUTINE (une contre mesure facile à contourner en analyse de donnée, mais un problème en phase de glitching).

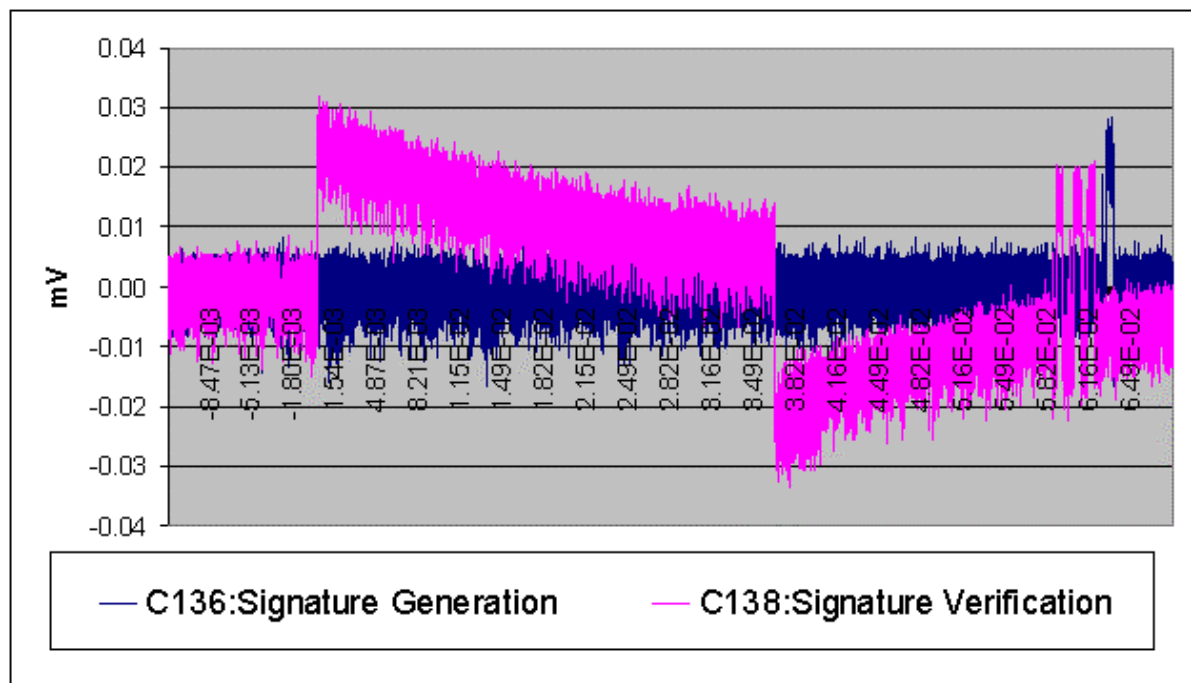
(...)

Nous ne savons pas quel pourrait être le microprocesseur utilisé, probablement il s'agit du PHILIPS ou du ATMEL car ils ont les mêmes caractéristiques électriques (intervalle de tension et fréquence) et date de la présumée production de la V7.0 ; de plus la comparaison entre le POWER TRACE (pendant le RESET, pendant le DE-ENVELOPPE et la vérification de la signature) montre que la V7.0 et la V7.1, utilisent le même hardware.

Il est probable que la V7.3 aussi, soit dans le même cas, mais non n'avons pas pu encore le vérifier.

POWER ANALYSIS – Génération et calcul de la signature

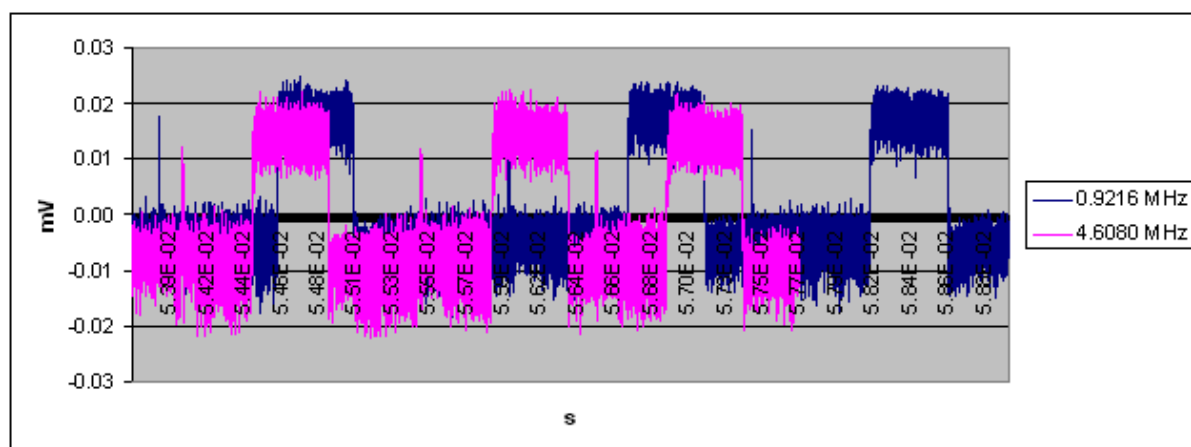
Dans le graphique suivant, nous avons confronté les courants absorbés pendant l'exécution d'une C1 38 et d'une C1 36, afin de comparer le processus de génération et celui de vérification de la signature.



Card clocked @4.608 MHz -

1) En utilisant une C1 38 avec signature valide, on observe 3 impulsions à forte consommation, après environ 20 ms de l'exécution de la DE-ENVELOPPE (identifiable par une impulsion longue). Il n'est pas dit que les impulsions soient nécessairement 3, en utilisant des instructions de longueurs différentes, on peut obtenir 1+ 2 impulsions (comme dans le graphique), mais aussi 2+2, 3+2, 4+2, confirmant la vérification de la signature par blocs de 64 octets (512 bits).

2) La commande C1 36, présente une seule impulsion (dans le cas en examen). En allongeant la réponse, nous devrions avoir la possibilité d'obtenir un nombre d'impulsion variable de 1 à 4.



3) En réduisant la fréquence de l'horloge à 0.9216 MHz, la durée des pics (y compris le large di DE-ENVELOPPE) se maintient constant.

On en conclu que :

- Génération et vérification de la signature ne sont pas symétriques
- Le crypto-processeur existe et a une horloge autonome (comme par exemple le PHILIPS FRAMEX CRYPTO-ENGINE)

En poursuivant l'analyse de la signature, nous pouvons faire des tests relatifs aux statuts 9002_A et 9002_B

4) En utilisant une C1 38 avec signature erronée (statut 9002_B), on continue à observer les 3 (ou plus) impulsions comme dans le cas d'une signature valide.

5) En utilisant une C1 38 sans NANO 82 (statut 9002_A), il y a encore 3 impulsions, ils apparaissent 10 ms après le DE-ENVELOPPE. Ceci est un bug : la signature est calculée même en absence de la NANO 82.

6) En utilisant une C1 38 sans NANO 82 et avec un P2 tel de demander une clé inexistante, on observe le même comportement que dans le point 2.
La signature est calculée sans l'initialisation du KEY-BUFFER. Dans le cas où n'importe quel autre contrôle rate, la signature n'est pas vérifiée.

POWER ANALYSI – la phase de SSE (ou DE-ENVELOPPE)

A l'intérieur d'une impulsion à forte consommation, on peut observer, en filtrant opportunément le signal lu, des choses plutôt intéressantes, sont présentes 66 itérations égales à 550 μ s (dont l'absorption dépend de la valeur assumée par les données en élaborations) à intervalle de 8 ms pour un courant à consommation plus faible.

En agrandissant une de ces itérations, on observe 24 sous-cycles, tous différents entre eux (et différents pour chaque une de 66 itérations principales).

POWER ANALYSIS – Analyse du bug 9600

En envoyant une C1 38 avec zéro données à décrypter du type :

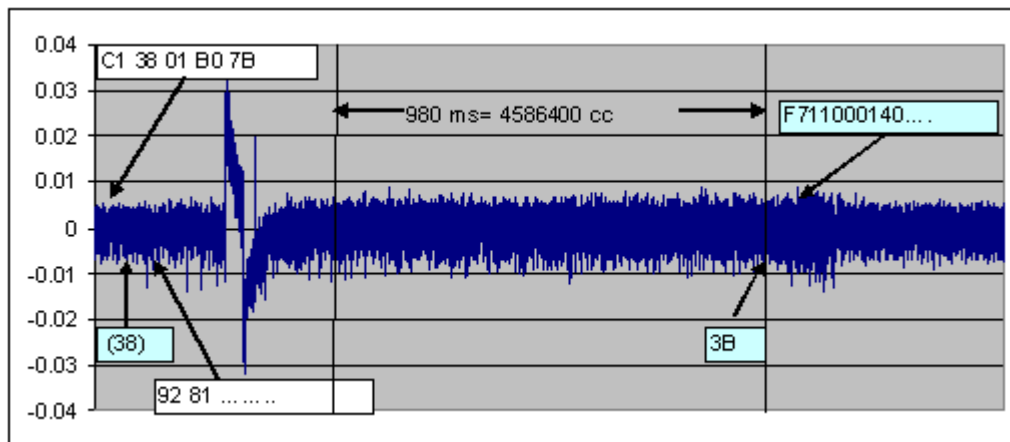
C1 38 P1 P2 LEN 9x y1

Nous aurons comme réponse l'ATR.

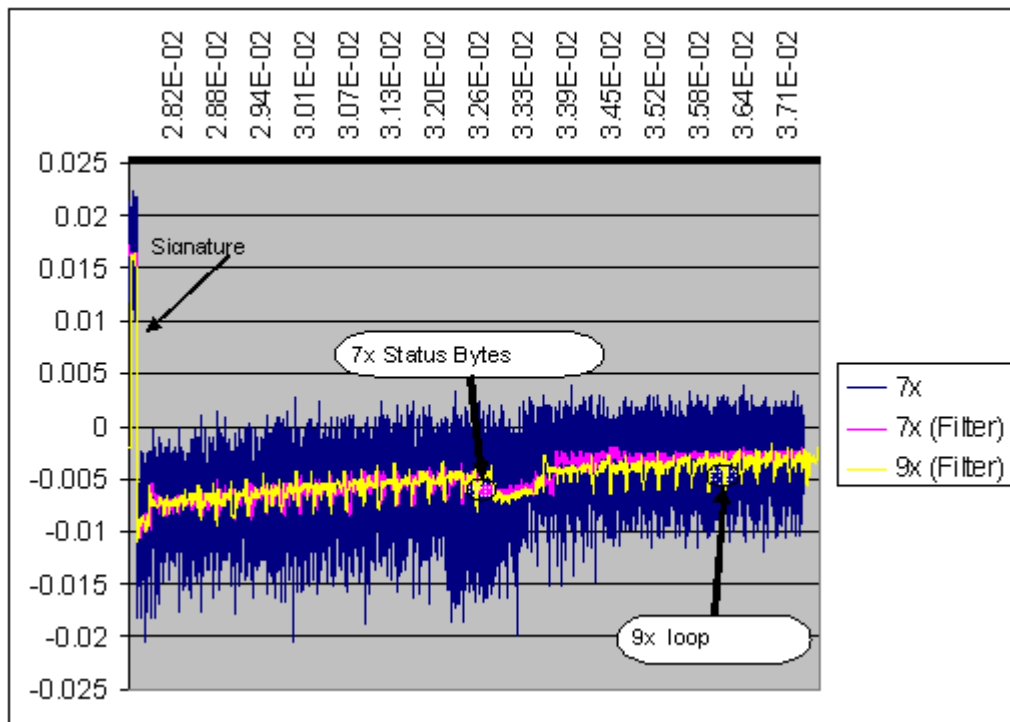
L'exécution de l'instruction aura évidemment une partie commune à la C1 38 normale et une partie commune à la routine de l'envoi de l'ATR.

Ceci sera aussi le cas dans la quantité de courant absorbé.

Le graphique suivant représente le fonctionnement typique pour une instruction de type indiqué. On note le long intervalle de temps qu'il faut entre l'envoi de l'instruction et la restitution de l'ATR (au environs de 1400 ms).



Essayons d'agrandir une partie du graphique :



En confrontant le cas P3 = 9x avec le cas P3 = 7x (qui répond normalement), on observe que pour les deux, est exécuté le DECRYPT d'un bloc (les 16 cycles sont clairement visibles dans les traces filtrées), après quoi la carte entre dans une boucle plutôt longue qui se conclut avec la restitution de l'ATR. La boucle a une durée supérieure aux 4.500.000 cycle d'horloge (hypothèse : cela pourrait être suffisant pour lire toute la RAM, ROM et EEPROM de la carte, y comprise une possible aire protégée, ce qui provoquerait le RESET). La routine de l'ATR est exécutée à partir de son point réel de départ, mais sans les tests sur le crypto-processeur et sur le générateur de nombre aléatoires, on ne voit pas les relatifs absorptions de courant, qui deviennent visibles, suite à un normal RESET de la carte.

LAST BUT NOT LEAST

ICON OF COIL qui est ce ?

Paraphrasant les paroles d'un célèbre auteur on pourrais dire que « ICON OF COIL EST CELUI QUI CROIT EN LUI » je voudrais pourtant casser certains mythes qui circulent sur la toile en précisant que :

GOYANSUPERSAYAN n'est pas l'auteur de cette DOC.

JACK70 n'est pas l'auteur de cette DOC

UOMORAGNO n'est pas l'auteur de cette DOC

(A ce dernier des remerciements pour avoir rendu disponible la documentation sur le forum qu'il modère)

Pour ceux qui ont déjà imprimé la v3.02

Celles qui suivent sont les pages ajoutées et/ou modifiées par rapport à la vielle édition (certaines ne sont que des modification esthétiques ou corrections lexico - grammaticales

Pages. 1, 5, 7, 8, 9, 10, 11, 12, 19, 20, 22, 26, 27, 33, 33 bis, 36, 37, 42, 43 et de la page 59 en suivant.

Des erreurs se sont introduites dans les page 72 et 73 de cette FAQ elles seront corrigées, après vérification, dans la prochaine version.

En préparation :

- La méthode des clés gémelles (théorie)
- Calcul de la signature (modèle possible)
- MASKING : reconstruction des XOR et des permutations
- CRYPT, DECRYPT & collisions : reconstruction des tables HASH.

**That's All Folks
Icon of Coil**

E2 82 AC 20 AC ...