

# Icon of Coil

## Capire gli ECM/EMM

- Quarta Edizione -

*In questo documento sono raccolte gran parte delle informazioni circolanti in rete che riguardano l'argomento SEKA 2, senza pretesa di essere un trattato esaustivo. Per la comprensione dell'argomento affrontato e' opportuna la conoscenza del SEKA 1, attraverso la lettura di FAQ & Prontuari comunemente reperibili in rete. Tutto quello che trovate scritto in questo documento serve per soli scopi di ricerca e di studio. La visione dei programmi delle Pay-Tv senza la sottoscrizione di un regolare abbonamento è vietata. L'autore non può essere ritenuto responsabile di eventuali danni o abusi commessi da terze parti.*

### La novita' del SEKA 2

A partire da un ECM loggato e' possibile effettuare una prima caratterizzazione del nuovo sistema:

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]

Tra due ECM diversi cambiano tutti i byte, ad eccezione di quelli evidenziati; dopo di essi non è riconoscibile nessun Nanocomando e non vi e' traccia della Signature: la ragione sta nel fatto che i dati sono stati criptati. E' ormai noto che nel SEKA 2 siano presenti due passaggi di crypt: la **SuperEncryption** (che opera su blocchi di byte) e la Secondary SuperEncryption (**SSE** oppure **ENVELOPE**), che agisce sui dati nella loro interezza. Questi processi di crypt sono **obbligatori** per i dati delle INS 3C/38/40.

L'algoritmo utilizzato per l'Envelope non e' noto, anche se e' stato ipotizzato un (de)crypt tipo RSA (per chi e' interessato esistono in Rete molti doc sull'argomento), la SuperEncryption, invece, e' costituita dallo stesso algoritmo del SEKA 1 a cui e' stato aggiunto un processo di mascheramento degli ottetti.

#### **SuperEncryption (SEKA 1)**

Si prende il corpo dell'istruzione e si suddivide in blocchi di 8 byte; a ciascun blocco si applica l'algoritmo di criptaggio con la chiave indicata da P2. Se rimangono dei byte in numero minore di 8 che non completano un blocco restano non codificati.

*L'obbligo di SuperEncryption viene fissato dal contenuto di alcuni flag memorizzati in EEPROM... per adesso non e' noto un modo per modificarli.*

## La struttura delle INS 38/3C/40

E' stata mantenuta l'impostazione classica del SEKA 1, in pieno rispetto dello standard ISO 7816:

**C1 38/3C/40 P1 P2 LEN + DATA PACKET**

### **P1**

- Bit 0...3 : Indice del Provider a cui inviare il comando
- Bit 4 : Utilizzo di Key primaria oppure Key primaria + secondaria
- Bit 5,6 : Indicano come inizializzare l'hashbuffer per il calcolo Signature
- Bit 7 : Non utilizzato

### **P2**

- Bit 0...3 : Indice della key da usare per il (de)crypt SuperEncryption
- Bit 4 : *Significato sconosciuto (deve essere = 1)*
- Bit 5,6 : Selezionano le tabelle per il (de)crypt e l'eventuale user ALGO (non attivo su V7)
- Bit 7 : Indicatore della SuperEncryption (deve essere = 1)

### **LEN**

E' la lunghezza del "data packet"

Esempio:

C1 40 01 B1 5C

**10 01** 12 08 F5 56 DA 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

Come già accennato, i due byte evidenziati si mantengono uguali per tutte le INS loggate, essi costituiscono dei parametri per il de-envelope e vengono gestiti in maniera diversa rispetto a tutti gli altri dati. E' stata data loro la definizione di parametri **P3** e **P4**. All'interno dei dati e' presente un ulteriore parametro che non e' visibile perche' criptato. La sua funzione sara' spiegata in seguito, per adesso e' sufficiente conoscerne l'esistenza, sapere che corrisponde all'ultimo byte e che e' stato definito parametro **P5**. Alla luce di tutto cio' possiamo dare una nuova rappresentazione della struttura delle INS:

**C1 38/3C/40 P1 P2 LEN P3 P4 + DATA PACKET + (P5)**

## Processo di esecuzione delle INS (card V7.0)

La sequenza logica (e temporale) dei controlli effettuati per l'invio di una C1 38/3C/40 e' la seguente :

- 1 - Normali controlli sul protocollo ISO7816 CLA/INS/LEN (possibili status d'errore 6D00, 6E00, 6700)
- 2 - Controllo su P1, esistenza del Provider (status 9004 se inesistente)
- 3 - Controllo sul bit 4 di P2 (deve essere = 1, status 9024 in caso contrario)
- 4 - Controllo sul bit 0 di P4 (deve essere = 1, status 9024 in caso contrario)
- 5 - Controllo su P3

Il trattamento riservato a **P3** e' simile a quello di un Nanocomando, pero' :

- Deve avere come valore minimo 10 (status 9024 per valori inferiori)
- Viene completamente ignorato, assieme ai dati ad esso associati, per i seguenti processi:
  - > Decrypt SSE (de-envelope) e decrypt SE
  - > Parsing ed esecuzione*("Parsing", tradotto letteralmente, vuol dire "Analisi del periodo"; nel nostro contesto significa "esame dei Nanocomandi e dei loro parametri")*
- Fa parte dei dati per il calcolo Signature

Per sapere quanti sono i dati associati a P3, si segue la classica tabella della LEN dei nanocomandi :

High Nibble | Nano Length

-----+-----

0 ... C		0 ... 12	( <i>IMPORTANTE: '0' non e' un</i>
D		16 byte	<i>valore ammissibile per P3)</i>
E		24 byte	
F		32 byte	

Osservazione: **P4** e' il primo byte di dati per **P3**.

Al termine di questo controllo si puo' ottenere lo status 6700 (LEN errata) se, dopo i dati di P3, non ci sono almeno 0x5A byte (90, in decimale), il motivo e' spiegato piu' avanti.

- 6 - Controllo sui bit 1,2 di P4 (devono essere = 0, status 9036 in caso contrario)

*P4 e' un indicatore per il de-envelope, ma la funzione precisa e' sconosciuta.*

**NOTA:** la sequenza di passi 1...6 non giustifica le seguenti risposte:

C1 38 01 91 00	Status previsto : 9024	Status ottenuto : 9036
C1 38 01 91 01 19	Status previsto : 6700	Status ottenuto : 9036

## 7 - Decrypt SSE (o de-envelope)

Pur non essendo conosciuto l'algoritmo, si e' potuto osservare che esso utilizza chiavi differenti per ciascun provider (ma uguali per tutte le card) e che opera sempre e solo sugli ultimi 0x5A byte di dati. Le conseguenze sono che:

- Un ECM/EMM indirizzato ad un provider non e' applicabile ad un provider diverso
- I dati di P3 (che non prendono parte al de-envelope) devono essere seguiti da almeno 0x5A byte (ecco spiegato il perche' dello status 6700 durante i controlli su P3)
- Eventuali byte aggiunti tra i dati di P3 e gli ultimi 0x5A byte non sono in SSE/envelope (*importante*)

Esempio:

*(NOTA: tutti i comandi mostrati sono fittizi, cioe' non corrispondono a dati realmente loggati)*

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 DA 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
```

In questo comando i byte evidenziati in giallo rappresentano P3 e relativi dati (high-nibble di P3 = 4, quindi 4 byte di dati), quelli evidenziati in celeste rappresentano i 0x5A byte sotto envelope, gli altri sono byte fuori dall'envelope (quindi sono byte in chiaro oppure sotto SuperEncryption).

*Cosa troviamo sotto il de-envelope ?*

Si nota immediatamente una grossa differenza rispetto al SEKA 1: la Signature e' esterna alla SuperEncryption e non e' in posizione fissa. Riprendendo il comando dell'esempio precedente ed ipotizzandone un possibile de-envelope:

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 00 P5
```

**(P5 ora e' in chiaro)**

...possiamo vedere P3 con i suoi dati (in giallo), seguito dai byte criptati con la SuperEncryption (in verde; e' importante ricordare che la parte sottolineata non ha subito de-envelope), dopo e' presente il Nano 82 + la Signature in chiaro ed il resto sono byte di riempimento fino al penultimo (compreso).

Il comando si conclude col parametro **P5**. Esso ha la funzione di indicare dove e' posizionato il Nano 82, cio' significa che la card non ricerca la signature scandendo tutto il comando alla ricerca del Nano, ma va "a colpo sicuro" alla posizione puntata da P5. Si hanno due **vincoli** sulla posizione dell'82:

- La Signature e' lunga 8 byte, non e' quindi ammissibile che il nano 82 possa stare nelle ultime **9** posizioni del comando;
- I dati per il calcolo Signature devono essere almeno 8, in altre parole il Nano 82 deve essere preceduto da almeno 8 byte;

#### 8 - Controlli sul valore assunto da P5

Il conto che fa la card per determinare la posizione del Nano 82 e' il seguente (calcolo ad 8 bit):

$$\text{Posiz.82} = \text{LEN} - (\text{P5} + 8)$$

I valori di P5 compresi tra 128 e 255 non sono ammissibili : si ha lo status 9037. Tale status si ottiene anche quando P5 assume valori nel range 1÷127, ma il risultato del calcolo non rispetta i **vincoli** del precedente punto '7'. Vi e' un ulteriore controllo (ancora oggetto di studio e dal meccanismo non conosciuto) che se, non superato, fa ottenere lo status 9038. La probabilita' di ottenere questo status al variare dei byte in envelope e' circa uguale a  $(1/3) \cdot (1/256)$ .

#### 9 - Controllo sul bit 7 di P2 (deve essere = 1, status 9035 in caso contrario)

E' il bit indicante la SuperEncryption.

#### 10 - Ricerca Key per il calcolo Signature

Per il calcolo Signature e' necessaria la key indicata da P2; a seconda della INS considerata non tutte le key sono utilizzabili:

INS 40 : Solo Management Key (Key Index nel range 0...B), in caso contrario si ha status 9013

INS 3C : Solo Operational Key (Key Index nel range C...F), in caso contrario si ha status 904A

INS 38 : Nessun vincolo sulla Key utilizzabile

A questo punto inizia la ricerca della Key in EEPROM, se non e' presente si ottiene lo status 901D, nel caso di Key primaria, oppure lo status 901F, se non e' stata trovata la Key secondaria. Se si utilizza la key 0F, l'eventuale secondaria non viene considerata (*nemmeno se il bit 4 di P1 = 1*). Una volta reperita la Key, ne viene verificato il checksum, se questo controllo fallisce si ha lo status 9028.

ATTENZIONE : lo status 9028 non appare solo in occasione del fallimento del controllo-checksum. Esso puo' comparire anche in conseguenza del fallimento di uno dei controlli precedenti il de-envelope, ma il meccanismo non e' stato ancora chiarito. I due casi si distinguono attraverso il tempo di risposta...

**9028 pre-SSE** : circa 73000 cicli di clock

**9028 key-checksum** : oltre 190000 cicli di clock

## 11 - Verifica presenza Nano 82 e calcolo signature

La card prende il valore calcolato al punto 8 e va a verificare se il Nano 82 e' effettivamente presente in tale posizione. Se non viene trovato si ha lo status 9002 (chiamato anche 9002\_A oppure 9002 tipo-1). Se il Nano 82 e' presente, allora si procede col calcolo Signature. I byte da cui dipende il valore della Signature comprendono tutti e soli i dati che precedono il Nano 82 fino a P3 incluso.

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 00 P5
```

Nell'esempio, tutti i byte evidenziati in celeste prendono parte al calcolo signature.

La key per il calcolo signature, come gia' detto, e' indicata dal nibble basso di P2, mentre i bit 6,5 indicano le tabelle hash da usare.

*Le "Tabelle hash" sono delle tabelle di conversione dati utilizzate dall'algoritmo di (de)crypt : cambiando le tabelle, cambia il risultato del (de)crypt.*

Bit 6   Bit 5	
-----+-----+-----	
0	0   Tabelle ROM (uguali al SEKA1)
0	1   Tabelle EEPROM (A)
1	0   Algoritmo "alternativo"
1	1   Tabelle EEPROM (B)

Se (bit 6, bit 5) = (1,0) si richiede l'esecuzione del (de)crypt tramite un algoritmo alternativo, memorizzabile nell'EEPROM della card. Nelle V7 tale algoritmo non e' presente: un'eventuale INS inviata con (bit 6, bit 5) = (1,0) otterrebbe lo status 9034.

*L'analisi dei bit 6,5 avviene immediatamente prima della verifica della presenza del Nano 82.*

E' giunto il momento del calcolo Signature: e' interessante notare che la lunghezza del comando non influenza il tempo impiegato per il calcolo, questo lascia pensare che esso sia eseguito tramite hardware dedicato (crypto-processore). Se la Signature calcolata e' diversa da quella presente nel comando si ha lo status 9002 (chiamato anche 9002\_B oppure 9002 tipo-2).

9002\_A e 9002\_B differiscono per i tempi di risposta: **time(9002\_B) > time(9002\_A)**

## 12 - Decrypt SuperEncryption

La fase di messa in chiaro dei dati in SE e' composta di due parti : unmasking + decrypt, ed agisce sui byte compresi tra i dati di P3 ed il 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
```

Osservazione: il processo agisce in maniera lievemente diversa per le INS 3C, analogamente a quanto accadeva nel SEKA 1.

### SuperEncryption

In caso di encryption delle C1 3C, non viene decryptato il primo ottetto di dati.

*NOTA: Nel SEKA 1 gli ottetti incompleti erano lasciati in chiaro, nel SEKA 2, invece, il processo di mascheramento (masking) non lascia nessun dato visibile.*

## 13 - Pre-parsing del corpo dell'INS ed esecuzione del comando

Dopo la SuperEncryption abbiamo una normalissima INS tipo SEKA 1, con i canonici nanocomandi, ma non si e' ancora giunti alla loro esecuzione. Esiste una fase di pre-analisi dei dati, che dovrebbe controllare se, "saltando" di nano in nano si arriva sull'82 (Nano signature). Un esempio rendera' tutto piu' chiaro:

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 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 C3 6C
C4 77 74 8B ... ecc. ecc.
```

L'analisi dei nanocomandi procede da sinistra verso destra...

- 1 + 1 byte      10 01 - P3 e P4, non considerati nell'esecuzione del comando, vengono saltati
- 1 + 32 byte    Nano F0 piu' 32 byte di dati (Customer Word Pointer bitmap)
- 1 + 2 byte    Nano 22 - Controllo data fine abbonamento
- 1 + 2 byte    Nano 21 - Impostazione data fine abbonamento
- 1 + 1 + 8 byte Nano 90 - Scrittura Key primaria (Indice 5D)
- 1 + 1 + 8 byte Nano 90 - Scrittura Key primaria (Indice 5C)
- 1 + 1 + 8 byte Nano 90 - Scrittura Key primaria (Indice 5E)

A questo punto si giunge al Nano 82 ed il processo termina correttamente. Supponendo, invece, di avere questo comando:

C1 40 01 B1 5C

10 01 [F0] 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 [41] 5E EF AB 5A [90] 5E BC 6F 91 [82] C9 2D C3  
6C C4 77 74 8B ... ecc. ecc.

1 + 1 byte      10 01 - P3 e P4, non considerati nell'esecuzione del comando, vengono saltati  
1 + 32 byte      Nano F0 piu' 32 byte di dati (Customer Word Pointer bitmap)  
1 + 2 byte      Nano 22 - Controllo data fine abbonamento  
1 + 2 byte      Nano 21 - Impostazione data fine abbonamento  
1 + 1 + 8 byte   Nano 90 - Scrittura Key primaria (Indice 5D)  
1 + 1 + 8 byte   Nano 90 - Scrittura Key primaria (Indice 5C)  
1 + 4 byte      Nano 41 - Scrittura PPUA

Qui si giungerebbe al Nano 90, che richiederebbe 9 byte di dati, oltrepassando il Nano 82 !!!  
Questo e' un esempio di parsing errato: In tal caso si ha lo status 9600 (errore di pre-parsing). Inoltre, risulta:

#### Status 9600 (citazione)

Se nel corpo dell'INS e' presente un nano 82 (oltre a quello normalmente presente in coda ai dati) si ha, in ogni caso, lo status 9600, pero' potrebbe darsi che se l'82 fosse a distanza minore di 8 byte da quello "vero" il preparsing potrebbe andare a buon fine...dipende da come e' implementato il controllo (*differenza di puntatori e salto condizionato dal carry?*)

Per le INS 3C e 40 esiste un'ulteriore controllo che, se non superato, fa ottenere lo status 9A00.

#### Status 9A00 (citazione)

Significato: E' stato incontrato in fase di pre-parsing un Nano riservato ad un'altra INS

Nano che rispondono 9A 00 se incontrati in ambito ECM: 1C 24 81 90 91 F6 F7

Nano che rispondono 9A 00 se incontrati in ambito EMM: 31 D1

Su 6.2 questo controllo veniva effettuato in fase di parsing, per cui tutti i nano parsati prima del nano riservato incontrato venivano eseguiti correttamente. Su 7.0 questo controllo è effettuato nel corso della verifica sul pre-parsing.

**Se tutti questi controlli vengono superati, il comando va (finalmente) in esecuzione.**

*Nelle card di versione successiva alla 7.0, alcuni controlli sono stati parzialmente rivisti : sembra, ad esempio, che lo status 9037 non sia piu' presente (oppure abbia significato diverso) e che, probabilmente, al suo posto si ottenga lo status 67 00.*



*(Versione riveduta di un post trovato in un Forum-sat)*

01) Verifica aderenza ISO	-> fallisce ->	6700, 6D00 o 6E00
02) P1: Esiste il provider LoNibble(P1)?	-> se no	-> 9004
03) Verifica ATR protection level	-> se si	-> 9B00
04) Verifica ATR level 3	-> se si	-> 9024
05) P2: bit4=1 ? (presenza par. extra)	-> se 0	-> 9024
06) P4: bit0=1 ?	-> se 0	-> 9024
07) P3: HiByte(P3) > 0 ?	-> se 0	-> 9024 ( <i>P3 senza parametri</i> )
08) P3: LL-Tab[HiByte(P3)] >= 5B ?	-> se < 5B	-> 6700 ( <i>&lt;90 byte sotto SSE</i> )

High Nibble	Lunghezza (hex)
0x00 -- 0x0C	0x00 -- 0x0C
0x0D	0x10
0x0E	0x18
0x0F	0x20

10) P5: valore in chiaro < o uguale a 8?-> se si	-> 9037
11) P5: valore in chiaro > MaxLen ?	-> se si -> 9037
12) P5: valore lecito ?	-> se no -> 9038 <i>** da studiare **</i>
13) P2: bit7=1 ? (SE obbligatoria)	-> se 0 -> 9035
14) P5: valore in chiaro punta a 82 ?	-> se no -> 9002a (82 assente)

15) P2: INS=3C e key operativa -> se no -> 904A  
P2: INS=40 e key di management -> se no -> 9013

16) P1 P2: Esiste la key ? -> se no -> 901D (prim.)/901F (sec.)  
*ECCEZIONE: La key 0F secondaria NON viene presa in considerazione per C138/C13C*

17) Verifica checksum della key -> se non ok -> 9028

-----  
CALCOLA SIGNATURE DELLA PARTE DA P3 A 82  
(escluso P3, i suoi parametri ed 82, tutto in SE)  
-----

18) INS=40, ATR level2, keyindex >7x      -> se si      -> 9022  
19) Seleziona tabelle hash (P2 bit 6,5) -> se user algo-> 9034  
20) Confronta con la signa del comando -> se diversa -> 9002b

-----  
DECRIPTA SE  
-----

21) Verifica parsing      -> se errato      -> 9600  
22) INS=3C verifica Nano "proibiti"      -> se presenti -> 9A00  
    INS=40 verifica Nano "proibiti"      -> se presenti -> 9A00

### Significato degli Status Byte (solo V7.0)

0101 Errore in fase di test-hardware: RAM interna  
0102 Errore in fase di test-hardware: EEPROM  
0104 Errore in fase di test-hardware: RAM esterna  
0105 Errore in fase di test-hardware: Dispositivi aggiuntivi (es: generatore di numeri casuali)  
0110 Errore in fase di test-hardware: ROM checksum errato  
  
6700 Lunghezza errata  
6B00 Parametro/reference byte (P1 o P2) errato  
6D00 Istruzione non supportata o protetta  
6E00 Classe non supportata (ammesso solo il valore C1)  
  
9000 Comando eseguito senza errori  
9001 Errore in fase di scrittura EEPROM  
9002 Nano 82 mancante (9002\_A) oppure signature errata (9002\_B)  
9003 Impossibile creare/modificare un record per il provider indicato (non autorizzato)  
9004 Provider inesistente  
9005 INS 40/44 : nano ammesso solo per il provider 00 (SEKA)  
9006 Spazio mancante per memorizzazione record o creazione provider  
9007 Provider non gestibile col Nano 24 (bloccato tramite INS 40 - Nano 01)  
9008 INS 40 - Nano 01 : non ammesso  
9009 INS 40 : PPUA mancante nel Bitmap del Nano F0  
9010 INS 30 : PIN errato  
9011 Istruzione non abilitata (testbit non correttamente impostato)  
9013 INS 40/44 : Key errata , ammesse solo le MK  
9014 Istruzione precedente errata  
    INS 02 non preceduta da INS 04, INS 32 non preceduta da INS 34  
    INS 36 non preceduta da INS 38, INS 3A non preceduta da INS 3C  
    INS 56/5A non preceduta da INS 5C, INS 42/8A non preceduta da INS 44  
    INS 40 - Nano 87 non preceduto da INS 06, INS 40/44 - Nano F6 non preceduto dal Nano 1B  
    INS 44 - Nano F7 non preceduto dal Nano 50  
9015 INS 32/34/36/38 : Valore "d1" errato  
    INS 3C : Nanocomando non abilitato (Flag Nano 2C, Flag Nano 15)  
    INS 40 : Nanocomando non abilitato (Flag Nano 32, Flag Nano 42, Flag Nano 80)  
9016 INS 40 - Nano 87 : errore nei dati forniti  
    INS 44 : provider non abilitato  
9017 INS 50/54/56 non abilitate (flag non correttamente impostato)  
    INS AC non abilitata (flag non correttamente impostato)

9018 Errore in fase di creazione provider (Provider già esistente)  
 9019 INS 40 - Nano 23 : provider creato con successo  
 INS 40 - Nano 25 : provider eliminato con successo  
 INS 40 - Nano 41 : PPUA modificata con successo  
 901A INS 3C - Nano 15/2C : errore in fase di acquisto gettoni  
 901B INS 3C - Nano 15/2C : gettoni insufficienti  
 INS 3C : Credit Record inesistente  
 901C INS 3C - Nano 15/2C : gettoni esauriti  
 901D Primary Key non disponibile  
 901E INS 40 - Nano 43 : impossibile aggiungere ulteriori gettoni (overflow)  
 901F Secondary Key non disponibile  
 9020 INS 44 - Nano 90/91 : Key Index errato  
 9021 INS 04 : Key errata, ammessa solo key 0F  
 9022 INS 40 : Key index errato (in ATR level 2)  
 9023 INS 44 : il Nano 71 deve precedere il Nano D1  
 9024 Istruzione non permessa in ATR level 3  
 INS 38/3C/40 : parametro/reference byte errato (SE/SSE flags)  
 9025 INS 44 - Nano D1 : il contatore ha raggiunto il valore 00  
 9026 INS 3C - Nano 31 : evento già visionato (nessuna decodifica)  
 9027 INS 3C - Nano 19 : decodifica in Preview  
 9028 Errore nel contenuto del key record (checksum errato)  
 INS B4 : checksum diverso da quello calcolato  
 Errore in fase di inizializzazione de-envelope  
 9029 INS 44 - Nano F6 : checksum errato  
 902B INS 44 - Nano 50/92 : errore in fase di esecuzione del nanocomando  
 902C INS 40 - Nano 50/92 : errore nel primo parametro (fuori intervallo)  
 902D INS 40 - Nano 50/92 : EEPROM flag non correttamente impostati  
 902E INS 40 - Nano 50/92 : errore nell'ultimo parametro  
 902F INS 40 - Nano F7 non preceduto dal Nano 50  
 9030 INS 40 - Nano 20/92 : errore in fase di verifica dell'user algo  
 9031 INS 40/44 - Nano 1B : flag non correttamente impostato  
 INS 40 - Nano E0 : errore in fase di verifica tabella hash  
 INS AC : parametro/reference byte non corretto  
 9033 INS 40/44 - Nano 1B error (flag non correttamente impostato)  
 INS 44 - Nano 90/91 : errore in fase di selezione tabella hash  
 INS 44 - Nano E0 : errore in fase di verifica tabella hash  
 9034 Errore in fase di encryption/decryption con user algo (algoritmo non attivo)  
 9035 INS 36/38/3C/40 : Superencryption obbligatoria (superencryption flag abilitato)  
 9036 INS 38/3C/40 : parametro/reference byte errato (P4 bit 0,1)  
 9037 INS 38/3C/40 : P5 fuori intervallo  
 9038 INS 38/3C/40 : valore di P5 errato  
 904A INS 3C : key errata, ammesse solo le Key Operative  
 9090 INS 40 : EEPROM non modificata (vedi INS 48)  
 90A0 INS 40 : EEPROM modificata (vedi INS 48)  
 9301 Data di scadenza superata  
 9302 INS 3C - Nano D1 : nessuna decodifica  
 9304 INS 3C - Nano 12 : visione protetta tramite Parental Control  
 9305 INS 3C - Nano F1 : trasmissione non destinata a questa zona geografica  
 9401 INS 36 : valore P1 errato  
 9402 INS 32/36 : valore di P2 errato  
 9600 INS 38/3C/40 : Errore di pre-parsing  
 INS 3C - Nano 15/2C : evento nullo  
 96xx INS 3C : xx Nano significativi eseguiti correttamente, ma nessun Nano D1 incontrato  
 INS 40 : xx Nano significativi eseguiti correttamente, ma errore di parsing (solo SEKA 1)  
 97xx Aggiornamento Eeprom non necessario per alcuni Nano. 'xx' è il Bitmap di questi Nano  
 98xx INS 40 - Nano 32 : evento nullo ('xx' nano significativi eseguiti correttamente)  
 99xx INS 40 - Nano 30/42 : Data errata ('xx' nano significativi eseguiti correttamente)  
 9A00 INS 40/3C : Nano non abilitato, INS 56 : Key Index errato  
 9Axx INS 44 : Nano non abilitato ('xx' nano significativi eseguiti correttamente)  
 9B00 Istruzione non ammessa in ATR protection level (card bloccata)

## Analisi del pre-parsing - Status 9600

### INS 40

Il pre-parsing per l'INS 40 funziona esattamente come descritto in precedenza, cioè la card esamina i nanocomandi presenti nell'istruzione, calcolando la posizione del nanocomando successivo in funzione della lunghezza teorica del Nano considerato in quel momento. Ad un certo punto si giungerà al Nano della signature oppure lo si oltrepasserà. Nel primo caso, il controllo è superato, nel secondo caso no e si otterrà lo status 9600. Da notare che non è necessario che un Nanocomando incontrato sia significativo (cioè che abbia una effettiva funzione).

### INS 3C - Il funny-bug 2

Anche per l'INS 3C il pre-parsing funziona come per la INS 40; inizialmente era stata ipotizzata la presenza del funny-bug, con meccanismo di funzionamento simile a quello presente nelle precedenti versioni della card (cioè in caso di "scavalco" della signature in fase di parsing).

#### **Funny bug SEKA 1 (citazione)**

Ci troviamo alla conclusione del processo di esame dei nanocomandi relativi all'istruzione 3C; in particolare si tratta di sommare il numero di byte dell'ultimo nanocomando al numero totale di byte processati e verificare se si sia raggiunta o meno la lunghezza complessiva del comando. Osservando la ROM notiamo tale comparazione ma, al contrario di un corretto algoritmo, non segue una routine di controllo ma si passa direttamente alle fasi di chiusura (a differenza dell'INS 40). Infatti segue la copia del buffer dei record in eeprom e il mascheramento del buffer dati. È proprio il passaggio diretto a tale fase di chiusura che attiva il bug: **ne deriva la scrittura del record buffer in eeprom, nel primo record.** Solitamente nel record buffer si trova la chiave usata nel comando, mentre i registri che puntano al numero di record sono resettati, indirizzando al primo. Se viene processato un nanocomando sconosciuto tale da provocare il superamento della lunghezza totale del comando, la chiave (o eventualmente qualsiasi altra struttura presente nel buffer dei record) viene copiata sul record 1 della eeprom.

In realtà la causa sembrerebbe diversa :

#### **Funny bug SEKA 2 (citazione)**

Il Funny Bug scatta solo ed esclusivamente per **P3** tali da scavalcare la Signature

Il bug è stato citato per completezza, sebbene non siano state pubblicate prove **certe** della sua esistenza.

### INS 38 - Il bug 38/36

Qui si osserva un bug macroscopico, che e' stato prontamente corretto sulle card V7.1: la routine di controllo della vecchia C1 38 e' stata rimossa e sostituita dalla stessa routine di pre-parsing dell'INS 40.

In pratica, il controllo sul pre-parsing viene superato se i dati sono organizzati secondo i criteri dell'INS 40 e non dell'INS 38 ! Vediamo come funzionava il pre-parsing della C1 38 in SEKA 1:

#### Istruzioni 36 e 38 - Lettura record e signature (citazione)

Questa coppia di istruzioni esegue la lettura dei record presenti sulla carta, con la gestione della signature sia sul comando inviato, sia sulla risposta generata. Inoltre permette di modificare alcuni byte dei record Bx eventualmente presenti. Prima va eseguita la 38 ( invio dati ) e poi la 36 ( richiesta dati ).

**C1 38 P1 P2 1A 16 xx 2A mm mm 2B nn nn 86 b0 b1 b2 b3 b4 b5 b6 b7 82 SIGNATURE**

I valori **16**, **2A**, **2B**, **86** non sono Nanocomandi veri e propri, è necessario che siano nella posizione indicata, altrimenti si ha errore **96 00**. *(Ecco quindi come funzionava il pre-parsing dell'INS 38 in SEKA 1, si trattava di un controllo sulla presenza degli pseudo-nano in ben precise posizioni, NdA)*

P1 e P2 hanno lo stesso significato che per l'istruzione 40, si possono usare PK oppure PK+SK ed è inoltre utilizzabile la SuperEncryption.

Il valore 'xx' indica il tipo di record da dumpare, mentre i byte 'mm mm' si comportano in maniera analoga ai dati dell'INS 34.

<b>00 vv vv</b>	Provider Package Bitmap record	
<b>01 vv vv</b>	Provider PPV Credit record	<b>vv vv</b> valori qualunque
<b>03 xx xx</b>	Provider PPV record	<b>xx xx</b> contiene l'EventID della trasmissione PPV
<b>04 vv yy</b>	Record Ex	<b>yy</b> specifica il tipo Record Ex da cercare
<b>06 zz zz</b>	Record generici	<b>zz zz</b> è il <b>Numero Record</b> da cui iniziare la lettura

Attenzione: nel caso in cui si abbia 'xx' uguale a **03** oltre a mostrare i PPV record, l'istruzione va a modificare il PPV-event spot, cioè il decimo ed undicesimo byte ( contando da sinistra ) del record trattato inserendo i due valori che seguono **2B**. Dopo c'è il valore **86** seguito da 8 byte.

Il dump vero e proprio viene eseguito con l'istruzione 36

**C1 36 P1 P2 LEN**

P1 deve essere uguale a P1 dell'istruzione precedente con, in più, il sesto bit posto ad uno.  
P2 indica, come sempre, la Key. Se il bit più significativo è settato, la risposta della carta è criptata.  
LEN deve essere maggiore di 13 ( esadecimale ).

L'esecuzione di questa istruzione va a buon fine solo se è preceduta da un'istruzione 38. La risposta all'istruzione 36 ha la seguente forma: il primo byte è la lunghezza della risposta, si ha poi il valore 86 seguito dagli 8 byte inseriti con l'istruzione 38, infine si ha il dump dei record secondo lo stesso significato dell'istruzione 32 seguiti dal valore 82 e dalla signature calcolata sulla risposta. E' importante notare che questa è l'unica istruzione che ha la risposta con signature.

Adesso non e' piu' necessaria la presenza dei valori 16, 2A, 2C, 86, da cui il **bug 36/38**. Infatti, se e' sufficiente seguire i canoni dell'INS 40, allora un'INS 38 col corpo di una C1 40 e' eseguita senza errori !

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]
```

Cambiando 40 in 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]
```

Attenzione: provando a fare la stessa cosa con un INS 3C non si ottiene analogo risultato:

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] oppure [90 37]
```

Conclusione: a parità di byte criptati, il de-envelope delle INS 3C porge un risultato diverso da quello delle INS 40/38. Potrebbe essere diversa la key utilizzata, oppure l'inizializzazione dell'algoritmo, oppure diversa la gestione di P5 che, comunque, si trova nella solita posizione (*dimostrabile, NdA*).

Nelle C1 40 d'esempio gli status byte sono sempre 90 00, in realtà esistono tantissime C1 40 con status diversi ed ugualmente utilizzabili per il bug. Gli status più frequenti sono questi:

- 97 xy
- 90 19
- 93 01
- 90 09

Torniamo alla C1 38 con status 90 00 ... questo risultato ci permette di lanciare con successo una C1 36 (si avrebbe altrimenti lo status 9014).

**C1 36 P1 P2 LEN**

Analizziamo i parametri di questa INS:

**P1** deve essere uguale a **2y** oppure **3y** dove “y” deve essere uguale al nibble basso di P1 della C1 38 precedentemente inviata, se tale vincolo non e' rispettato si ha lo status 9401.

**P2** indica la Key e le tabelle hash da usare per criptare la risposta (stesso significato delle C1 38/3C/40).

**LEN** deve essere maggiore di 13 ( esadecimale ), altrimenti si ha lo status 6700.

La risposta che si ottiene dipende dal contenuto del corpo dell'INS 38. Quest'ultima, se inviata seguendo rigorosamente la sintassi richiesta, avrebbe questa forma (dopo che la card ha decriptato il comando):

**C1 38 P1 P2 LEN**

```
P3 + dati di 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 00 P5
```

Ricordando che, a causa del bug, la card non controlla la presenza dei “nano”, ma prende direttamente “**d1**”, “**d2 d3**”, “**d4 d5**”, “**b0 ... b7**” risultera' un prelevamento dei parametri secondo il seguente criterio (contando i byte a partire da quello immediatamente successivo ai dati di P3) ...

- il secondo per d1
- il quarto e quinto per d2 d3
- il settimo e ottavo per d4 d5
- dal decimo al diciassettesimo (compresi) per b0 ... b7

Esempio:

```
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
```

Trasformo in 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]
```

I dati vengono prelevati dopo che sono stati messi in chiaro. Ipotizzando che il decrypt sia il seguente...

```
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
```

... si capisce quali sono i dati prelevati dalla card (in giallo) ed utilizzati per la successiva C1 36. Essa puo' generare una risposta in chiaro oppure criptata la cui lunghezza ed il cui significato variano a seconda del valore assunto da **d1**.



Risposta con 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 + bytes ad FF [90 35]
```

In questo caso la risposta e' in chiaro, sono presenti alcuni parametri caratteristici (detti pseudo-nano):

Pseudo-nano 86 - E' seguito dai byte b0...b7 dell'INS 38 (vedi)

Pseudo-nano 83 - Precede il Bitmap Package per il Provider indicato dal low nibble di P1

Pseudo-nano 04 - Indica che non ci sono ulteriori informazioni

Nano 82 - E' quello della Signature (se la risposta e' in chiaro, la Signature non e' presente)

Dopo sono trasmessi tanti byte a FF quanti sono necessari ad arrivare alla giusta lunghezza della risposta.

```
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]
```

Questa e' una risposta criptata, rimane riconoscibile il solo Nano 82, seguito dalla signature. E' inoltre presente un byte in piu' rispetto alla C1 36 in chiaro, che e' quello immediatamente evidenziato in giallo (1C, in decimale 28). Tale valore indica quanti dati significativi ci sono nella risposta e che sono tutti quelli che lo seguono fino alla signature compresa.

Risposta con 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]
```

Richiedendo una risposta alla C1 36 in chiaro...

```
C1 36 21 00 LEN
36
86 b0 b1 b2 b3 b4 b5 b6 b7
84 YY YY YY YY YY YY YY YY
04 82 + bytes ad FF [90 35]
```

Pseudo-nano 86 - E' seguito dai byte b0...b7 dell'INS 38 (vedi)

Pseudo-nano 84 - Precede il Credit Record per il Provider indicato dal low nibble di P1

Pseudo-nano 04 - Indica che non ci sono ulteriori informazioni

Nano 82 - E' quello della Signature (se la risposta e' in chiaro, la Signature non e' presente)

Dopo sono trasmessi tanti byte a FF quanti sono necessari ad arrivare alla LEN richiesta.

Richiedendo una risposta alla C1 36 criptata (ho ommesso la C1 38)...

```
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]
```

Dall'esame delle risposte criptate non si riesce a distinguere il caso d1 = 00 dal caso d1 = 01.

### Risposta con 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]
```

Al momento dell'invio della C1 36, la card esegue una ricerca in EEPROM. Tale ricerca ha lo scopo di reperire un Record Bx...

#### Record Bx : PPV Record (citazione & adattamento)

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

Il record Bx viene creato al momento dell'acquisto di un evento PPV. In pratica, con un'apposita richiesta dati il decoder verifica la presenza di questo record per un dato PPV Event Id, se presente è possibile visionare la trasmissione PPV. Il primo byte indica il PPV record-type (0 = unused / 1 = normal use / 2 = special use), i successivi due indicano l'Event ID a cui si riferisce il record. Il byte indicato con **dn** è il Numero di Diffusione. Quando uno stesso evento viene trasmesso su più canali, il PPV Event ID è lo stesso su tutti i canali mentre il numero di diffusione cambia. Il numero di diffusione cambia anche fra due successive trasmissioni dell'evento sul solito canale. Serve quindi per identificare l'evento temporalmente e spazialmente ( il canale ). Ad ogni incremento del numero di diffusione corrisponde una diminuzione del numero di visioni ancora disponibili. Se l'evento non è stato ancora visionato, assume il valore di FF Il byte **cv** è il numero di visioni ancora disponibili. Seguono i due byte **ec ec** ( gettoni associati all'evento, si aggiungono tramite INS 3C - Nano 15 ) e due byte indicanti la data della prima visione dell'evento ( **dd dd** ). Se l'evento non è ancora stato visionato, assumono il valore FF FF. Infine c'è il PPV event-spot (due byte che normalmente valgono 00), modificabile con le INS 36/38.

#### Esempio :

Supponiamo di avere il PPV Event **0E 38**

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

Quando il Film è stato acquistato, ma non è ancora stato visionato

**01 0E 38 0F 04 00 00 16 A1 00 00 B1**

Quando il Film è disponibile ed è stato visionato almeno una volta.

**0E 38** = PPV-EventID

**0F** = Numero di diffusione / **FF** ancora nessun numero di diffusione è stato fornito

**04** = Numero di visioni disponibili

**16 A1** = Data della prima visione / **FF FF** = Evento non ancora visionato

Nella C1 38 e' indicato l'Event ID da ricercare tramite i byte **d2 d3**, mentre i dati **d4 d5** rappresentano il PPV event-spot. Se nell'istruzione viene specificato un Event ID che non è stato memorizzato nella carta, sarà considerato l'Event ID che ha il valore ( esadecimale ) immediatamente superiore a quello specificato. Di conseguenza l'Event-ID 00 00 individua in ogni caso un PPV Event memorizzato.

Se e' stato trovato l'event ID si ha la seguente risposta in chiaro...

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 YY

**04 82** + bytes ad FF [90 35]

Pseudo-nano 86 - E' seguito dai byte b0...b7 dell'INS 38 (vedi)

Pseudo-nano B1 - Precede il PPV record

Pseudo-nano 04 - Indica che non ci sono ulteriori informazioni

Nano 82 - E' quello della Signature (se la risposta e' in chiaro, la Signature non e' presente)

Dopo sono trasmessi tanti byte a FF quanti sono necessari ad arrivare alla LEN richiesta.

Risposta criptata...

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 DA 72 9F 99 D4 70 ...

[97 04]

La risposta e' piu' lunga rispetto ai casi precedenti. Se sono presenti piu' PPV record e' possibile fare una richiesta multipla con una singola C1 36; basta utilizzare valori sufficientemente grandi per LEN.

E' necessario a questo punto commentare lo status che si e' ottenuto, perche' e' indicativo di una *tentata* scrittura su EEPROM.

#### PPV-Spot (citazione)

Nel caso in cui si abbia **d1** uguale a 03 oltre a mostrare il PPV record, l'istruzione va a modificare il PPV-event spot, cioe' il decimo ed undicesimo byte ( contando da sinistra ) del record trattato inserendo i valori **d4 d5**.

### Risposta con d1 = 04 : Record Ex

Contrariamente a quanto scritto in molti doc, per ottenere questa risposta non e' necessario indirizzare il comando al Provider 00 00 (provider SEKA).

```
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]
```

A seconda del valore di **d3** , si seleziona il Record Ex di cui interessano i dati (d3 deve essere uguale al primo byte del record)

Richiesta in chiaro...

```
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 + bytes ad FF [90 35]
```

Pseudo-nano 86 - E' seguito dai byte b0...b7 dell'INS 38 (vedi)

Pseudo-nano B2 - Precede il SEKA Record (i primi 0x0B byte di un record E0)

Pseudo-nano 04 - Indica che non ci sono ulteriori informazioni

Nano 82 - E' quello della Signature (se la risposta e' in chiaro, la Signature non e' presente)

Risposta criptata...

```
C1 36 21 90 LEN
36
1F 22 ED 88 F1 77 36 13
00 CD EB 42 42 4C 95 69
DA 8D 13 B1 AB 48 5A 82
D4 17 33 69 FE 5E 15 32 ...
[90 00]
```

Da notare che la lunghezza della risposta e' la stessa del d1 = 03, e' pero' possibile distinguere i due casi osservando gli status byte che, in questo caso, assumono sempre valore 9000.

### Risposta con d1 = 06 : Record generico

In questo caso si vanno a leggere i record "generici", cioè nel formato in cui sono memorizzati sulla carta (leggibili anche con l'INS 32/34). Il record visualizzato e' il primo che contiene dati relativi al provider indicato dal low-nibble di P1, la ricerca del record parte da quello specificato nell'INS 38 attraverso i byte **d2 d3**.

```
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]
```

Risposta in chiaro...

```
C1 36 21 00 LEN
36
86 b0 b1 b2 b3 b4 b5 b6 b7
D2 ii ii yy yy yy yy yy yy yy yy yy yy yy yy 00 00
03 82 + bytes ad FF [90 35]
```

Pseudo-nano 86 - E' seguito dai byte b0...b7 dell'INS 38 (vedi)

Pseudo-nano D2 - Precede il Record

I dati associati a D2 sono organizzati in questo modo:

- 0x02 byte usati per mostrare l'indice del record
- 0x0C byte indicanti il contenuto del record
- 0x02 byte uguali a 00 00

Pseudo-nano 03 - Possibili ulteriori informazioni

Nano 82 - E' quello della Signature (se la risposta e' in chiaro, la Signature non e' presente)

Risposta criptata...

```
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]
```

E' possibile il dump di piu' record con una singola C1 36; basta utilizzare LEN sufficientemente grandi.

Il caso piu' frequente: le risposte "corte"

Capita molto spesso che, in risposta ad una C1 36, si ottenga qualcosa del genere:

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]

Si parla in questi casi di "risposte corte", cioe' con il minimo numero di dati significativi. Il corrispondente contenuto in chiaro puo' assumere le seguenti forme

C1 36 21 00 LEN

36

86 b0 b1 b2 b3 b4 b5 b6 b7

04 82 + bytes ad FF [90 35]

C1 36 21 00 LEN

36

86 b0 b1 b2 b3 b4 b5 b6 b7

03 82 + bytes ad FF [90 35]

La prima risposta (pseudo-nano 04) si ottiene soltanto nelle condizioni indicati dalla seguente tabella:

Valore assunto da "d1"	Valore assunto da LEN (hex)	Condizione al contorno
00	maggiore di 1D	Bitmap package non presente
01	maggiore di 1D	Credit record non presente
03	maggiore di 20	Evento PPV con ID maggiore o uguale a d2 d3 non presente
04	maggiore di 20	Record Ex con primo byte uguale a d3 non presente
06	maggiore di 25	Record con index maggiore o uguale a d2 d3 non presente

La seconda risposta (pseudo nano 03) si ottiene in varie occasioni. Un modo e' quello di utilizzare valori di d1 diversi da 00, 01, 03, 04, 06. I restanti modi sono indicati nella seguente tabella:

Valore assunto da "d1"	Valore assunto da LEN (hex)	Condizione al contorno
00	minore o uguale a 1D	Bitmap package non presente
01	minore o uguale a 1D	Credit record non presente
03	minore o uguale a 20	Evento PPV con ID maggiore o uguale a d2 d3 non presente
04	minore o uguale a 20	Record Ex con primo byte uguale a d3 non presente
06	minore o uguale a 25	Record con index maggiore o uguale a d2 d3 non presente

Nella descrizione delle possibili risposte alla C1 36 e' stata fatta la distinzione tra risposta criptata e risposta in chiaro. Vediamo adesso come e' possibile generare quest'ultima.

### C1 36 in chiaro di tipo 1 - Flag SuperEncryption

Se il bit 7 di P2, indicante la SuperEncryption, e' uguale a 0 si ha una condizione di errore, perche' sulle 7.0 la SuperEncryption e' obbligatoria anche per le C1 36 ed infatti la card risponde con lo status 90 35. Prima dello status, pero', ci viene "regalata" dalla card la risposta in chiaro, fino al Nano 82 (la Signature e' calcolata sui dati criptati, in assenza di essi dopo il Nano 82 troviamo solo byte di riempimento uguali a FF).

Esempio:

```
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 in chiaro di tipo 2 - Crypt con user algo

Se il bit 7 di P2 e' correttamente settato, ma i bit 5,6 sono tali che P2 = Cx oppure Dx, il processo di crypt della risposta non puo' avere luogo, perche' l'user algo non e' presente/attivo sulle 7.0. La card allora risponde con la risposta in chiaro, seguita dallo status 9034.

Esempio:

```
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 in chiaro di tipo 3 – Il bug sulla LEN

Puo' capitare che la card risponda con l'INS in chiaro (anche se non richiesta), seguita dallo status 9015. Questo accade quando si utilizza per **d1** un valore diverso da 00, 01, 03, 04, 06 e si supera una certa LEN (dipendente dal **d1** specificato).

Esempio:

```
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]
```



Questo comportamento, noto col nome di "bug sulla LEN", permette il dump di una piccola parte di ROM.

### Il bug sulla LEN (citazione)

Il bug presente sulle V7 e' discendente diretto del bug sulla len presente nel C134/32 delle card V6.0 e inferiori. Il bug era il seguente:

```
C1 34 00 00 03 XX 00 00
C1 32 00 00 YY
```

A seconda del valore assunto da 'XX' si ha un valore minimo ammissibile per 'YY', in funzione di una tabella contenuta in ROM. Sappiamo che i valori standard per 'XX' sono 00,01,03,04,06 ma la ROM non ci vieta di usarne altri!

Per valutare la lunghezza minima si usa una tabella i cui elementi sono letti sommando all'indirizzo base (della tabella) il valore XX, senza nessun controllo.

Se ad XX sostituisco valori > 06 vado a leggere 'indirettamente' un pezzo di ROM (i byte successivi alla tabella)...ho detto indirettamente perche' devo fare in questo modo:

```
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
```

Incrementare la LEN della C1 32 finche' non cambia la risposta della card...*(Finche non si ha lo status 9015, NdA)* quando cio' avviene ho trovato il valore del byte (il byte vale 'YY-1').

Sulle V7 il bug e' stato tolto dalla C1 34/32 ma e' rimasto nella C1 36/38! Si puo' azzardare un dump di parte della ROM, ma ci vuole molta pazienza ed un bel numero di INS. La C1 38 seka 1 aveva la seguente struttura:

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

dove 'XX' ha lo stesso significato della C1 34, quindi dobbiamo 'giocare' sul suo analogo presente nel comando seka2.

L'unica cosa da ricordarsi e' che la C1 36 ammette come LEN minima 0x14, perche' con valori inferiori si ha lo status 67 00, e che ,a differenza delle precedenti versioni di kard, si puo' superare tranquillamente il valore 0x5Fh.

*(omissis)*

il calcolo della LEN minima avviene pescando i dati da una tabella in ROM. A questo punto la gestione del valore trovato segue due strade diverse, a seconda che si tratti di un INS 32 oppure un INS 36, perche' quest'ultima restituisce in output gli stessi dati della 32 piu' altra roba (...) in totale fanno 20 byte in piu' (0x13 in esadecimale). Per le C1 36 ci si riconduce al valore presente in ROM prendendo la LEN che da' come status 90 15 e togliendo 0x14.

*(omissis)*

- 9015 non e' contemplato per i valori 'normali' (00,01,03,04,06) di **d1**.

- Quando hai una C1 36 che non da' 9015 per **nessuna** LEN vuol dire che **d1** punta ad un valore in ROM maggiore o uguale ad EC oppure per il motivo del punto precedente. I due casi sono normalmente distinguibili, verificando la transizione (o meno) nano 03 --> nano 04 a partire da una certa LEN.

### Contenuto della ROM letta attraverso il bug sulla LEN

```
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
```

I byte mancanti non sono determinabili. Di essi si puo' solo sapere che hanno valore maggiore od uguale a **0xEC** (*vedi pagina precedente, NdA*).

## Costruzione di una C1 38/40 "custom" a partire da un EMM

Per questa esercitazione sono necessari :

SmartTimer 1.6 o versioni superiori

EMM di attivazione o EMM di aggiornamento chiavi operative

antercedenti il 27/01/2003, altresì detto "**Black Monday**"

### **Fase 1 : Preparazione del comando iniziale :**

Nel caso si disponga di un EMM loggato precam esso ci appare nelle seguente forma :

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

Senza addentrarsi troppo nella struttura di un EMM precam, possiamo distinguere le parti piu' importanti e necessarie a ricostruirne l'equivalente destinato alla card:

**65** Lunghezza del comando precam

**06xxxyzz** PPUA a cui è indirizzato l'EMM

**0070** Provider a cui è indirizzato l'EMM (in questo caso prov **01** : per la relazione  
Provider ID ↔ Numero Provider, vedi tabella)

**B0** Chiave con cui è criptato e firmato l'EMM in questo caso MK 00

### **Tabella Provider**

Tipo card Versione	Numero Provider (Low-Nibble P1)				
	0	1	2	3	4
<b>ITA</b> <b>(V7.0)</b>	<b>0000</b> SEKA	<b>0070</b> TEL£+DIGIT@L£	<b>0071</b> +C@LCIO	<b>0072</b> STREAM	<b>0073</b> P@LC0
<b>FRA</b> <b>(V7.1)</b>	<b>0000</b>	<b>0080</b> C@N@LSATELLIT£	<b>0081</b> C@NAL+	<b>0082</b> SPARE-A	assente
<b>POL</b> <b>(V7.0)</b>	<b>0000</b>	<b>0065</b> CYFR@+			assente
<b>ESP</b> <b>(V7.0)</b>	<b>0000</b>	<b>0064</b> C@N@LSATÉLIT£	<b>0066</b> C@N@LSATÉLITE2	<b>0067</b> C@N@LSATÉLITE3	assente
<b>HOL</b> <b>(V7.3)</b>	<b>0000</b>	<b>006A</b> C@N@LDIGITAAL	<b>006B</b> OPERATOR 2	<b>006C</b> OPERATOR 3	<b>006D</b> OPERATOR 4

Altri Provider ID esistenti : 0076 - 0084 - 0086 (PRO TV) - 0088 (AB S@T)

La conversione dell'EMM nella forma equivalente destinata alla card si ottiene in questa maniera: prima di tutto si calcola la lunghezza reale del comando, sulla base della lunghezza del comando precam (nell'esempio, 65):

$$\text{LEN} = (\text{Lunghezza comando precam} - 4)$$

Poi si compila il comando:

C1 38 01 B0 61 10 01 509392031...

Nel caso, invece, si disponga di un EMM loggato sarà sufficiente cambiare la INS 40 in INS 38, eliminare gli status byte ed il byte di ACK.

*A seconda del software di logging utilizzato l'ACK può non essere presente. Tale byte si trova tra la LEN del comando ed i dati, esso assume lo stesso valore dell'INS.*

#### Esempio con ACK-byte

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

#### Esempio senza ACK-byte

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

A questo punto possiamo inviare il comando:

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 seguito del quale invieremo la INS complementare alla 38 ovvero la INS 36. Essa ha la possibilità di restituire, sia in forma criptata che in chiaro, dei dati che sono dipendenti dalla INS 38 immediatamente precedente ed in funzione della chiave e delle tabelle di crypt selezionate dai parametri P1 e P2. Tramite un semplice gioco di variazione dei parametri P1 e P2 andiamo alla ricerca di una risposta che soddisfi le condizioni imposte dai nuovi algoritmi di crypt.

## Fase 2 : Esecuzione ciclo INS 38 e INS 36 :

Come già accennato nella descrizione delle INS 36/38, esistono dei vincoli per l'esecuzione dei comandi.

### Parametri di P1 e P2 ammessi nella INS 36

Per i valori ammissibili si farà riferimento ai casi utili nella pratica, per una trattazione rigorosa della sintassi si faccia riferimento alla descrizione dell'INS (pag.13).

P1 High-nibble	Puo' assumere il valore <b>2</b> per tutti i valori del Low-Nibble di P2 Puo' assumere il valore <b>3</b> solo per i valori del Low-Nibble di P2 uguali a 0, 1
P1 Low-nibble	<b>Deve</b> assumere lo stesso valore dei <b>P1 Low-nibble</b> della C1 38
P2 High-nibble	Puo' assumere i valori 9, B, F <i>(Con 'D' non si può avere risposta criptata vedi pag.24 : C1 36 in chiaro di Tipo 2, NdA)</i>
P2 Low-nibble	Puo' assumere i valori 0, 1, (2, 3), C, D, E ... tenendo però conto che C, D, E non sono utilizzabili per creare EMM, ma solo ECM...

Oververo valori ammissibili di P1 e P2 (esempio con **C1 38 01 B0 LEN 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 **31 B0** LL, C1 36 **31 F0** LL  
C1 36 **31 91** LL, C1 36 **31 B1** LL, C1 36 **31 F1** LL

E, qualora se ne disponga :

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

Il ciclo di invio INS 38 + INS 36 sarà quindi:

**Invio C1 38 P1 P2 LEN P3 P4 + DATI**  
**Invio C1 36 P1 P2 LL**

Per **LL** useremo il valore **14** (hex):  
in questo modo si otterranno solo  
risposte "corte".

Tale ciclo terminerà una volta che tutte le combinazioni possibili riguardanti le INS 36 sono state inviate. Le risposte ad ogni singola INS 36 vanno annotate: al termine della prova si utilizzeranno solo quelle che avranno la forma imposta dai parametri P3 e P4 della nuova decodifica.

### Forma ammissibile della risposta alla C1 36 (risposte "corte")

#### 1) La forma della risposta deve essere obbligatoriamente criptata

Una veloce verifica per sapere se si e' ottenuto una risposta criptata e' tramite l'analisi degli status byte: se assumono il valore 90 00, la risposta e' sicuramente criptata.

2) I primi due byte della risposta (che diventeranno il P3 e P4 della C1 38) devono obbedire alla regola :

**P3 Nibble alto maggiore di 1 e minore di 9**

**P4 Nibble basso uguale a 1 o uguale a 9**

Ovvero valori ammissibili della risposta:

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.....	...

### Fase 3 : Costruzione nuova INS "custom " ed aggiramento Envelope

Nel caso si ottenga una risposta valida, cioe' corrispondente ai vincoli della Fase 2, possiamo andare a creare la nuova INS "custom", come esempio utilizzeremo una risposta del tipo:

```
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]
```

Eliminiamo l'ACK-byte (36, se presente), la length della risposta (13) e status byte (90 00), in modo da avere

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

Componiamo il nuovo comando:

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

ed aggiungiamo almeno 0x5A byte con valore pari a 0x00: essi rappresentano i byte che subiranno il de-envelope ma che non prenderanno parte a calcolo Signature ed esecuzione, grazie alla possibilita' di aggiramento dell'envelope, tramite la procedura spiegata piu' avanti.

E' consigliabile, anche per cio' che si andra' a realizzare nelle successive prove, partire con una length **LL = 0x75** ovvero occorre aggiungere 0x62 byte di valore pari a 0x00. Di questi 0x62 byte i **primi 8 post-signature saranno in "chiaro" e fuori envelope**, i **restanti 0x5A** saranno i byte in envelope (vedere la teoria sul processo di esecuzione delle INS per ulteriori chiarimenti).

Alla fine avremo :

```
C1 38 21 B0 75
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
```

Se inviamo questa INS otterremo lo status 9002 (possibili anche 9037 e 9038, a seconda del provider), perche' l'envelope non è stato ancora aggirato. L'aggiramento si ottiene cambiando il valore dell'ultimo byte dell'INS da inviare incrementandolo fino a che si passi dallo status 9002/9037/9038 allo status 9600. Cio' accade perche' la variazione di un byte si riflette sul risultato di tutto il de-envelope, in particolare sul valore assunto da **P5**, quando si ha il valore "giusto" (tale che si indirizzi correttamente il Nano 82 del comando che abbiamo costruito) allora siamo riusciti ad aggirare l'envelope. (NOTA: Potremmo cambiare uno qualsiasi dei byte in envelope - *gli ultimi 0x5A byte* - ma per comodita' e un po' per convenzione si utilizza l'ultimo byte. Non sempre esiste un valore per l'ultimo byte tale da generare l'aggiramento dell'envelope, allora si varia anche il penultimo byte).

A pagina 33 e' presente la tabella che indica i giusti valori da assegnare al penultimo ed ultimo byte (NLB/LB) in funzione del numero di byte in chiaro (post-signature) dell'INS.

Consultando la tabella deriviamo :

INS = **38**, Byte in chiaro = **08**, Provider **1** → NLB = **0x00**, LB = **0x3C**

```
C1 38 21 B0 75
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 3C
```

→ [96 00]

Qualora ci si trovi in una situazione non prevista in tabella si puo' fare la variazione a "mano" oppure ci si appoggia alla funzione "Find P5" del programma SmartTimer (finestra CMD studio).

Osservazione: I valori riportati in tabella non sono gli unici validi, possono esistere altre coppie NLB/LB che danno medesimo risultato: sono elencati i primi che si trovano eseguendo la variazione a partire da 00 00.



ITALIA		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
Numero di byte in chiaro dopo la signature e prima dei 5A byte in SSE	00	00 59	00 53	01 0D	01 14	01 A8	00 3A	00 63	02 AC	00 7B	01 9B
	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

SPAGNA		INS 38 & 40				INS 3C			
		00 00	00 64	00 66	00 67	00 00	00 64	00 66	00 67
Numero di byte in chiaro dopo la signature e prima dei 5A byte in SSE	00	00 3D	00 6D	00 28	01 99	01 04	00 32	00 1C	00 0E
	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

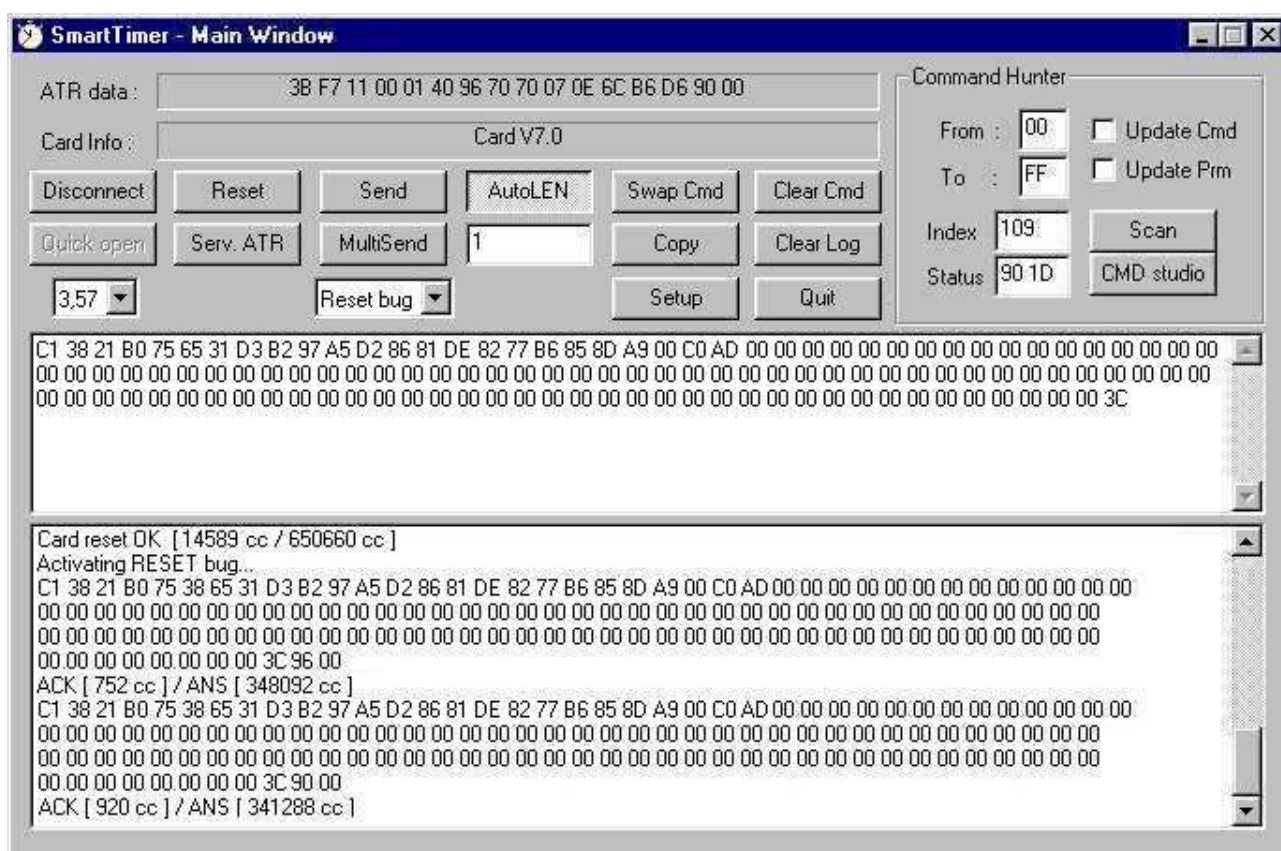
#### **Fase 4 : Reset Bug e ottenimento dello status 9000 su di una INS "custom"**

Per passare dallo status 9600 allo status 9000 occorre semplicemente eseguire un ciclo che, per il cultori delle procedure “manuali” consiste nell'effettuare ripetutamente questa procedura:

## Reset card

## Invio INS

Per i piu' pigri il tasto Multisend di SmartTimer (modalita' "Reset Bug") risolve il problema da solo. Ecco un piccolo LOG d'esempio :



## Che cos'e' il RESET-bug

*Lo status 9600 indica che il controllo sul pre-parsing non e' stato superato, allora com'e' possibile che dopo un certo numero di RESET si ottenga 9000?*

La causa e' da ricercarsi in un bug nell'implementazione dell'unmasking le cui conseguenze si manifestano quando si hanno dati in quantita' inferiore ad 8; riprendiamo il comando utilizzato nella precedente Fase 3:

C1 38 21 B0 75

```
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 3C
```

I byte in verde (*P3 e relativi parametri*) non vanno considerati per il decrypt. Restano, prima del Nano 82, solamente 3 byte, insufficienti a formare un ottetto completo: in questo caso l'unmasking "va in crisi", nel senso che il risultato diviene dipendente non solo dai dati, ma anche dal restante contenuto del command buffer e da una parte di RAM ad esso adiacente dove (tra le tante cose) sono presenti dei byte che variano ad ogni reset (*il meccanismo e' tuttora oggetto di studio*). Ad ogni RESET si ha un diverso risultato del decrypt, con la possibilita' di ottenere un parsing-corretto (da cui lo status 9000).

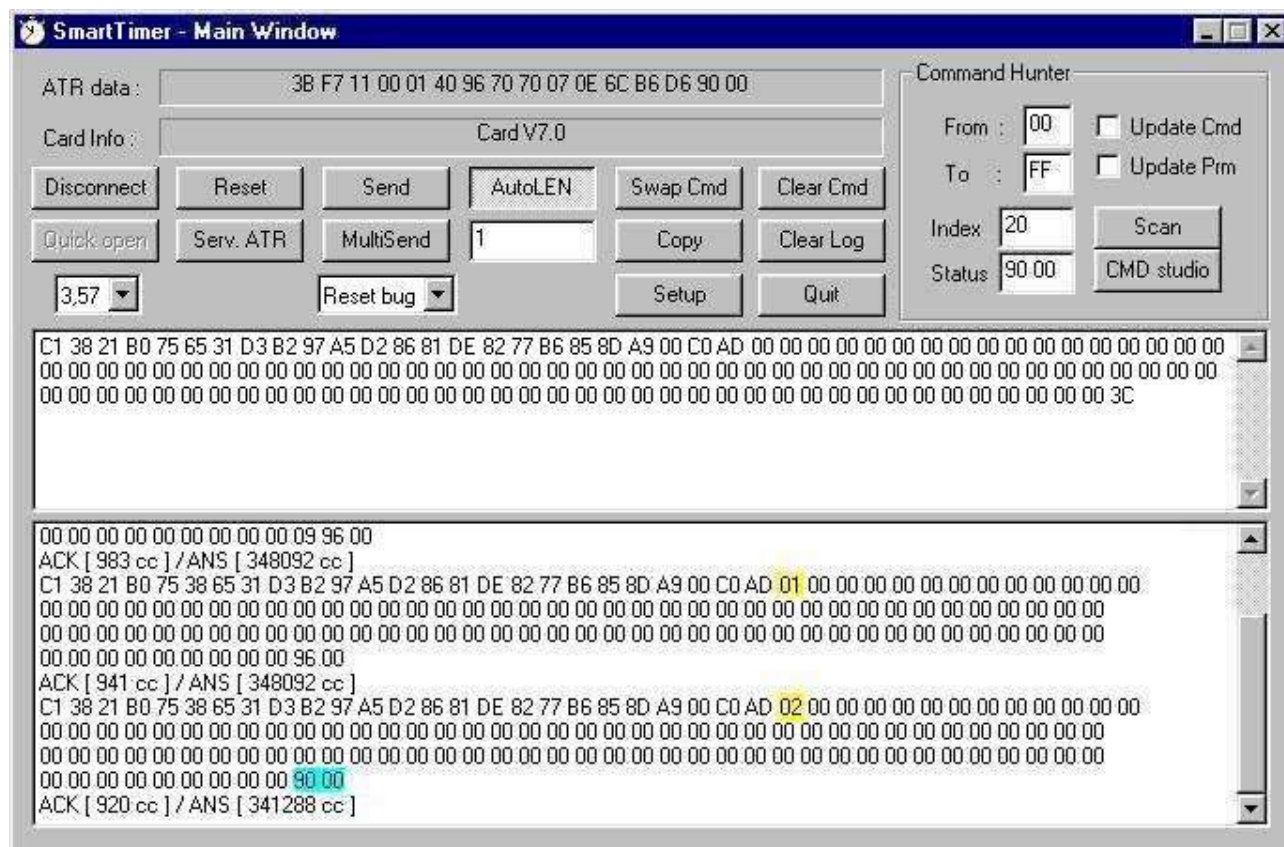
Il numero di byte disponibili per il decrypt dipende da P3: a parita' di lunghezza del comando, piu' e' grande l'high-nibble di P3 e minori sono i dati per il decrypt. Si possono avere addirittura **zero dati da decryptare**: *in tale condizione la card non porta a normale compimento l'esecuzione dell'INS (o risponde con l'ATR oppure si "blocca" finche' non si invia un RESET...non si escludono scritture impreviste in EEPROM).*

#### **Fase 4 bis : "BYTE bug" ed ottenimento dello status 9000 su di una INS "custom"**

Questo metodo ottiene lo stesso risultato del Reset Bug ma ha con se' delle proprieta' che lo rendono piu' appetibile. Il metodo consiste nella variazione del valore di uno o piu' byte posti tra l'ultimo byte della signature ed il primo byte in envelope. A titolo d'esempio, variamo il byte evidenziato nella nostra INS:

[illegible]

Anche in questo caso, per i cultori del metodo “manuale”, il programma SmartTimer ci viene in aiuto. Nel nostro caso imponiamo come "Index byte" da variare il byte 20 (nel programma è espresso in decimale), impostiamo la casella “From” a 00 e la casella “To” ad FF , la casella “Status” la settiamo a 9000 che e' la condizione di uscita del nostro test, avviamo la funzione “Scan” ed aspettiamo che SmartTimer finisca ...



## Che cos'e' il BYTE-bug (o "byte curiosi" bug)

Abbiamo detto che l'unmasking buggato va a dipendere da altri dati, oltre ai byte da mettere in chiaro, vediamo di descrivere brevemente questo comportamento, (l'analisi dettagliata sara' fatta in fase di descrizione del processo di SuperEncryption). L'unmasking sfrutta come parametro la "somma delle WORD modulo 0x4000" di una parte dei dati del corpo dell'INS da smascherare. Quando i dati sono inferiori ad 8, l'indice usato per prelevare i dati del calcolo assume valore "negativo" : cosi' si va ad estendere la somma a tutti i dati presenti del commandbuffer (che seguono i byte da smascherare) oltre ad una regione di RAM successiva. Se uno qualunque di questi dati varia, mantenendo la stessa "somma delle WORD modulo 0x4000" l'unmasking non cambia. Viceversa, cambiando la "somma delle WORD" cambia il risultato, con la possibilita' di ottenere un parsing corretto.

## Caratteristiche e proprieta' del BYTE-bug

*(Tratto da post presenti sui Forum-sat)*

Si costruisca una C1 38 (a partire da una risposta "corta" alla C1 36) con LEN abbastanza lunga da permettere la variazione dei byte dopo la signature e prima degli ultimi 0x5A.

Senza resettare mai si vari uno dei byte "in chiaro" dopo la signature finche' non si ottenga lo status 90 00. Detto "i" tale byte, si modifichi ora il byte **i+2n** o **i-2n**. Per ogni "n" tale che il nuovo byte variato sia ancora nei limiti sopra esposti (e cioè tra l'ultimo byte di signature e il primo byte antecedente gli ultimi 0x5A) si notano delle cose molto interessanti:

- La sequenza degli status 9600/9000 ottenuta è identica.
- C'e' una ripetitività del Low-nibble ogni 4 high-nibble.
- Ripetendo la stessa procedura con un corpo sotto SSE differente il fenomeno si ripete ma per altri valori.

Esempio chiarificatore:

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

Supponiamo che un byte che porta allo status 9000 sia uguale a 0x3C:

```
...82 s0 s1 s2 s3 s4 s5 s6 s7 00 3C 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 .... 90 00
...82 s0 s1 s2 s3 s4 s5 s6 s7 00 BC 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 .... 90 00
```

Esiste una logica nella sequenza di status che si ottengono. Questi byte entrano in gioco quando deve essere decriptato un ottetto incompleto.

$$(\dots)$$

In realta' i byte vanno considerati in coppia... facciamo un esempio:

```
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 00 00 00
00 00 00 00 00 1E [90 00]
```

Se siamo nella condizione "Nibble alto di P3 dispari" si suddividono i byte in chiaro a coppie, a partire dal secondo di essi. Se siamo invece nella condizione "Nibble alto di P3 pari" vale lo stesso discorso, ma le coppie di byte vanno prese a partire dal primo byte in chiaro.

```
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 1E [90 00]
```

Per adesso non si prendono in considerazione i byte sotto envelope.

Queste coppie di byte vanno viste come **quantita' a 16 bit (WORD)**, cosi' facendo si scopre che tutte le combinazioni possibili di WORD che danno luogo alla stessa **somma modulo 0x4000** (16384 in decimale) danno risultato identico !

Per il comando prima mostrato si ha  $(\text{somma WORD mod } 0x4000) = (1500 \text{ mod } 0x4000) = 1500$ .

Due comandi equivalenti potrebbero essere, ad esempio, questi:

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 1E [90 00]
```

$(\text{somma WORD mod } 0x4000) =$   
 $(5500 \text{ mod } 0x4000) = 1500$

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 1E [90 00]
```

$(\text{somma WORD mod } 0x4000) =$   
 $(1500 + 3FFE + 4002 \text{ mod } 0x4000)$   
 $= (9500 \text{ mod } 0x4000) = 1500$

Volendo fare un discorso piu' generale:

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 byte in SSE/Envelope
```

Tenendo presente la distinzione tra high-nibble di P3 pari e dispari, i byte in chiaro vanno suddivisi in WORD (nell'esempio q00...q24, ma possono essere in quantita' differente)...



High Nibble P3 dispari	High Nibble P3 pari
(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)
Il byte "spaiato" va considerato come (00 q00)	Il byte "spaiato" va considerato come (q24 00)

NOTA: il valore 00 e' stato introdotto per trattare comodamente il byte spaiato, non si riferisce a nessun byte realmente presente in RAM !

Fissati i byte P3 P4 x0 x1 x2 x3 x4 x5 x6 x7 82 s0 s1 s2 s3 s4 s5 s6 s7 e variando i byte in chiaro si ottengono al massimo 16384 decrypt diversi (in esadecimale: 0x4000), e' inoltre possibile identificare l'insieme di tutte le combinazioni che danno luogo a medesimo decrypt tramite una semplice relazione:

#### High-Nibble P3 dispari

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

#### High-Nibble P3 pari

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

Fin qui tutto bene, pero' un RESET e' sufficiente per scombinare tutto. Ad ogni RESET sembrano cambiare i decrypt del comando e quindi gli effetti. Si procede allora con un altro esperimento...

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
```

27

Si varia una WORD in chiaro del comando (i byte **XX YY**) e si annotano i valori ai quali corrisponde lo status 9000 (si costruisce la cosiddetta "**successione dei 9000**"). Dopodiche' si invia un RESET e si ripete il procedimento. La prova si effettua con le WORD da **0000** a **3FFF** : data la periodicit  di comportamento (secondo la somma WORD mod 0x4000) non serve usare altri valori.

Si ottengono due successioni di valori, il primo risultato e' che il numero totale di WORD che danno luogo a 9000 e' costante ad ogni RESET. Il secondo risultato, ben piu' importante, e' che le successioni sono tra loro legate. Prendendo i valori ricavati e calcolando i "**delta**", cioe' le differenze tra WORD consecutive, si va a costruire la "**successione delle delta**". Confrontando le due successioni si vede che si ottengono gli stessi valori, ma cambia il punto di partenza (e' come se le successioni fossero "traslate").

Successione (A)		Successione (B)	
WORD	DELTA	WORD	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
(eccetera)		(eccetera)	

Alla luce del fatto che:

- Tra la successione (A) e (B) c'e' stato un RESET, ma non sono stati variati byte del comando
- Lo status 9600/9000 dipende dalla **somma WORD mod 0x4000**

si conclude che e' possibile legare il RESET-bug al BYTE-bug, in pratica, agli effetti del pre-parsing la variabilita' dovuta al RESET puo' essere considerata come una WORD incognita da aggiungere nella somma. Infatti, una volta allineate le successioni delle 'delta' si vede che la differenza tra le WORD omologhe e' costante per tutta la sequenza (ma CAMBIA ad ogni reset):

Ecco alcuni valori d'esempio:

```
35 CD - 01 10 = 34 BD
35 DA - 01 1D = 34 BD
35 E4 - 01 27 = 34 BD
35 EB - 01 2E = 34 BD
```

Adesso e' possibile trarre la seguente conclusione:

**i byte RANDOM entrano a far parte della sommatoria delle WORD**

High-Nibble P3 dispari

$$[(\text{random WORD}) + (q_0 \ q_1) + (q_2 \ q_3) + (q_4 \ q_5) + (q_6 \ q_7) + (q_8 \ q_9) + (q_{10} \ q_{11}) + (q_{12} \ q_{13}) + (q_{14} \ q_{15}) + (q_{16} \ q_{17}) + (q_{18} \ q_{19}) + (q_{20} \ q_{21}) + (q_{22} \ q_{23}) + (q_{24} \ q_{25})] \text{ modulo } 0x4000$$

High-Nibble P3 pari

$$[(\text{random WORD}) + (q_0 \ q_1) + (q_2 \ q_3) + (q_4 \ q_5) + (q_6 \ q_7) + (q_8 \ q_9) + (q_{10} \ q_{11}) + (q_{12} \ q_{13}) + (q_{14} \ q_{15}) + (q_{16} \ q_{17}) + (q_{18} \ q_{19}) + (q_{20} \ q_{21}) + (q_{22} \ q_{23}) + (q_{24} \ q_{25})] \text{ modulo } 0x4000$$

(...)

Altra proprieta' legata al bug:

**nella somma delle WORD entrano anche i byte in envelope DOPO che sono stati decriptati**



### Fase 5a : Utilizzi di una Ins "custom" per generare EMM

Dopo che si sia ottenuto uno status 9000 mediante "Reset Bug" o "Byte Bug" si aprono varie possibilità: generare ulteriori INS "custom" mediante utilizzo di INS 38/36 (in particolare per cercare le risposte "lunghe") oppure convertire la nostra INS 38 nella corrispettiva INS 40. Analizziamo il secondo caso, ricordando che la nostra INS "custom" e' cosi' formata (caso di status 9000 ottenuto con "BYTE bug"):

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

Sostituiamo il byte 38 con il byte 40, ottenendo:

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

Ovvero nella sua forma estesa :

```
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
```

Inviando il comando **senza resettare** ed otterremo:

```
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]
```

#### ATTENZIONE !!!

I C1 40 cosi' costruiti, se inviati, possono causare l'alterazione o cancellazione dei dati memorizzati nella card, col rischio di renderla inutilizzabile !!!

(Si possono avere anche altri status)

In conclusione, l' EMM custom e' stato accettato.

Per verificare se l'EMM abbia scritto qualcosa in EEPROM si puo' utilizzare la semi-sconosciuta **INS 48**, inviando un bel :

**C1 48 00 00 01 FF**

Ad ogni C1 40 successivo ad essa, si puo' ottenere lo status **90A0** oppure **9090**, a seconda che ci sia stata una scrittura in EEPROM oppure no. L'effetto della C1 48 cessa al momento del reset della card.

### ***Fase 5b : Utilizzi di una Ins "custom" per generare altre risposte alla C1 36***

Puo' risultare interessante procedere, una volta costruita l'INS custom, alla generazione di un gran numero di altre risposte valide. Prendiamo un'INS custom generica, costruita a partire da una risposta corta, e analizziamola:

**C1 38 P1 P2 LEN**

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

**+ eventuali byte in chiaro + 90 byte in envelope**

Riferendosi ai P3, si e' gia' visto che il suo high-nibble ci indica quanti byte *non* considerare per l'esecuzione del comando, inoltre per gli high-nibble troppo grandi (maggiori di 8): la card non risponde come dovrebbe, in particolare per P3 = 9x si ottiene in uscita l'ATR e per P3 superiori non si riceve nessuna risposta

P3=1x -> dato saltato = P4

P3=2x -> dati saltati = P4 x0

P3=3x -> dati saltati = P4 x0 x1

P3=4x -> dati saltati = P4 x0 x1 x2

P3=5x -> dati saltati = P4 x0 x1 x2 x3

P3=6x -> dati saltati = P4 x0 x1 x2 x3 x4

P3=7x -> dati saltati = P4 x0 x1 x2 x3 x4 x5

P3=8x -> dati saltati = P4 x0 x1 x2 x3 x4 x5 x6

I byte in SE che verranno decriptati saranno i seguenti:

P3=1x -> byte in SE = x0 x1 x2 x3 x4 x5 x6 x7

P3=2x -> byte in SE = x1 x2 x3 x4 x5 x6 x7

P3=3x -> byte in SE = x2 x3 x4 x5 x6 x7

P3=4x -> byte in SE = x3 x4 x5 x6 x7

P3=5x -> byte in SE = x4 x5 x6 x7

P3=6x -> byte in SE = x5 x6 x7

P3=7x -> byte in SE = x6 x7

P3=8x -> byte in SE = x7

La prima domanda da porsi e' : "dato che l'INS 38 richiede un certo numero di parametri, dove li va a prelevare in caso di dati insufficienti?". La risposta e' immediata: utilizza i byte successivi a x0...x7, cioe' il Nano 82, la signature ed i byte che la seguono (se necessari). Ricordando che la C1 38 "canonica" presenta questa forma:

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 ecc...

e che l'INS custom e' cosi' costruita:

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 ecc...

risulteranno le seguenti corrispondenze :

P3	d1	d2 d3	d4 d5	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

Quando si costruisce l'INS, conosciamo quali sono i byte che non saranno decriptati, allora possiamo sapere in anticipo quali valori assumeranno (*alcuni o tutti*) i parametri della C1 38. Vediamo come sfruttare questa caratteristica per la generazione di altre risposte.

### Len 0x13 - Generazione multipla

Abbiamo gia' visto qual'e' il contenuto in chiaro delle risposte di lunghezza 13:

C1 36 21 00 LEN

36

86 b0 b1 b2 b3 b4 b5 b6 b7

03/04 82 ...

Agendo su b0...b7 e' possibile variare il crypt della risposta, in maniera da generare una "famiglia" di C1 36 differenti tra loro. I metodi per variare b0...b7 si scoprono esaminando la precedente tabella: o si cambiano i byte della signature (ma questo imporrebbe la ricerca di una nuova INS "custom") oppure si cambiano i byte m0, m1, ecc. che, ricordo, possono essere "in chiaro" (*vedi pag. 31*). Naturalmente si deve tenere conto del BYTE-bug, quindi la variazione dei byte deve essere effettuata in modo da mantenere costante la (somma WORD mod 0x4000). Supponendo di avere a disposizione l'INS 36 di pag. 31...

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]

...possiamo utilizzarla all'interno di uno script per Winexplorer che esegue la generazione multipla delle risposte. Quello che segue e' stato commentato per la comprensione del metodo:

```
' Generatore di INS corte (LEN = 0x13)
' Filtro P3 : (>0F) AND (<90) - Filtro P4 : (=x1) OR (=x9)

' Funzione cnvByte
' Converte un byte da esadecimale a stringa
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

' Funzione cnvWord
' Converte una word (2 byte)
' da esadecimale a stringa (necessita della funzione cnvWord)
Private Function cnvWord(Wdata)
TWdata = (Wdata AND 65535)
cnvWord = cnvByte(TWdata\256) + cnvByte(TWdata)
End Function

' Funzione cnvWordSWP
' converte una word (2 byte)
' da esadecimale a stringa ed scambia i byte (necessita della funzione cnvWord)
Private Function cnvWordSWP(Wdata)
TWdata = (Wdata AND 65535)
cnvWordSWP = cnvByte(TWdata) + cnvByte(TWdata\256)
End Function
```



[illegible]

```

' Ricerca dello status 9000 e risposta a lunghezza voluta
Sc.Print("Building root-command...")
For Root = 0 to 16383
Sc.Write(Ins38)
Sc.Read(01)
' Come si vede, la ricerca della risposta "buona" viene
' fatta sfruttando il BYTE-bug: "Root" e' una word che viene
' fatta variare da 0000 a 3FFF finche' non si ottiene lo status 9000
' Accanto ad essa e' presente una word che viene tenuta fissa a FFFF
' e che risultera' comoda in seguito.
' ATTENZIONE: per come e' costruito il comando, lo script funzionera'
' SOLO se nei dati di partenza P3 = 4x, 6x oppure 8x !!!
' Per usare P3 = 3x, 5x, 7x va messo "cnvWordSWP" al posto di "cnvWord"
Sc.Write(Data+cnvWord(Root)+cnvWord(&HFFFF)+Zeros+NLB_LB)
Sc.Read(02)
myByte = Sc.Getbyte(0)
' Ecco il test sullo status 90xx
If (myByte = &H90) Then
Sc.Write(Ins36)
Sc.Read(2)
LenAnsw = Sc.Getbyte(1)
' Ecco il test per vedere se e' una INS buona (LEN = 13)
If (LenAnsw = LenWanted) Then
Sc.Read(LenWanted+2)
RootFound = 1
Exit For
End If
' Quelle che danno 9015 hanno come primo byte 86 (maggiore di 13)
If (LenAnsw > LenWanted) Then
Sc.Read(LenWanted+1)
Else
Sc.Read(LenWanted+2)
End If
End If
Next
' Se si arriva qui vuol dire che si e' trovato la risposta
' giusta OPPURE si sono esaurite le 16384 possibilita'.
' RootFound indica se si e' trovato il comando
If (RootFound = 1) Then
Sc.Print("FOUND!"+Chr(13)+"Ready to build answers (please wait)"+Chr(13))

```

```

' Faccio un loop su 2 WORD
' e mostro le risposte filtrate
' Tentativi: 65536
For Param = 0 to 65535
Sc.Write(Ins38)
Sc.Read(01)
' Qui si vede come vengono trovate le risposte buone:
' a partire dal valore Root si variano due WORD in maniera
' complementare, mantenendo cosi' costante la (somma WORD modulo 0x4000)
' Anche qui (per P3 = 3x, 5x, 7x) va messo "cnvWordSWP" al posto di "cnvWord"
Sc.Write(Data+cnvWord(Root+Param)+cnvWord(&HFFFF-Param)+Zeros+NLB_LB)
Sc.Read(02)
Sc.Write(Ins36)
Sc.Read(LenWanted+4)
' Qui si prelevano i primi due byte della risposta e si
' valuta se sono dei valori validi per P3 e P4
myP3 = Sc.Getbyte(2)
myP4 = Sc.Getbyte(3)
myP4c= myP4 AND &H0F
' Se P3 P4 sono ok allora mostro la risposta
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
' Se siamo sfortunati... :(
Sc.Print("NOT FOUND! Try with different C1 38!")
End If
' Qui finisce lo script
Wx.Closeport
End Sub

```

Lo script cosi' com'e' e' pronto all'uso, basta sostituire i propri dati la' dove vengono inizializzate le variabili. I vari programmini piu' o meno miracolosi presenti in rete usano un procedimento analogo per generare le INS corte.

### Len 0x1C - Procedura per tentativi

Il discorso si complica per lunghezze maggiori; nel caso della  $LEN = 1C$  e' necessario individuare un'INS corta il cui decrypt dia luogo ad un  $d1 = 00$  oppure  $01$ . Per fare questo ci viene in aiuto (al solito) il BYTE-bug. Sapendo che ad ogni "somma WORD modulo  $0x4000$ " corrisponde un diverso decrypt dell'INS di partenza, si fanno variare i byte in chiaro post-signature fino ad ottenere una  $C1\ 38$  con status  $9000$  ed annessa  $C1\ 36$  con risposta a lunghezza voluta. Dato che sono possibili  $16384$  combinazioni, esistono ragionevoli possibilita' (*ma non la certezza*) di trovare il  $d1$  giusto. Se non lo si trova (*caso sfortunato*) si devono cambiare i dati dell'INS di partenza. Lo script precedente e' valido anche per questo tipo di ricerca, una volta effettuate delle opportune (*piccole*) modifiche:

[illegible]

```

NLB_LB = "00 27"
' Modificato : la LEN e' passata ad 1D
Ins36 = "C1 36 21 91 1D"
' Modificato : aggiunta la visualizzazione della risposta in chiaro
' per capire (dallo pseudo-nano 83/84) se d1 = 00 oppure 01
Ins36p= "C1 36 21 01 1D"
' Modificato : in questo caso vogliamo le risposte lunghe 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
' Modificato : visto che la ricerca puo' impiegare del tempo,
' e' stato inserito per comodita' un indicatore di "STEP"
If ((Root Mod 256) = 0) Then Sc.Print("Step : "+cnvWord(Root)+chr(13))
Sc.Write(Ins38)
Sc.Read(01)
' Per usare P3 = 4x, 6x lasciare cosi com'e'
' Per usare P3 = 3x, 5x, 7x va messo "cnvWordSWP" al posto di "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
' Modificato : in questo caso, prima di partire con la
' generazione delle risposte, mostra l'INS in chiaro
' per capire che valore si e' ottenuto per dl
Sc.Print("FOUND!" + Chr(13))
Sc.Write(Ins38)
Sc.Read(01)
' Per usare P3 = 4x, 6x lasciare cosi com'e'
' Per usare P3 = 3x, 5x, 7x va messo "cnvWordSWP" al posto di "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)
' Per usare P3 = 4x, 6x lasciare cosi com'e'
' Per usare P3 = 3x, 5x, 7x va messo "cnvWordSWP" al posto di "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
' Modificato : il filtro su P3 e' solo "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

```

```

Sc.Print(Chr(13))
End If
Next
Else
Sc.Print("NOT FOUND! Try with different C1 38!")
End If
Wx.Closeport
End Sub

```

Osservazione: per questo genere di ricerca non e' possibile partire da un'INS custom con P3 = 8x : come si puo' vedere dalla tabella si avrebbe in tale condizione un valore fissato per d1, che andrebbe a cadere esattamente sul nano 82, il quale non subisce nessun decrypt e/o unmasking.

#### Len 0x1F/0x24 - Procedura per tentativi "pilotati" (uso della signature)

Questa famiglia di risposte richiede, oltre al giusto valore di **d1**, anche dei parametri **d3 d4** validi:

<b>d1</b> = 03	Esistenza di un evento PPV con ID <b>maggiore o uguale a d2 d3</b>
<b>d1</b> = 04	Esistenza di un record Ex con primo byte <b>uguale a d3</b>
<b>d1</b> = 06	Esistenza di un record con index <b>maggiore o uguale a d2 d3</b>

La condizione si riferisce ai dati relativi al solo provider interessato dal comando (low-nibble di P1).

Queste condizioni impongono la scelta di un'INS custom opportuna per poter generare le risposte lunghe.

Osserviamo la corrispondenza tra byte dell'INS custom e parametri "d2 d3":

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

Per le INS "custom" con P3 compreso fra 0x10 e 0x5F si nota che i parametri dipendono dal (de)crypt del comando, mentre per P3 compreso fra 0x60 e 0x8F sappiamo esattamente quanto essi valgono, perche' vengono prelevati i byte della signature che sono in chiaro. La costruzione delle risposte "lunghe" sara' quindi costituita da due fasi distinte: ricerca del comando "corto" di partenza e ricerca della risposta lunga.

### Len 0x1F/0x24 - Ricerca del comando corto

Si e' detto che serve una risposta con P3 compreso tra 0x60 e 0x8F, in realta' dovremo restringere il campo di ricerca:

- Per P3 = 6x, d2 e' fissato al valore 82, e' possibile solo la variazione di d3 (non utilizzabile per d1 = 03,06)
- Per P3 = 8x, d1 e' fissato al valore 82 (vedi l'[osservazione](#) presente nella precedente pagina)

La conseguenza e' che saranno utili solo INS custom con P3 = 7x. Si procedera' alla ricerca delle risposte corte utilizzando, ad esempio, il primo script delle pagine precedenti con una piccola modifica al "filtro" delle risposte. Al posto di:

```
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
```

si sostituisce :

```
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
```

Di tutte le risposte ottenute andra' selezionata quella "buona". Abbiamo detto che "d2 d3" = "s0 s1", dove s0 ed s1 sono i primi due byte della signature.

Se volessimo, ad esempio, una risposta lunga 0x1F con d1 = 03 ed avessimo sulla card un record Bx relativo all'event-ID **16 00** andrebbero cercati:

**"s0 s1" < 16 00**

Se volessimo, invece, una risposta lunga 0x1F con d1 = 04 ed avessimo sulla card un record Ex di tipo **50** (record presente su tutte le card V7 correttamente attivate) andrebbero cercati:

**s0 qualunque**

**s1 = 50**

Se volessimo, infine, una risposta lunga 0x24 ed avessimo (per il provider indicato da P1) l'ultimo record posizionato all'indice **00 21**, andrebbero cercati:

**"s0 s1" < 00 21**



### Len 0x1F/0x24 - Ricerca della risposta lunga

Una volta trovato il comando corto da usare come "seme" si procede alla ricerca delle INS lunghe utilizzando la procedura per tentativi analoga a quella delle risposte a lunghezza 0x1C. Il secondo script delle pagine precedenti permette (con poche modifiche) di eseguire la ricerca:

[illegible]

```

NLB_LB = "00 27"
' Sostituire i P1 e P2 che interessano ed al posto di LEN
' il valore 20 (per le risposte 0x1F) oppure 25 (per le risposte 0x24)
Ins36 = "C1 36 P1 P2 LEN"
' Sostituire il P1 che interessa ed al posto di LEN
' il valore 20 (per le risposte 0x1F) oppure 25 (per le risposte 0x24)
Ins36p= "C1 36 P1 00 LEN"
' Inserire la lunghezza della risposta voluta
LenWanted = &H1F
' Sostituire &H24 se si vuole quelle a 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 dovra' essere = 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 dovra' essere = 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 dovra' essere = 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

```

### Fase 5c : Utilizzi di una INS "custom" per generare ECM

Le INS 3C hanno la particolarita' di avere il primo ottetto che non subisce SuperEncryption, volendone costruire una "custom" si deve tenere conto di questo fatto. Sapendo che se in un comando ci sono zero dati da decryptare la card non risponde (o risponde con l'ATR), allora si conclude che, per costruire una 3C, bisogna rispettare una LEN minima per la quale in fase di parsing si abbiano almeno 9 bytes *dopo* P3 + suoi dati e *prima* della signature. E' quindi impossibile ottenere risposta da un ECM custom a partire dalle risposte "corte" (LEN = 0x13).

Il metodo per ottenere una 3C "custom" valida e' quello di cercare una risposta all'INS 36 la cui LEN sia almeno 0x1C, in cui P3 e P4 siano ammissibili ed in cui il nibble basso di P2 sia maggiore o uguale a 0x0C (altrimenti status 904A). La costruzione degli ECM custom segue lo stesso procedimento della costruzione degli EMM, l'importante e' ricordarsi che la tabella degli NLB\_LB della C1 3C e' diversa da quella delle C1 38/40.

Nella costruzione di INS 3C mediante risposta "lunga" si ha lo scatenarsi del bug per  $30 \leq P3 \leq 9F$ . La formula giustifica lo stesso comportamento delle 38, ma per valori diversi di P3 a causa del primo ottetto non criptato. A questo punto si verifica come debba valere la seguente:

**con le INS "lunghe", se il bug scatta, il 90 00 non permane (status instabile)**

Questo significa che inviando ripetutamente la stessa INS, senza resettare e senza variazione di byte si ottengono status diversi! Questo significa che, ad ogni invio, cambia qualche condizione che provoca un diverso *unmasking*.

#### Osservazione 1:

L'instabilita' dello status si puo' conseguire anche con le C1 38/40, sempre utilizzando le risposte lunghe e con valori di P3 tali da ottenere meno di 8 byte di dati da decryptare.

#### Osservazione 2:

Il *bug 9600* ha 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"...

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 00
00 00 00 00 00 63
```

## Welcome to Las Vegas - Probabilità di avere un parsing corretto (citazione & traduzione)

Dopo le voci insistenti di "programmini miracolosi" ho deciso di capire se fossero presenti dei bug non ancora scoperti dalla gente *comune* o se, effettivamente, tutte le informazioni fossero di dominio pubblico. Mi sono veramente sorpreso per il risultato ottenuto: le condizioni di parsing corretto pongono una restrizione molto forte sulla struttura delle INS generate in maniera casuale:

Facciamo l'ipotesi che le proprietà statistiche dei byte in chiaro e di quelli criptati siano le stesse (cioè che la SuperEncryption goda di sufficiente proprietà di dispersione).

Prendendo un comando lungo **n**, è interessante cercare di scoprire tutte le combinazioni di nanocomandi SEKA che possono generarsi nel comando stesso, in modo tale da avere un parsing corretto e quindi l'esecuzione da parte della card. Tutte le possibilità possono essere calcolate applicando ricorsivamente una "forma grammaticale" nella quale il simbolo **n** è espresso in **n → (n-1) & 0**, o in altri termini, un nano **Mx** può essere sostituito dal nano **(M-1)x** più il nano **0w**.

Applicando quanto detto a un nano qualunque "**Nx yy ... yy**" (nano + dati) e tenendo conto della non espandibilità del nano **0w**, possiamo costruire tutte le possibilità (NOTA: nella trattazione non saranno considerati i Nanocomandi Dx, Ex, Fx, ma con poche considerazioni aggiuntive possono comunque rientrare nella procedura).

Si definisce "**Type**" questa struttura, di conseguenza una INS **Type 110** conterrà i seguenti dati:

**1x mm 1y nn 0z**

Invece, per una INS **Type 100**, si ha:

**1x mm 0y 0z**

Consideriamo quest'ultima struttura, quante INS possono essere costruite con essa? Avendo 3 byte con un nibble fisso ed un byte a valori qualunque, risulterà:

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

Dove:

<b>m</b>	è il numero di Nanocomandi della <b>Type</b> ( 3 in questo caso )
<b>n</b>	è la lunghezza dell'INS
<b>(n - m)</b>	è il numero di byte a valori qualunque

L'espressione precedente non rende conto di tutte le possibili **permutazioni** dei nanocomandi. Questa proprietà di permutazione si definisce "**molteplicità**". Dovremo quindi calcolare le permutazioni con ripetizioni di **k** elementi (cioè i nanocomandi), avendone **ki** per ogni tipo:

$$PERM(k, (k_1, k_2, \dots, k_n)) = k! / ((k_1!) * (k_2!) * \dots * (k_n!))$$

Nel caso trattato (**Type 110**) avremo:

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

Infatti i possibili risultati della permutazione sono "110", "101", "011". Con tutto quanto enunciato ora possiamo calcolare la probabilità di ogni **Type** su una INS generata casualmente e, conseguentemente, la probabilità di avere un parsing corretto. Nella seguente tabella si potranno osservare le probabilità per dati dell'INS di lunghezza variabile da 2 a 9 byte.

**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	16777216		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	2,19902E+12	0,78%	9,26%
31	1,09951E+12	2	2,19902E+12	0,78%	9,26%
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%	61,58%
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%	

Diverse cose possono essere estrapolate dai numeri rappresentati in tabella: per prima cosa, si vede che la probabilità di avere un parsing corretto varia da 6.46% per INS con lunghezza di 2 byte al 10.15% per INS con lunghezza di 10 byte, un dato riscontrabile anche sperimentalmente. Un'altra importante proprietà che si ricava è la seguente:

*Per una INS di lunghezza (N+1) la probabilità di avere il Nano Nx è molto maggiore di ogni altra configurazione a parsing corretto.*

In conclusione, se assumiamo che la INS con Type che includono "1" "0/1" "0y" sono le più pericolose, ed escludendo la possibilità di generare cancellazioni multiple di chiavi è possibile stimare la proporzione tra la probabilità di produrre INS dagli effetti **“positivi”** e probabilità di produrre INS dagli effetti **“negativi”**.

*Per esempio, un 80 xx xx xx... è circa 130 volte più probabile della cancellazione di una chiave ...*

E' finalmente "rivelato" il mistero della generazione dei nano e questo porta ad un grosso passo avanti : possiamo usare svariati nanocomandi che potranno aiutarci a capire meglio il funzionamento della card. La situazione è analoga a quella del passaggio da un modello "relativistico" ad un più controllabile modello "classico" dove il determinismo ci permette di prevedere il risultato delle nostre azioni.



Sono stati definiti numerosi modelli riguardo al funzionamento della SuperEncryption; dall'incrocio e la fusione di queste teorie nasce il modello che seguirà in queste pagine, ma prima è necessario analizzare una serie di prove sperimentali. A tal proposito, si considerino le risposte lunghe 0x24 ed associamo un *nome* ai byte criptati:

```
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

ed anche ai corrispettivi byte in chiaro...

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

Costruiamo una tabella "byte in chiaro" vs "byte criptati", dove riportare la variazione dei byte criptati in funzione della variazione dei byte in chiaro (\* = variato, . = invariato):

[illegible]

Dall'analisi per righe si osserva l'organizzazione ad ottetti dei byte *in chiaro*:

- (1) p00---p07 : il contenuto di questi 8 byte condiziona c00---c07 e c13---c1A
- (2) p08---p0F : il contenuto di questi 8 byte condiziona c00---c0F e c13---c1A
- (3) p10---p17 : il contenuto di questi 8 byte condiziona c00---c12 e c13---c1A

Nei comandi lunghi 0x24 non si riescono a variare gli ultimi byte, ma facendo prove analoghe su comandi di lunghezza diversa si puo' concludere che:

- (4) p18---p1A : il contenuto di questi 3 byte condiziona c00---c12 e c13---c1A

Prima osservazione: gli ultimi 8 byte cambiano variando un qualsiasi byte.

Teniamo da parte questa considerazione, che sara' utile piu' avanti, ed andiamo a considerare i punti 1 & 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	* * * * * * * *	* * * * * * * *	. . .	* * * * * * * *
(...)				

Analizzando per colonne, sembra presentarsi una suddivisione in ottetti anche per i byte criptati. Si puo' ipotizzare questa sequenza logica di operazioni: dato che il contenuto di p08---p0F influenza c00---c0F, mentre il contenuto di p00---p07 influenza c00---c07 ma non c08---c0F, allora...

- il crypt(p08---p0F) avviene PRIMA del crypt(p00---p07)
- il crypt(p00---p07) dipende dal crypt(p08---p10)

*NOTA BENE: Per adesso non si ipotizza quali siano gli operandi del crypt. La scrittura "crypt( )" va interpretata come una generica "funzione di cifratura dati".*

Osserviamo il punto 3: in questo caso tutti i byte cambiano. Dal processo descritto precedentemente si potrebbe, *per induzione*, affermare :

- il crypt(p10---p17) avviene PRIMA del crypt(p08---p0F)
- il crypt(p08---p0F) dipende dal crypt(p10---p17)

Pero' tra il punto 2 ed il punto 3 non c'e' un ottetto criptato di differenza, bensì solamente 3 byte...

	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 osservi che l'ottetto p10---p17 e' parzialmente sovrapposto all'ottetto p13---p1A (gli otto byte pre-signature) e quest'ultimo e' variabile in ogni condizione (*vedi tabella*), ma abbiamo un dato di fatto: c10-c12 pur apparendo "spaiati", non sono rimasti in chiaro, segno che il processo di crypt si occupa anche di loro. Formuliamo 3 ipotesi :

Ipotesi 1 : Potrebbe essere `crypt(p10 p11 p12 00 00 00 00 00)`

Ipotesi 2 : Potrebbe essere `crypt(p10 p11 p12 p13 p14 p15 p16 p17)`

Ipotesi 3 : Potrebbe essere `crypt(p10 p11 p12 x13 x14 x15 x16 x17)`

*L'ultima scrittura significa che il crypt utilizza per i byte 13-17, quelli risultanti dal crypt(p13---p1A)*

Abbiamo un indizio, cioè che c10-c12 variano se varia uno qualunque dei byte p10-p17 e quindi possiamo eliminare la prima ipotesi. Anche la terza ipotesi e' da escludere: con tale metodo i byte c10-c12 varierebbero modificando un qualunque byte in chiaro. Allora, le seguenti assunzioni dovrebbero essere vere:

- il `crypt(p10---p17)` avviene PRIMA del `crypt(p08---p0F)`

- il `crypt(p08---p0F)` dipende dal `crypt(p10---p17)`

Inoltre si osserva il seguente fatto :

- il crypt degli 8 byte pre-signature avviene DOPO il `crypt(p10---p17)`

E' ragionevole pensare che avvenga come ultimo processo. Analizzando il punto 4, però, si deduce che i byte **p18-p1A** devono entrare in gioco anche nella fase iniziale del processo.

Ipotesi A : I byte restano in chiaro e `crypt(p10---p17)` dipende da essi

Ipotesi B : I byte subiscono un `crypt(...)` e `crypt(p10---p17)` dipende da essi

Ipotesi C : I byte restano in chiaro (prima della fase finale), ma `crypt(p10---p17)` dipende dal `crypt(p18---p1A)`

Osservazione: quando si parla di crypt di ottetti incompleti, si suppone di "aggiungere" i byte mancanti attraverso dei valori prefissati. Questo processo si chiama *padding*, di solito viene adottato il padding con *zeri*, ma non e' l'unico metodo incontrato nella teoria degli algoritmi di cifratura. Ad esempio, nell'algoritmo IRDETO 1 si legge questo :

#### IRDETO 1 (citazione)

La preparazione del pacchetto da (de)criptare si ottiene tramite la suddivisione in blocchi da 8 byte ciascuno.

Blocco 1: byte 0..7

Blocco 2: byte 8..15

... e cosi' via fino all'ultimo blocco che, ovviamente non dovra' obbligatoriamente terminare con ridondanza di 8 byte (cioe' puo' essere piu' corto di 8 byte) e nel caso lo fosse aggiungiamo i bytes esadecimali **0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67**; tanti quanti ne sono necessari per arrivare ad 8 byte anche con l'ultimo pacchetto.

E' possibile dimostrare (e sara' fatto *a posteriori*, cioe' una volta descritto il modello) la validita' delle seguenti affermazioni :

- p18 p19 p1A restano *in chiaro*
- il crypt(p10---p17) dipende dal crypt(p18 p19 p1A)

Riassumendo il tutto:

- p18 p19 p1A restano in chiaro
- il crypt(p10---p17) dipende dal crypt(p18 p19 p1A)
- il crypt(p10---p17) avviene PRIMA del crypt(p08---p0F)
- il crypt(p08---p0F) dipende dal crypt(p10---p17)
- il crypt(p08---p0F) avviene PRIMA del crypt(p00---p07)
- il crypt(p00---p07) dipende dal crypt(p08---p10)
- il processo si conclude col crypt(c13---c17 p13---p1A)

Esse non sono sufficienti a descrivere il comportamento della SuperEncryption, vanno aggiunte ulteriori considerazioni:

#### SuperEncryption (citazione)

Esistono combinazioni di byte in chiaro danno il primo ottetto cambiato, ma non gli 8 byte pre-signature.

Esempio 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

Esempio 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

Esempio 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 (citazione)

Numerando da 1 ad N gli ottetti *interi*, si definiscono "*esterni*" i byte contenuti all'interno degli ottetti da 1 ad N-1

(...)

Quando si variano soltanto i byte *esterni* si ha che il numero di risultati del decrypt dell'ultimo ottetto sia limitato a sole 16384 combinazioni.

L'ultima citazione e' illuminante : 16384 (0x4000 in esadecimale) e' proprio la quantita' usata quando si va ad eseguire il modulo all'interno della "somma delle WORD", con una rapida verifica si trova il legame tra le risposte per ciascuna delle coppie d'esempio prima riportate.

(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

Il crypt dell'ultimo ottetto sembra dipendere dalla "somma WORD mod 0x4000" del primo ottetto

Utilizzando risposte piu' lunghe (LEN = 0x24) si possono aggiungere altre considerazioni :

```
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
```

Queste due risposte hanno medesimo ottetto pre-signature e...

```
(E71F + 73BA + 5AF5 + AD81 + E1C5 + 7ADE + CA5C + 3B77) MOD 4000 = 05C5
```

```
(B5F1 + 65F5 + E0B8 + 66B1 + C4C0 + 9578 + F01D + 9821) MOD 4000 = 05C5
```

...stessa "somma WORD" estesa all' intero corpo dell'INS (ultimi otto byte esclusi, naturalmente).

**Il crypt dell'ottetto pre-signature dipende dalla "somma WORD mod 0x4000" dei restanti dati**

Focalizzando l'attenzione sul primo e secondo ottetto di una risposta lunga 0x24 si osserva :

```
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
```

```
(47 55 + 18 7C + 8F 14 + 19 47) MOD 4000 = 082C
```

```
(DE 8E + F0 7F + CC E7 + 6C 38) MOD 4000 = 082C
```

**Il crypt del primo ottetto dipende dalla "somma WORD mod 0x4000" del secondo ottetto**

La regola puo' essere estesa ad un qualunque ottetto del comando criptato, esclusi gli 8 byte pre-signature:

**Il crypt di un ottetto dipende dalla "somma WORD mod 0x4000" dell'ottetto alla sua destra**

Da notare che il primo ottetto da criptare/decriptare, non avendo un ottetto alla sua destra, utilizzerà una particolare "somma WORD" generata in fase di inizializzazione dell'algoritmo.

A questo punto e' possibile formulare una possibile teoria di funzionamento, ma prima vanno definite le seguenti operazioni: *cypher* e *masking*

Cypher : e' il classico criptaggio SEKA1, i parametri da passare in ingresso sono l'ottetto da criptare e la chiave utilizzata (indicata dal low-nibble di P2 e dal bit 4 di P1). E' inoltre possibile selezionare le tabelle hash per il cypher (*tramite i bit 6,5 di P1*).

Masking : e' un processo di criptaggio ad algoritmo sconosciuto (*da alcuni test risulterebbe un'operazione di XOR -- con valori tabellati -- associata ad una permutazione, ma non si ha la certezza di cio'...*); i parametri da passare in ingresso sono l'ottetto da mascherare e la chiave utilizzata. Quest'ultima e' formata da 14 bit ed e' costruita a partire da un'ottetto di dati *diverso* (almeno nell'implementazione teorica...poi ci sono i bug 😊) da quello da mascherare. *La chiave si costruisce facendo la somma delle WORD mod 0x4000.*

NOTA: Per ragioni "storiche" e' stata mantenuta la dizione *masking*, sebbene sia piu' giusto parlare di *chaining*, alla luce degli ultimi aggiornamenti del modello-SuperEncryption.

#### *SuperEncryption - Un possibile modello (ultimo aggiornamento : 21/05/03)*

Il processo si puo' suddividere in tre fasi:

- *Inizializzazione*
- *Encryption del corpo dell'INS*
- *Encryption della coda (ottetto pre-signature)*

Si prende il corpo dell'INS fino al nano 82 escluso e lo si divide in ottetti, partendo da sinistra verso destra. E' possibile avere dei byte che non formano un ottetto completo, essi saranno definiti *byte spaiati*. Una volta effettuata la suddivisione, si numerino gli ottetti in senso crescente, partendo da quello piu' a destra procedendo verso sinistra (la numerazione parte da 0). Definiamo alcune quantita' di comodo :

**(Ai)** : ottetto i-esimo in chiaro

**(Mi)** : risultato del masking dell'ottetto i-esimo

**(Si)** : somma WORD mod 0x4000 da usare per il masking

**(Ci)** : risultato del cypher dell'ottetto i-esimo

**'key'** : la chiave usata per il cypher (la normale chiave del SEKA)

**(LB)** : gli otto byte pre-signature

**(BS)** : i byte spaiati

**(A\_init)** : ottetto generato in fase di inizializzazione

**(M\_init)** : risultato del masking di (A\_init)

**(C\_init)** : risultato del cypher di (M\_init)

**(MF)** : risultato del masking di (LB)

**(CF)** : risultato del cypher di (MF)

### Fase 1 : inizializzazione

Si verifica se sono presenti *byte spaiati* :

- Se si'

```
A_init = padding(BS)
M_init = mask(A_init, 0000)
C_init = cypher(M_init, key)
```

- Se no

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

Si ipotizza un padding fatto con *zeri*. **L'ottetto (C\_init) sara' usato per il calcolo di (S0)**, ma NON sostituisce nessuna parte del testo da criptare : i byte spaiati eventualmente presenti rimangono in chiaro.

### Fase 2: encryption del corpo dell'INS

Questa fase avviene a carico degli ottetti interi ed e' costituita dal mascheramento e dalla cifratura con l'algoritmo SEKA1 (*mask & cypher*). Il mascheramento dipende dall'ottetto adiacente (*alla destra*), attraverso l'ormai nota "somma WORD mod 0x4000". Si realizza cosi' un meccanismo di cifratura a catena (*Cipher Block Chaining*).

```
Si = sum(Mi-1)
Mi = mask(Ai, Si)
Ci = cypher(Mi, key)
```

Per  $i=0$  vale la somma fatta  
sull'ottetto (C\_init)

Per implementare questo processo e' necessario eseguire "*prima masking e poi cypher*" ottetto per ottetto, cioe' non si possono separare i due processi, come nella teoria del "*prima cypher e poi masking*" proposta in passato.

### Fase 3 : encryption della coda

```
SF = sum(CYPHERTEXT-8)
MF = mask(LB, SF)
CF = cypher(MF, key)
```

La somma SF utilizza tutti i byte del corpo dell'INS (a questo punto gia' criptati) ad esclusione degli ultimi 8. L'implementazione nella card di questo processo e' affetta da bug (si puo' avere un **pointer underflow**).

Il modello e' applicabile con successo su un qualsiasi numero di byte superiore o uguale a 8 (e dunque anche su ottetti singoli). Se i dati da (de)criptare sono esattamente 8 la "fase 2" non viene eseguita (e la somma SF vale 0000).



### SuperEncryption - Verifiche sperimentali

Una volta identificato il modello e' possibile analizzare alcuni test per la sua validazione (rivisitazione di post presenti sui Forum-Sat)

#### Prova numero 1 - L'elaborazione di (LB)

Si costruisca una C1 38 (a partire da una risposta "corta") in grado di generare una risposta lunga 0x24, ad esempio la seguente :

```
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 00 09
```

Tramite essa, si ricavano 256 risposte lunghe 0x24 e 256 risposte "corte" : si tratta di far variare il byte **yy** ed inviare due coppie di C1 38/36 con LEN della risposta differente.

```
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

... e ...

```
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 forma in chiaro delle risposte e' questa (la prima e' quella "corta", la seconda e' quella "lunga"):

```
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
```

Naturalmente card diverse possono dare risposte diverse, a seconda della configurazione dei record, ma ai fini del risultato non vi sono differenze.

*Osservazione : per come sono state costruite le risposte, il primo ottetto in chiaro sara' lo stesso.per tutte.*

Dalle 256 + 256 risposte generate si estragga una coppia *lunga* + *corta* tali da avere i primi due byte criptati uguali. In generale, le due risposte avranno un valore differente per **yy**

*Risposta criptata*

**23 1E** EA 08 17 F0 61 10  
ED 4E 82 E7 91 26 7E 62  
B0 70 F8

**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

*Risposta in chiaro*

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

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

In questo caso abbiamo yy = 7B per la lunga e yy = 67 per la corta. Adesso e' necessario spiegare che cosa implica avere i primi due byte uguali. Si pensi alla suddivisione in ottetti delle INS ed al modo di operare della SuperEncryption : le due risposte hanno il primo ottetto *in chiaro* uguale, ma (in generale) crypt differente, per effetto del *masking*. Oltretutto, nella risposta corta, gli ultimi sei byte del primo ottetto fanno anche parte dei byte pre-signature elaborati durante la "fase 3" della SuperEncryption; restano fuori soltanto i primi due byte e proprio essi stiamo osservando : nel caso in cui il masking applicato al primo ottetto porgesse lo stesso risultato per entrambi i comandi, allora sarebbero uguali proprio i primi due byte (a parita' di cypher).

*Per avere lo stesso masking dovranno essere uguali le "somme WORD" usate per il mascheramento.*

A questo punto, se variassimo il primo ottetto, otterremmo dei crypt diversi dai precedenti, ma sempre accomunati dai primi due byte. Questa proprieta' servira' piu' avanti.

Si costruisca adesso una risposta lunga che abbia il terzo byte uguale ad **86** (lo si fa variando i byte compresi tra la signature ed yy, con yy = 7B).

Si generi, inoltre, una risposta corta con yy = 67 e stesso ottetto della "lunga":

### Risposta criptata

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

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

### Risposta in chiaro

86 E0 5C 00 45 6B 00 00  
67 03 82 + SIGNATURE

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

Si puo' notare l'uguaglianza dei primi due byte, in accordo con la proprieta' descritta precedentemente.

### A cosa serve il byte fissato al valore 86 ?

Analizziamo il processo della SE applicato alla risposta "corta". Si avranno i seguenti passi:

- masking (86 E0 5C 00 45 6B 00 00) e cypher del risultato
- elaborazione dell'ottetto pre-signature, chiamiamolo (LB) per comodita'.

Quello che si vuole dimostrare e' che l'elaborazione di (LB) consiste in un mask & cypher. Si procede nella dimostrazione analizzando la risposta "lunga", in particolare il primo ottetto, di cui conosciamo sia il valore in chiaro, sia il valore dopo il mask & cypher :

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

"70 56" coincidono (per costruzione) con i primi due byte della risposta corta. Gli altri sono in generale differenti per effetto dell'elaborazione di (LB). Si suppone che il processo sia il seguente :

### dati in chiaro

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

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

(70 56 86 74 79 F9 D6 DF) 67 03 82 + SIG. -> dopo il primo mask & cypher (vedi ipotesi)

(70 56 05 B6 20 15 DF 76) FA B4 82 + SIG. -> dopo l'elaborazione dell'ottetto pre-signature

Ipotesi : data l'uguaglianza dei primi due byte con quelli della lunga, si puo' ragionevolmente supporre che il risultato del mask & cypher di 86 E0 5C 00 45 6B 00 00 porga lo stesso risultato della risposta lunga.

*dati criptati*

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

Nella risposta lunga il masking ha usato la seguente somma WORD :

**(46 FB + CD 8A + 11 36 + 25 A9) mod 4000 = 0B64**

Questo valore e' *diverso* dalla somma dei "byte spaiati" della risposta corta...

**6703 mod 4000 = 2703**

Si ha la conferma del non utilizzo, da parte del masking del primo ottetto, dei byte spaiati in chiaro.

Riprendiamo il processo-SE applicato alla "corta", questa volta evidenziando l'ottetto pre-signa:

86 E0 (5C 00 45 6B 00 00 67 03) 82 + SIG. -> in chiaro  
70 56 (86 74 79 F9 D6 DF 67 03) 82 + SIG. -> dopo il primo mask & cypher (*ipotesi*)  
70 56 (05 B6 20 15 DF 76 FA B4) 82 + SIG. -> dopo l'elaborazione dell'ottetto pre-signature

Dall'elaborazione di **86 74 79 F9 D6 DF 67 03** si arriva all'ottetto **05 B6 20 15 DF 76 FA B4**.

Ecco dove serve la presenza del byte **86** : e' possibile costruire una risposta il cui primo ottetto in chiaro sia proprio uguale a **86 74 79 F9 D6 DF 67 03** (il primo byte corrisponderebbe allo pseudo-nano della risposta al C1 36) :

C1 38 21 90 LL

71 D1 D6 56 6A F2 72 5B	(I byte evidenziati sono successivi
DA F3 82 A4 A2 23 7F 48	alla signature e precedenti i 90 byte
A0 74 79 F9 D6 DF 67 03 etc...	in SSE/envelope)

C1 36 21 90 2C (Risposta lunga con d1=03, due PPV record)

Siamo quasi giunti al termine della verifica; facendo variare i byte dei record dumpati si riesce ad arrivare ad una risposta di questo tipo :

**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

- Il primo ottetto e' il mask & cypher di 86 74 79 F9 D6 DF 67 03
- Nella risposta corta, invece, era il risultato dell'elaborazione di (LB), costituito dagli stessi byte del punto precedente.

Risultato : *L' elaborazione di (LB) e' un mask & cypher*

Possiamo ricavare anche la somma WORD associata al mascheramento :

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

Consideriamo l'INS corta...

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

I primi due byte sono proprio "**70 56**" e sono i soli che, in questo caso, prendono parte alla somma WORD.

#### *Prova numero 2 - Possibile implementazione del decrypt-SE*

Generando la "successione delle delta" per le seguenti INS otteniamo il medesimo risultato (a parte il disallineamento, vedi pag. 41 per il significato delle "tabelle delle 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

Analizzando le tabelle si riesce ad identificare di quanto sono disallineate (vedi pag. 42) e si trova, ad esempio, la seguente corrispondenza:

$$(y1\ y2 - x1\ x2) \bmod 4000 = 0F9E$$

La quantita' "**0F 9E**" non è casuale, ma coincide con la differenza tra "somme WORD" dei seguenti byte:

<b>D8</b> 82 55 6C E6 D4 AC B2 3C 50	prima INS
<b>56</b> 82 6A 8F 4E 36 F3 4D AB 92	seconda INS

"**D8**" e "**56**" sono byte in SuperEncryption: la "somma WORD" sta considerando i byte *prima* della loro elaborazione, allora :

**indipendentemente dall'ordine in cui sono eseguiti (de)cypher ed (un)mask di un ottetto, la "somma WORD" e' eseguita prima di entrambi i processi**

**NOTA:**

Se i dati da (de)criptare sono esattamente 8 la "fase 2" non viene eseguita (e la somma SF vale 0000).

Ecco i risultati di alcune prove con le C1 38 "custom" (tempi medi su campioni significativi di uguale numerosita') :

```
(0x1A byte in SE -> 3 ottetti + 2 byte) : 534878 cc
(0x19 byte in SE -> 3 ottetti + 1 byte) : 532580 cc
(0x18 byte in SE -> 3 ottetti + 0 byte) : 487071 cc
(0x17 byte in SE -> 2 ottetti + 7 byte) : 489213 cc
(0x11 byte in SE -> 2 ottetti + 1 byte) : 474492 cc
(0x10 byte in SE -> 2 ottetti + 0 byte) : 424974 cc
(0x0F byte in SE -> 1 ottetto + 7 byte) : 429111 cc
(0x0E byte in SE -> 1 ottetto + 6 byte) : 431064 cc
(0x0D byte in SE -> 1 ottetto + 5 byte) : 428460 cc
(0x08 byte in SE -> 1 ottetto + 0 byte) : 340008 cc
(0x07 byte in SE -> esecuzione buggata) : 341730 cc
(0x06 byte in SE -> esecuzione buggata) : 341668 cc
(0x03 byte in SE -> esecuzione buggata) : 343557 cc
```

La variazione del numero di byte in SE e' possibile attraverso l'invio di INS uguale lunghezza, ma parametro P3 differente. Nell'esecuzione di ciascun comando entrano in gioco molti fattori. La tendenza di comportamento e' di aumentare il tempo necessario alla risposta, in funzione della lunghezza del comando. Proviamo a scindere questi tempi in piu' parti :

- a- Una parte costante
- b- Una parte che aumenta all'aumentare del numero di byte (singoli) processati
- c- Una parte che aumenta all'aumentare del numero di OTTETTI processati
- d- Una parte "random" (in cui posso includere anche il pre-parsing, se si effettuano prove uguali con un numero sufficiente di INS differenti).

NOTA: Le prove sono eseguite a parita' di key e tabelle hash, quindi i tempi di accesso a tali dati sono mantenuti costanti (key diverse → diversa posizione in EEPROM → diverso tempo necessario per prelevarle).

Nella parte "c" e' compreso il decypher & unmask; si puo' facilmente separare il "decypher" dal resto (anche dall'unmasking), ottenendo un tempo di circa 39000 cicli di clock per ogni ottetto cifrato.

Dai casi presi in considerazione si estrapolano i seguenti risultati :

Byte in SE	Numero (de)cipher
0x19 0x1A ...	5
0x11 ... 0x18	4
0x09 ... 0x10	3
0x01 ... 0x08	1

Infatti, nel modello della SE erano presenti :

- Un *cypher* in fase di inizializzazione
- "**N**" *cypher* nella seconda fase, con N uguale al numero di ottetti interi
- Un *cypher* nel processo di elaborazione dell'ottetto pre-signature

E' importante la valutazione del comportamento in presenza di soli 8 byte da criptare/decriptare; il processo prende una via leggermente diversa, eseguendo UN solo *cypher/decypher* : la sola fase eseguita e' la terza. In questo caso particolare la somma :

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

non sara' applicata ad alcun byte, assumendo il valore 0000. Si giustificerebbe anche il comportamento in presenza di un numero di byte da decriptare inferiore ad 8, probabilmente la routine e' stata implementata supponendo implicitamente che i dati da mettere in chiaro fossero *almeno* 8, non prevedendo che il puntatore per la somma potesse assumere un valore "negativo" (un po' come accadeva con *zzbug*, per intendersi). In queste condizioni la somma WORD interesserebbe un certo numero di byte posti oltre i dati da decriptare.

Non a caso, nel *byte-bug* si e' osservato come la somma WORD vada a dipendere dalla signature, dai byte in chiaro e dai byte in (de)envelope, confermando tale *sforamento*. Il *reset-bug* si puo' giustificare col fatto che la somma possa proseguire anche nell'area di RAM che segue il command buffer, andando ad interessare una zona dal contenuto non predicibile (si pensi alla ROM della 6.0: durante la fase di startup ci sono pagine di RAM riempite con valori casuali, per verificare il funzionamento del "random generator"). Anche *'l'instabilita' dello status per INS lunghe* e' giustificabile: la somma delle WORD prosegue finche' l'indice non assume un preciso valore (*zero?*) : la RAM "letta" e' in quantita' fissa, ma la posizione da cui iniziare la lettura e' imposta dalla posizione del primo byte da decriptare. Nelle INS lunghe, il primo byte si trova in posizione piu' avanzata, di conseguenza la somma andra' a prelevare il contenuto di indirizzi di memoria posizionati piu' "avanti" in RAM. Se cosi' facendo si coinvolgesse una regione adibita a contenere dei registri e/o buffer si avrebbe una possibile variazione di comportamento ad ogni invio del comando.

Si puo' agevolmente ricavare di quanti byte "sfora" la somma WORD, in funzione della LEN del comando e del valore di P3. Resta da spiegare il fenomeno che si ha in presenza di zero byte da decryptare, cioe' il blocco della card (oppure l'invio dell'ATR).

Forse si va ad oltrepassare la RAM, cadendo in un area "protetta" o non mappata, causando l'entrata in funzione di un dispositivo di protezione della card (*memory management unit*, era presente nelle 3.1/4.0) oppure si vanno ad alterare delle parti di RAM destinate ad altro uso (*stack?*) oppure e' proprio l'esecuzione del firmware a prendere un percorso inatteso, innescando qualcosa di analogo a zzbug.

Esistono molte strade da percorrere per capire meglio questo bug, soprattutto per prevedere "a priori" quale sara' il risultato del decrypt affetto da bug. Si osservi, ad esempio, questo :

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 byte in envelope

Trattandosi di una C1 3C, i primi 8 byte (in verde) restano in chiaro, restano soltanto i byte in azzurro che, essendo meno di 8, saranno elaborati in maniera buggata :

decypher & unmask ( 00 80 xx xx xx xx xx xx ) = x7 x6 x5 x4 x3 x2 x1 x0

Il risultato andra' a sovrascrivere in ogni caso (*altra conseguenza del bug !*) gli 8 byte pre-signature, modificando anche i byte che precedono quelli da mettere in chiaro :

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 byte in envelope

Risultato : il primo ottetto, che nella C1 3C dovrebbe rimanere inalterato, e' invece parzialmente modificato, con relative conseguenze sul parsing dei nanocomandi.

Nel caso delle INS 38/40, invece, il risultato andrebbe ad influenzare i dati di P3, la cui sovrascrittura non provoca nessun particolare effetto.

C1 40 21 B0 76

65 31 D3 B2 97 A5 D2 86 81 DE 82 + signature + 90 byte in envelope

C1 40 21 B0 76

65 31 x7 x6 x5 x4 x3 x2 x1 x0 82 + signature + 90 byte in envelope

*Questo fenomeno meriterebbe ulteriori studi...*



### SuperEncryption - La fase di masking

Il primo tentativo di interpretazione del processo di mascheramento ha utilizzato la cosiddetta catena di istruzioni **3C/3A/04/02** :

- Si costruisce una C1 3C/Nano D1 con CW note
- Si leggono i dati in chiaro DW (tramite C1 3A)
- Si prendono le (DW) separatamente e si criptano con l'INS 04
- Si legge il risultato con l'INS 02

Lo scopo della prova e' di arrivare a dimostrare che, nelle INS 3C, il decrypt delle CW avviene tramite unmasking seguito da decypher. Si verifichera' inoltre la sua diversita' dal decrypt-SE.

*Prima domanda* : INS 3C/Nano D1 utilizza l'unmasking?

*Seconda domanda* : Se INS 3C fa anche unmask, tratta gli ottetti singolarmente oppure tratta le CW come se fossero un testo in SE lungo 16 byte? (puo' sembrare la stessa cosa ma non lo e', basta pensare a come agirebbero unmask & decypher nei due casi)

*Terza domanda* : INS 04 utilizza il masking ?

Si parte da alcuni dati sperimentali :

```
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
C1 3A : C0 3C 43 BB CD 56 D7 E3 52 6D 95 05 70 4C 09 AF
C1 04 : DD F8 3F 67 E6 C6 EC 21 37 CA 89 B8 E0 C6 97 A6
```

*(Questa prova usa gli stessi dati della '2' ma chiave diversa)*

```
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
```

Esaminando le CW (INS 3C) e le DW (INS 3A) si osserva che cambiando i valori di un ottetto criptato non cambiano i valori del decrypt dell'altro ottetto.

Questo esclude che i 16 byte siano trattati come se fossero un unico testo lungo 16 byte, rispondendo alla [seconda domanda](#) e facendo assumere significativita' alle prove con le INS 02/04 applicate alle singole DW.

Si osserva inoltre che tra CW e DW omologhe non sembra esserci alcuna correlazione : e' un risultato atteso, perche' e' uno degli effetti del de-cypher.

Non possiamo ancora dire nulla sulla presenza o meno dell'unmask nel processo della 3C. Quindi, per la [prima domanda](#) si hanno tre possibili risposte. Ciascun ottetto:

- X1- Potrebbe subire solo de-cypher
- X2- Potrebbe subire unmask e poi de-cypher
- X3- Potrebbe subire de-cypher e poi unmask

Per quanto riguarda la [terza domanda](#), si puo', per simmetria, fare un discorso analogo esaminando le DW (INS 3A) ed il loro crypt con le INS 04/02. Le possibili risposte saranno:

- Z1- Potrebbe subire solo cypher
- Z2- Potrebbe subire mask e poi cypher
- Z3- Potrebbe subire cypher e poi mask

Esaminando una qualsiasi CW (INS 3C) e il suo crypt con INS 04/02 si osserva una chiara correlazione tra i due ottetti, pur non essendo uguali. Cio' significa che tra l'uno e l'altro c'e' di mezzo un'elaborazione non troppo complessa.

[Chi fa uso di questa elaborazione? La 3C oppure la 04?](#)

Si possono immaginare due possibili comportamenti :

- a- INS 3C fa unmask & INS 04 non fa mask
- b- INS 04 fa mask & INS 3C non fa unmask

Pensando alle precedenti ipotesi (X1...X3 e Z1...Z3) unite a quest'ultima considerazione, si conclude che non tutte le configurazioni "**Xi**" - "**Yj**" sono ammissibili :

- se X1 e' valida allora si escludono X2, X3, Z1
- se Z1 e' valida allora si escludono Z2, Z3, X1

Esaminiamo adesso le CW/DW "1" e "3": i risultati delle INS 04/02 sono uguali partendo da DW diverse con chiavi diverse. Ragioniamo per assurdo: se fosse vera la coppia "X3" - "Z1" si avrebbe il seguente processo (catena 3C/3A/04/02):

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

Con l'*unmasking* a cavallo tra de-cypher & cypher non si potrebbe pervenire allo stesso risultato finale. Di conseguenza :

- se Z1 e' valida allora si escludono Z2, Z3, X1, X3

Analogo discorso se si considera la coppia "X1" - "Z2" :

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

Con il *masking* a cavallo tra de-cypher e cypher non si potrebbe pervenire (nelle prove in esame) allo stesso risultato finale. Quindi :

- se X1 e' valida allora si escludono X2, X3, Z1, Z2

Restano due possibili opzioni:

"X1" - "Z3" : INS 3C solo de-cypher / INS 04 cypher & masking

"X2" - "Z1" : INS 3C unmasking & de-cypher / INS 04 solo cypher

Per selezionare quella giusta e' necessario esaminare piu' in dettaglio i risultati relativi alla "catena 3C/3A/04/02". Si consideri un'INS 3C nella generica forma :

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

Una successiva INS 3A restituisce le due DW che vengono *separatamente* ri-cryptate attraverso l'INS 04/02.

Proviamo a dividere i risultati relativi alla prima CW da quelli relativi alla seconda CW :

CW1								risp. INS02							
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								risp. INS02							
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	8D	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

Si scopre immediatamente la presenza di una *permutazione* dei byte e di una loro ulteriore elaborazione. Quest'ultima risulta essere un semplice XOR, infatti applicando tale operatore tra un byte della CW e l'omologo presente nella risposta all'INS 04/02, si ottiene sempre lo stesso risultato. Si riesce a determinare un **ottetto a valori noti** da utilizzare nello XOR. Si nota un differente risultato tra CW1 e CW2 (diverse permutazioni e diverso ottetto "noto"), ma il processo e' del tutto analogo. Effettuando le prove con tabelle hash diverse oppure su provider differenti si ottengono i medesimi risultati, quindi non vi e' dipendenza da tali parametri.

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

Il processo di *masking CW* prevedera' prima la permutazione e poi l'operazione di XOR con l'ottetto "noto". E' evidente la possibilita' di scambiare tra loro i processi di mascheramento e XOR, avendo l'accortezza di permutare opportunamente anche i byte dell'ottetto "noto". Sebbene cio' non comporti nessuna differenza ai fini del risultato finale, potrebbe risultare utile per comprendere meglio il meccanismo di funzionamento del masking. Per adesso possiamo trarre due conclusioni :

- Vista la diversita' di trattamento tra ottetto 1 ed ottetto 2 della C1 3C, si puo' affermare senza dubbio che l'unmasking sia incluso nel processo di messa in chiaro delle CW, il quale cambia a seconda dell'ottetto considerato.

- La coppia di INS 04/02 *potrebbe* non eseguire il masking *oppure* eseguirlo con permutazioni e XOR diversi.

#### Chi usa il masking? (Ipotesi da verificare)

- Il crypt di un ottetto mediante INS 04 *non dovrebbe* utilizzare il masking
- Il decrypt delle chiavi (INS 40 - Nano 90/91) *dovrebbe* utilizzare l'unmasking

**Sfortunatamente, questo processo di mascheramento degli ottetti non sembra uguale a quello subito dai byte in SuperEncryption**, infatti data la seguente risposta ad una C1 36 con key 0F :

#### *Risposta criptata*

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

#### *Risposta in chiaro*

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

...si scopre che non e' possibile ricostruire le tabelle di permutazione/XOR in funzione della somma WORD, tramite la seguente procedura...

- 1) Si annota il primo ottetto criptato
- 2) Si varia il primo ottetto in chiaro mantenendo fissi i restanti bytes (cioe' si fissa la "somma WORD" del secondo ottetto)
- 3) Si annota la "somma WORD mod 4000" del secondo ottetto
- 4) Si annota il primo ottetto in chiaro
- 5) Si cripta tramite INS 04/02 (ecco perche' serve la key 0F) il primo ottetto in chiaro
- 6) Si analizzano i risultati alla ricerca delle permutazioni e degli XOR
- 7) Si itera il procedimento variando il secondo ottetto (conseguente variazione della somma WORD)
- 8) Si analizzano i risultati alla ricerca della relazione tra permutazioni, XOR e somma MOD 0x4000

La procedura avrebbe funzionato se nella SuperEncryption avesse agito *prima* il cypher e *poi* il masking, ma non si riesce a giungere a nessuna evidenza sperimentale analoga al caso delle CW/DW. Quindi :

**su ogni ottetto processato agisce *prima* il masking e *poi* il cypher,  
in maniera inversa a quanto accade nel processo di decodifica delle CW**

Il metodo per la costruzione della tabella va allora modificato nel seguente modo :

- 1) Si annota il primo ottetto criptato
- 2) Si varia il primo ottetto in chiaro mantenendo fissi i restanti bytes  
(cioe' si fissa la "somma WORD" del secondo ottetto)
- 3) Si annota la "somma WORD mod 4000" del secondo ottetto
- 4) Si annota il primo ottetto in chiaro
- 5) Si itera il procedimento variando il secondo ottetto (conseguente variazione della somma WORD)
- 6) Si costruisce una INS 3C con **04 D1 A1 A2 A3 A4 A5 A6 A7 A8** ecc. dove A1...A8 e' dato dal masking (secondo l'INS 3C-Nano D1) applicato al primo ottetto della risposta *criptata* della C1 36
- 7) Si inviano C1 3C e C1 3A
- 8) Si annota la prima DW (che corrisponde al risultato del decypher del primo ottetto della C1 36, senza che sia applicato l'unmasking)
- 9) Si confronta la DW con l'ottetto in chiaro
- 10) Si analizzano i risultati alla ricerca della relazione tra permutazioni, XOR e somma MOD 0x4000

Bisognerà tenere conto del fatto che ogni ottetto dovrebbe essere permutato sempre in un certo modo, determinato dalla sua posizione nella INS.

#### **Ipotesi sulla somma WORD (citazione)**

**3F FF** che quota di informazioni può portare?

**FF** è un bel puntatore per una tabella di 256 byte con cui risolvere il problema della pseudochiave. Rimarrebbe il problema della permutazione, ma **3F** corrisponde (contando anche lo 0) ad un bel 64 decimale che guarda caso è anche il quadrato di 8. Potrebbero essere le permutazioni, ma non sarebbero tutte quelle possibili; per coincidenza sono anche un numero che con un'altra tabella di 256 byte tramite il setting dei bit consente di dare le info corrette per la permutazione stessa.

Ultima considerazione sul masking : si può escludere che tale processo usi lo stesso algoritmo dell'envelope, una prima motivazione proviene dall'analisi dei dati (le particolari proprietà statistiche di NLB ed LB negli ECM/EMM) ed una seconda, più importante, proveniente dalla power analysis, dato che l'envelope utilizza il *presunto* crypto-engine (o criptoprocessore), mentre le operazioni di (de)crypt ed (un)mask non lo usano.

## Il calcolo Signature nel SEKA2

Prendiamo come punto di partenza il calcolo signature del SEKA 1:

### Signature SEKA 1 (citazione)

Questa procedura utilizza l'algoritmo di criptaggio SEKA. I dati utilizzati per il calcolo sono quelli del corpo dell'istruzione fino al Nano 82. Si suddividono i dati in blocchi da 8 byte ciascuno, fino all'ultimo blocco che, ovviamente, può essere anche più corto di 8 byte. In tal caso si aggiungono tanti byte "00" quanti ne sono necessari per arrivare alla dimensione giusta del blocco. La procedura completa è la seguente:

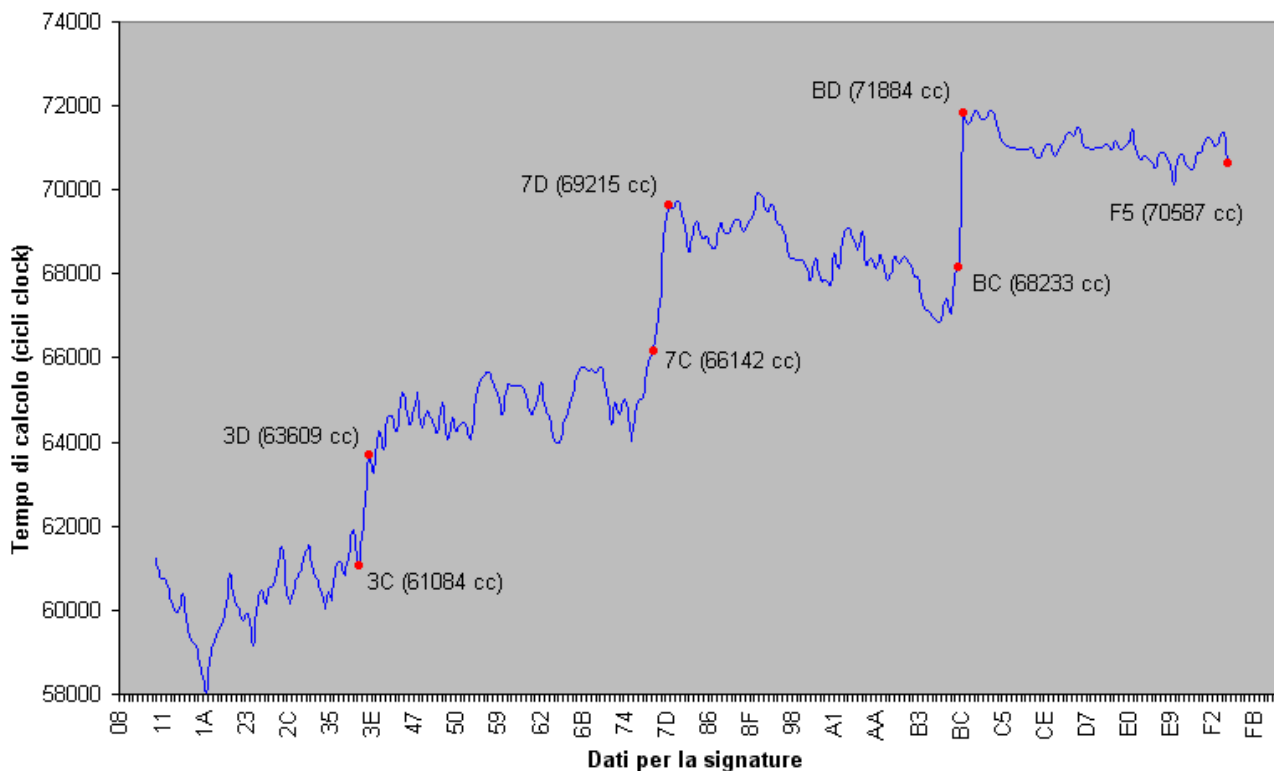
- 1) Si inizializza l'hashbuffer (contiene 8 byte)
- 2) XOR tra il blocco di 8 byte di dati di ingresso ed il contenuto del buffer
- 3) Esecuzione dell'algoritmo di criptaggio SEKA ( il risultato viene posto nel buffer )
- 4) Se i blocchi da processare non sono terminati si prende il blocco successivo e si torna al passo 2

Al termine della procedura nel buffer è presente la Signature calcolata. L'inizializzazione dell'hash-buffer varia a seconda del valore assunto dai 3 bit più significativi di P1:

- a) **x01xxxxx** Hash-Buffer inizializzato con : **UA UA UA UA UA UA 00 00**  
( Unique address – Il numero seriale della carta )
- b) **x10xxxxx** Hash-Buffer inizializzato con : **PP PP PP PP 00 00 00 00**  
( La PPUA del Provider a cui è indirizzato il comando )
- c) **x11xxxxx** Hash-Buffer inizializzato con : **SA SA SA 00 00 00 00 00**  
( Lo Shared Address – Primi 3 byte della PPUA )
- d) **x00xxxxx** Hash-Buffer inizializzato con : **00 00 00 00 00 00 00 00**

Nel SEKA 1, quindi, erano eseguiti un certo numero di crypt per giungere alla signature, il loro numero dipendeva dalla lunghezza del comando (in particolare, dal numero di ottetti in cui poteva essere suddivisa l'INS). Per cercare di comprendere il calcolo signature in SEKA 2 si può tentare un'analisi dei tempi di risposta, ottenendo dei risultati interessanti...

I dati della prova si riferiscono al tempo impiegato da una INS 38 per la verifica della signature.



Contrariamente a quanto si credeva, il calcolo della signature non e' del tutto indipendente dalla lunghezza del comando : si osserva un sensibile incremento del tempo impiegato dalla verifica della signature nei seguenti passaggi :

3C → 3D byte

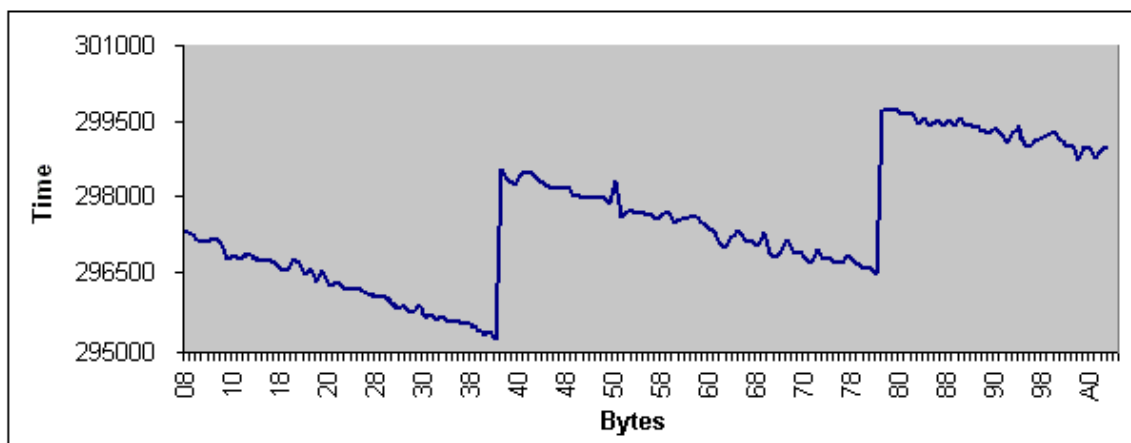
7C → 7D byte

BC → BD byte

Viene naturale supporre che l'algoritmo di calcolo lavori a blocchi di 64 byte, i risultati dell'elaborazione di ciascun blocco vengono concatenati ed il risultato finale e' un ottetto. Esistono molti algoritmi che utilizzano 64 byte di dati, ma restituiscono un numero di byte superiore ad 8. Potrebbe quindi essere presente un processo di "troncamento" del risultato (come nel calcolo-signature di IRDETO 1).

Ci si potrebbe chiedere il motivo del primo "salto" a 3C, anziche' quando la lunghezza dei dati vale 64 (40 in esadecimale). Probabilmente vengono aggiunti alcuni byte in fase di preparazione del calcolo (accade in alcuni algoritmi noti). I quattro byte "mancanti" potrebbero trattarsi dell'equivalente (SEKA 2) dell'inizializzazione dell'hashbuffer (SEKA 1).

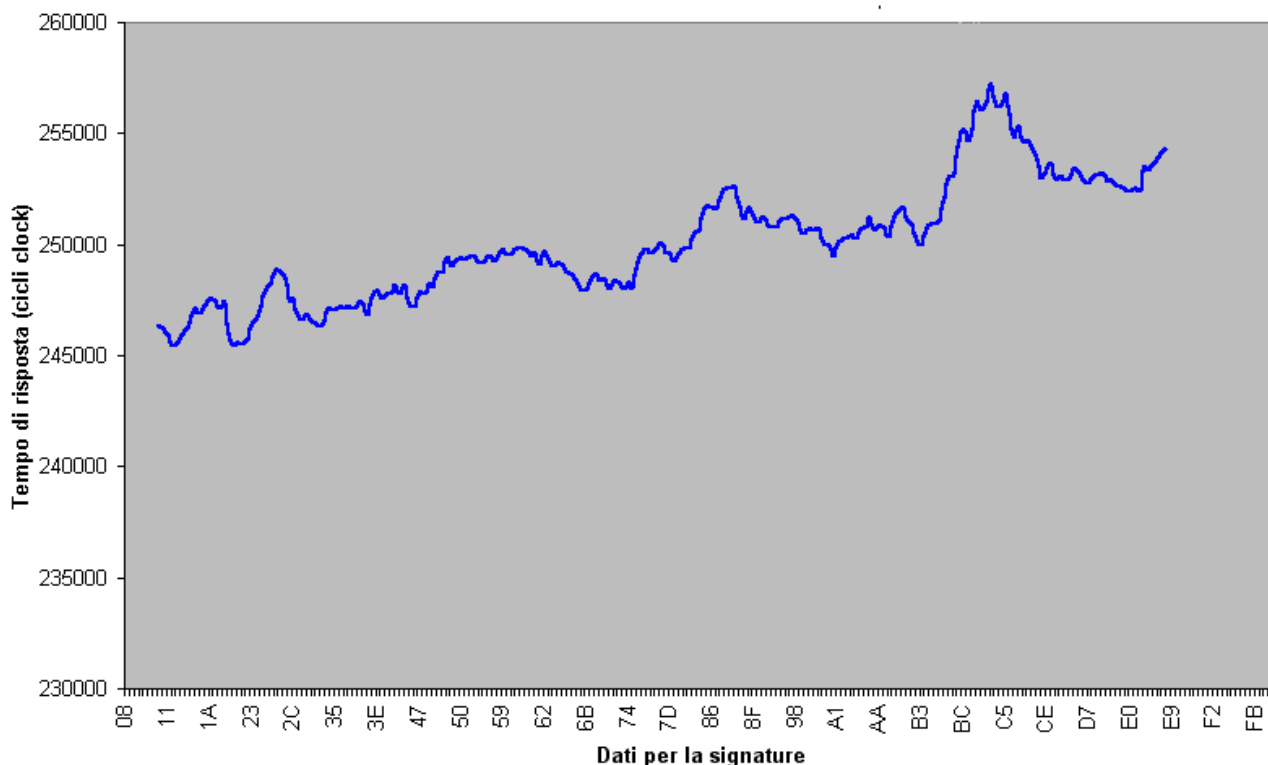
Dal grafico non si riesce ad osservare un'ulteriore comportamento che solo prove piu' approfondite (ed accurate) mostrano : la presenza di una rampa discendente sovrapposta all'andamento a gradini. Essa non dipende dal calcolo signature vero e proprio, ma compare nelle prime fasi dell'esecuzione dell'INS.





### Signature SEKA 2 - Il timing del 9002-A

Consideriamo il processo di esecuzione delle INS (*Quick reference a pag. 9/10*) : il test sulla presenza del Nano 82 avviene prima di tanti altri controlli, ma il tempo necessario a restituire questo status risulta maggiore di quello relativo a tanti controlli successivi. Questo significa che vengono eseguite delle operazioni prima di restituire questo status. Si osserva che tra lo status 9002\_A e 9002\_B c'è una differenza pari al tempo necessario al prelevamento della key dall'EEPROM e dalla sua messa in chiaro. Questo lascerebbe pensare che il calcolo signature sia comunque eseguito, anche in assenza del nano 82. L'analisi dei tempi di risposta sembra avvalorare questa ipotesi :



**Il tempo di risposta ha un andamento "a gradini", come il grafico relativo al calcolo-signature.**

Si pensi che lo status 9034, prodotto dal fallimento del controllo immediatamente precedente alla verifica signature (quello del 9002\_B), non presenta questo andamento.

#### Fantaseka (citazione):

Se fosse così, la card non trovando il nano 82 salterebbe di brutto alla verifica signa, la key non è stata caricata ed userebbe il key-buffer così com'è (che può contenere il risultato dell'ultima key operation) quindi è non predicibile (ed il controllo fallisce), ma se nel key-buffer becca una key che valida la signa, allora passa dritto a fare decrypt SE etc etc. E visto che di fantaseka si tratta allora ammesso di poter forzare una key conosciuta nel suo bel bufferino.....

### Signature SEKA 1 - Routine alternativa (Tratta da un post datato Marzo 2001)

Per completezza, viene riportato l'algoritmo per il calcolo signature in SEKA 1 usato da alcune particolari istruzioni (e diverso dall'algoritmo standard, sfruttato dagli ECM/EMM).

La sintassi è la seguente:

**C1 INS P1 P2 LEN + DATA PACKET**

#### **P1**

Bit 0...3 : Indice del Provider a cui inviare il comando  
Bit 4 : Utilizzo di Key primaria oppure Key primaria + secondaria  
Bit 5,6,7 : VEDI DOPO !!!

#### **P2**

Bit 0...3 : Indice della key da usare per il (de)crypt SuperEncryption  
Bit 4,5,6 : VEDI DOPO !!!  
Bit 7 : Indicatore della SuperEncryption

#### **LEN**

E' la lunghezza del "data packet"

A cosa servono i bit 7,6,5 di P1 ed i bit 6,5,4 di P2 ?

Servono all'inizializzazione delle keys utilizzate negli algo di crypt/decrypt e di calcolo signature.  
Supponiamo di avere un'INS indirizzata al provider 01, con key index pari a 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 key utilizzata per il calcolo signature non e' la key 06, indicata dal low-nibble di P2.

Tale key viene utilizzata per criptare un ottetto che è così generato :

- si parte da un ottetto *nullo* : 00 00 00 00 00 00 00 00

- a partire dall' (X+1)-esimo byte di dati per una lunghezza di Y viene sovrascritto tale ottetto.

### Esempi:

```
X=00; Y =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 -> 02 03 04 00 00 00 00 00
X=01; Y =04; 00 00 00 00 00 00 00 00 -> 01 02 03 04 00 00 00 00
```

Il caso in esame e' il secondo, quindi:

```
- Ottetto generato :    02 03 04 00 00 00 00 00
- Key primaria 06 :    49 63 6F 6E 43 6F 69 6C
- Ottetto criptato :    03 B1 FF 19 98 6D 2F 82
```

Quest'ultimo rappresenta la chiave con la quale va calcolata la signature, ma non e' finita qui, ci sono altre due cose da impostare :

1) Nel normale calcolo della signature deve essere inizializzato l'hashbuffer tramite, a seconda dei casi, UA / SA etc... Anche qui e' necessario fare cio', tale ottetto sara' tutto *nullo* in partenza, poi verrà sovrascritto a partire dal *primo* byte di dati per una lunghezza di X.

### Esempi:

```
X=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 -> 10 01 00 00 00 00 00 00
X=01 ; 00 00 00 00 00 00 00 00 -> 10 00 00 00 00 00 00 00
```

2) Nel normale calcolo della signature c'e' un indice che ci da' il punto di partenza. Ordinariamente questo indice vale zero, questo significa che la signature viene calcolata a partire dal *primo* byte di dati. In questo caso, invece, questo puntatore assumerà il valore :

$$\text{Offset} = X + Y$$

Dagli esempi precedenti risulta che l'ottetto di partenza per il calcolo della signature e' :

```
Offset = 00 --> 10 01 02 03 04 05 06 82      (Ricordarsi che
Offset = 05 --> 05 06 82 00 00 00 00 00      e' una routine del
Offset = 05 --> 05 06 82 00 00 00 00 00      SEKA 1, NdA)
```

In pratica, vengono scartati i primi X+Y bytes nel calcolo.

```
- Hash-buffer : 10 01 00 00 00 00 00 00
- 1° Ottetto :  05 06 82 00 00 00 00 00
- XOR tra i due: 15 07 82 00 00 00 00 00
```

Cripto il risultato con la key costruita in precedenza (cioe' 03 B1 FF 19 98 6D 2F 82) ottenendo la signature : 54 EC C2 B8 4C A7 8A A5

### Signature SEKA 2 - Gli EMM post-black Monday sono davvero inutili ?

E' noto che le INS costruite attraverso il bug 36/38 sono indirizzabili solamente alla card con cui si e' realizzato il comando stesso. Cio' e' causato dal valore assunto da P1 (*vedi pag.86 - signature SEKA 1*) che impone l'inizializzazione dell'hashbuffer con il *card serial*, diverso per ogni carta. Sfortunatamente le operazioni utilizzate per l'inizializzazione sono diverse dal SEKA 1 :

#### Prove sulla Signature (citazione):

Una C1 36 relativa ad una certa card non e' buona anche per un'altra anche se la key utilizzata e' uguale : la C1 36 cripta e segna la risposta inizializzando l'hash-buffer con il sernum della card (ultimi 6 bytes del C1 0E) pertanto detta stringa puo' essere reinviata con la 38 solo a quella stessa card che l'ha creata.

(...)

Ho fatto un esperimento, ho preso la mia vecchia 4.0 ed ho creato una MK = 00 00 00 00 00 00 00 00 poi ho inviato, dopo essermi calcolato la signa, il seguente comando :

```
C1 40 01 02 12 80 FF FF FF FF FF FF FF 82 B4 60 A0 EB 43 A4 92 B2
[90 00]
```

Adesso supponiamo che il sernum della mia vecchia 4.0 fosse 00 00 01 02 03 04. Ho reinviato :

```
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]
```

Inizializzando il buffer con il sernum ed eseguendo lo XOR dei primi byte della stringa con il suddetto la signa era uguale. A questo punto non ho resistito alla tentazione : ho preso una 38 buona, ho eseguito lo XOR dei primi 6 bytes nella modalita' di cui sopra ed ho inviato alla mia V7 -> C1 38 01 ..... etc. etc.

Risposta : [90 02] ... l'hash non viene piu' inizializzato come prima

(...)

certo e' che se si riuscisse a trovare una 38 del tipo C1 38 01 xE 13 ..... ove x =9, B, F essa farebbe scattare il bug su tutte le card...

Quindi non e' ancora conosciuto un modo per rimuovere la dipendenza dal "card serial" e costruire il cosiddetto comando *universale* valido per far scatenare il bug su tutte le **V7.0**.

*Sono in fase di studio dei procedimenti con cui costruire un EMM "alterato" e buggabile a partire da EMM con LEN = 63 (con nanocomando che impedisce l'utilizzo del bug). Essendo dei metodi che, allo stato attuale, non garantiscono il sistematico raggiungimento del risultato voluto, non sono riportati in questa edizione del DOC.*

## Esecuzione "parziale" delle INS : il metodo send-wait-reset

*Per questa esercitazione e' necessario :*

*SmartTimer 1.82x (beta)*

In una delle ultime "beta" del programma e' presente un'inedita opzione, "**sw-Unloop**" che e' cosi' spiegata :

*Invia soltanto il primo comando, poi attende un tempo in millisecondi pari al numero indicato nel riquadro a destra del pulsante "Multisend" e poi resettta la card. Utile per far eseguire solo una parte dell'INS.*

Cio' significa che la card viene resettata prima del completamento dell'INS, ottenendo un'esecuzione parziale. Questo puo' essere utile per far eseguire solo un certo numero di nanocomandi di un ECM/EMM. Consideriamo ad esempio un' EMM d'attivazione e supponiamo di voler riscrivere le chiavi, senza alterare la PPUA : e' sufficiente resettare la card *prima* che venga eseguito l'ultimo nanocomando (Nano 41 - Scrittura PPUA). Per determinare l'istante di tempo giusto conviene procedere per tentativi, partendo da un valore piuttosto piccolo ed incrementandolo finche' non si e' ottenuta l'esecuzione dei soli nanocomandi voluti. Si deve ricordare che nella card sono implementate delle routine che introducono ritardi "random" : e' possibile che inviando piu' volte lo stesso comando, con stesso tempo di attesa in millisecondi, si abbiano effetti diversi (tipicamente, l'esecuzione di un nanocomando in piu' od in meno). Tramite un ECM e' possibile dare un altro esempio dell'utilizzo di questa "feature". Supponiamo di voler cancellare i preview record senza scriverne uno nuovo : e' sufficiente costruire un C1 3C - Nano 27 con data *alta* (= maggiore della data dei preview record presenti, minore della data di fine abbonamento) e resettare prima della creazione del record Ax (che avviene *dopo* la cancellazione dei *vecchi* record).

Il metodo *send-wait-reset* permette di scoprire una caratteristica delle card V7.0 : la presenza di una routine di controllo sulle scritture in EEPROM, presente anche nelle V6.0 ed assente nelle precedenti versioni.

Supponiamo di voler cancellare una key (INS 40 - Nano 10) : la loro cancellazione prevede l'annullamento di tutti i byte del record. Interrompendo l'esecuzione del nanocomando prima della cancellazione del record descriptor (l'ultimo byte) si avrebbe *teoricamente* un key-record parzialmente alterato (e quasi sicuramente inutile a causa del key-checksum non valido). Questo procedimento funziona, ad esempio, su una V4.1 OKI, mentre provandolo su di una V6.0 non da' nessun effetto : se un write non e' portato completamente a compimento, al successivo avvio della card (in seguito ad un reset) e' ripristinata la configurazione precedente la scrittura.

Visto che anche sulle V7.0 la prova della cancellazione parziale del key-record fallisce, si puo' ipotizzare che anch'essa adotti un sistema di verifica delle scritture in EEPROM.

## Altro punto di vista : Power Analysis

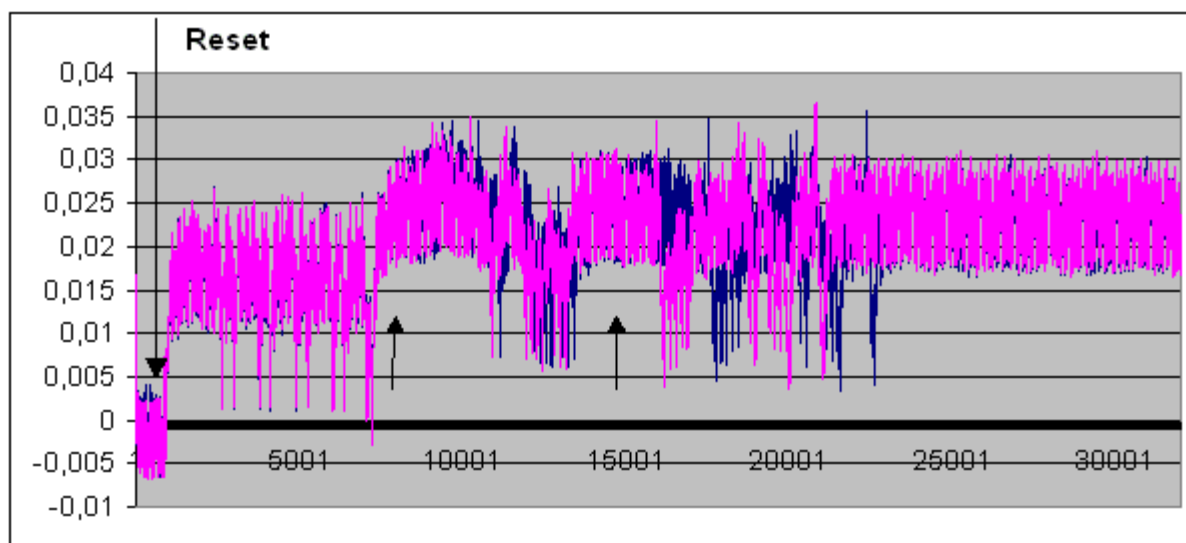
*Analisi della V7 orientata all'hardware - Tratto da un Forum-Sat*

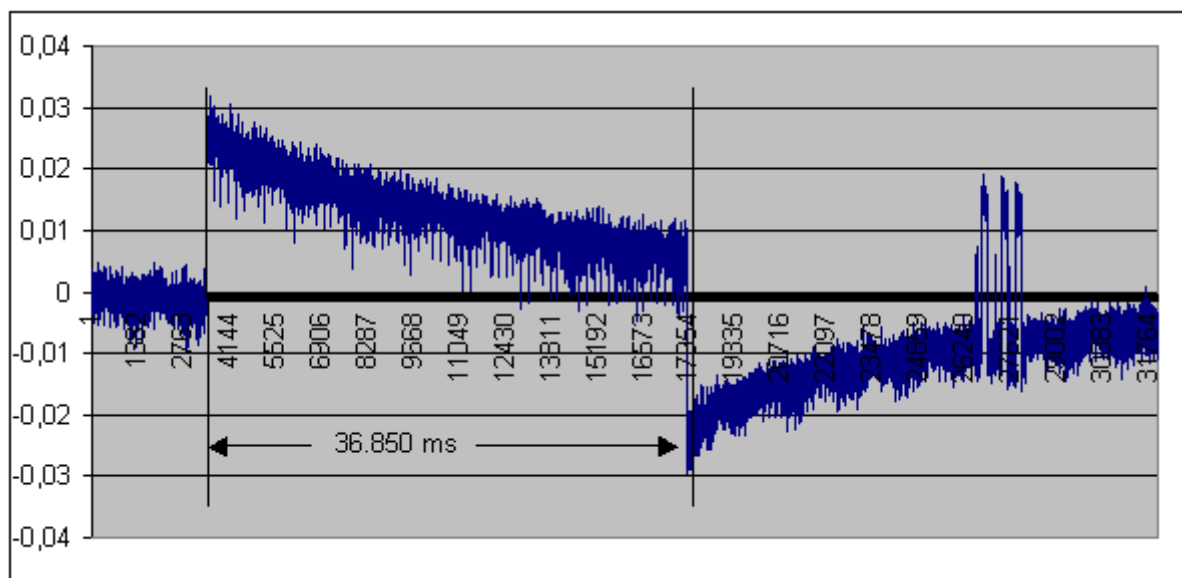
Per prima cosa si esamina visivamente la superficie dei contatti e del chip sottostante, il tutto ricondurrebbe al produttore GEMPLUS (anche alla luce di quello che si sente in giro nei Forum pubblici). Pensando alla data di produzione/distribuzione, alle caratteristiche elettriche ed ai partner abituali della GEMPLUS si potrebbe ipotizzare che l'hardware usato sia il Philips **P8WE5xx** (con criptoprocessore). Altra candidata potrebbe essere una variante della serie **AT90SCxxx** (Atmel), come si sente dire nei Forum francesi. Anche la Siemens/Infineon possiede dei modelli con crypto-engine, ad esempio le **66CXnnn**, e' pur vero che gran parte degli hardware precedenti hanno subito con successo attacchi software volti al dump, quindi potrebbe essere stato necessario un cambio di fornitore. Potrebbe addirittura trattarsi dello stesso micro delle DSS-HU card (**TMS370C-P3**), ma sono solo ipotesi.

Per quanto riguarda le caratteristiche elettriche, la card funziona in due intervalli di tensione distinti: 2.7-3.3V e 4.5-5.5V (tensioni tipiche rispettivamente delle famiglie logiche low-voltage e standard) con rilevamento dei livelli fuori specifica e (talvolta) conseguente reset. Il range di frequenze accettato si estende da 0.9 MHz fino a circa 6 MHz, con rilevamento di valori fuori banda e reset conseguente. Pare che la dispersione dei valori assunti da questo limite superiore sia piuttosto alta.

La card non sembra completamente protetta dal glitch...alcuni glitch provocano modifiche all'esecuzione del firmware, senza che avvenga il reset, ma non si e' trovato sistematicamente un uso "utile" (anche se rimane una via di sperimentazione aperta). Nella V7 si ha un uso abbondante delle random delay routine, ma **e' scomparso il jitter del clock** presente sulla V6; il jitter rende molto difficoltoso (se non, addirittura, inapplicabile) l'utilizzo dell'unlooper.

Utilizzando l'oscilloscopio digitale si possono aprire nuove strade per la comprensione dei fenomeni che avvengono all'interno della card. Il metodo consiste nell'analisi dell'assorbimento della corrente (Simple Power Analysis), interponendo, ad esempio una resistenza tra il pin di GND della card e la massa vera e propria. L'universo che si apre e' immenso: se la card non contiene protezioni contro Simple Power Analysis o Differential Power Analysis, il consumo del microcontrollore rivela informazioni sullo stato dei registri e/o sul tipo di opcode in esecuzione in quel momento.





Nel primo grafico si osserva il comportamento durante i cicli di clock immediatamente successivi al RESET, in due misurazioni distinte (blu e magenta). Si vedono chiaramente i cicli di clock esatti in cui viene richiamata la "random delay routine" (indicati dalle frecce).

Nel secondo grafico (con scala temporale diversa) si osserva il consumo durante l'esecuzione di una C1 38. Si nota l'incremento di consumo dovuto *quasi sicuramente* alla presenza di un criptoprocessore. La cosa piu' sorprendente e' la durata dell'impulso ad alto consumo, dato che e' costante qualunque sia il clock, cosa che potrebbe suggerire la presenza di un PLL interno.

(...)

I tre picchi di breve durata ed alto consumo dovrebbero riferirsi alla verifica-signature.

(...)

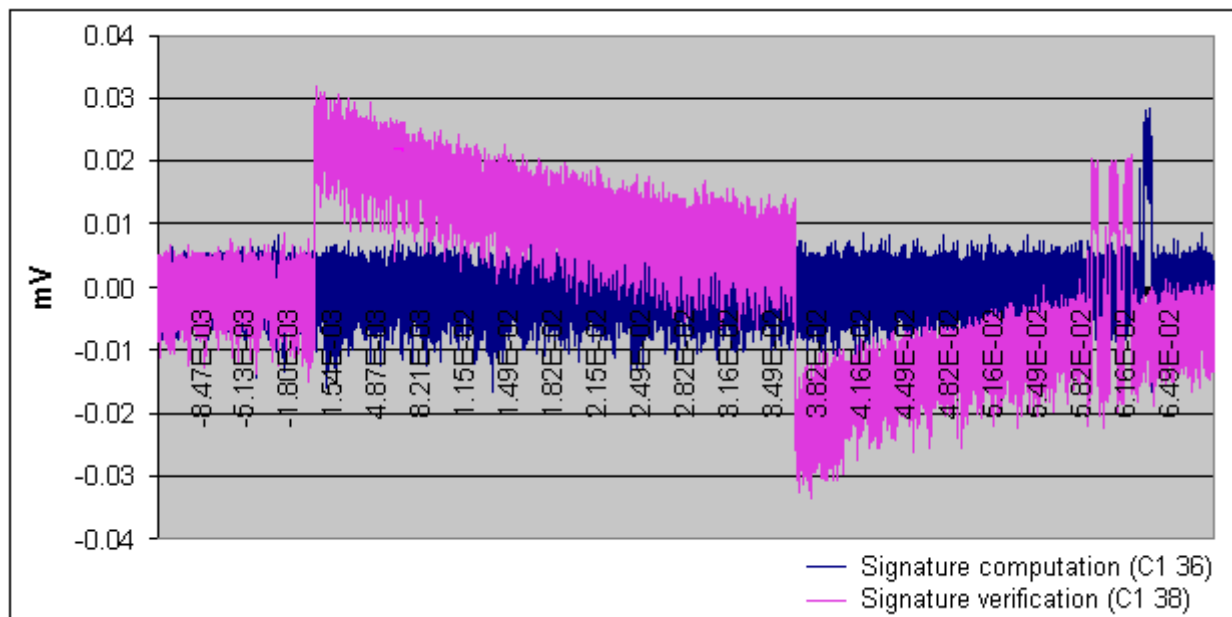
Il problema principale e' capire se l'analisi della corrente assorbita e' un buon metodo per ottenere informazioni dai moderni microcontrollori (**AT90SC**, **P8WE**, **DS5002FP**...). La risposta e' senza dubbio affermativa, perche' (*come vedremo piu' avanti*) non si notano protezioni contro Single Power Analysis tramite rumore pseudo-random, ne' nel core, ne' nel cryptoprocessore; l'unica contromisura implementata e' l'introduzione di random delay routine (una contromisura software facile da aggirare in sede di analisi dei dati, ma un problema in fase di glitching).

(...)

Non so quale possa essere il microcontrollore utilizzato, probabilmente si tratta di un Philips o Atmel perche' hanno le stesse caratteristiche elettriche (intervallo di tensione e frequenza) e data di produzione *presunta* della V7.0; inoltre, il confronto tra le *power trace* (durante il reset, durante il de-envelope e durante la verifica della signature) mostra che V7.0 e V7.1 utilizzano lo stesso hardware. Probabilmente anche le V7.3 condividono lo stesso hardware, ma non e' stato verificato.

(...)

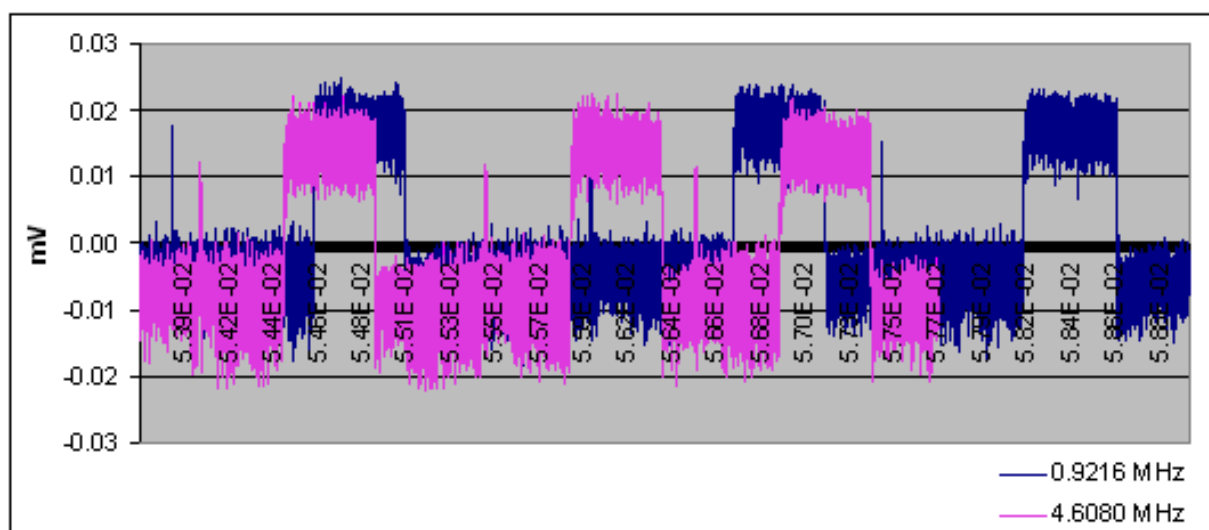
Nel grafico seguente sono confrontate le correnti assorbite durante l'esecuzione di una C1 38 e di una C1 36, allo scopo di confrontare il processo di generazione e quello di verifica della signature.



- Card clocked @4.608 MHz -

1) Usando una C1 38 con signature valida, si osservano 3 impulsi ad alto consumo, dopo circa 20 ms dall'esecuzione del de-envelope (identificabile con l'impulso "lungo"). Non e' detto che gli impulsi siano necessariamente 3 : utilizzando INS di lunghezza differente si possono ottenere 1+2 impulsi (come in figura), ma anche 2+2, 3+2, 4+2 confermando la teoria della verifica signature tramite blocchi di 64 byte (512 bit).

2) Il comando C1 36, invece, presenta un solo impulso (nel caso in esame). Allungando la risposta dovrebbe essere possibile l'ottenimento di un numero di impulsi variabile da 1 a 4.



3) Riducendo la frequenza di clock a 0.9216 MHz, la durata dei picchi (compreso quello "largo" di de-envelope) si mantiene costante.



Si conclude che :

- Generazione e verifica della signature non sono simmetriche
- Il cryptoprocessore esiste ed ha un clock autonomo (come, ad esempio, il **Philips FameX** crypto-engine)

Proseguendo nell'analisi della signature, si possono fare delle prove relative agli status 9002\_A e 9002\_B :

4) Usando una C1 38 con signature errata (status 9002\_B), si continuano ad osservare i 3 (o piu') impulsi, come nel caso di signature valida.

5) Usando una C1 38 senza Nano 82 (status 9002\_A), ci sono ancora i 3 impulsi, essi compaiono 10 ms dopo il de-envelope. Questo e' un BUG: la signature e' calcolata anche in assenza di Nano 82.

6) Usando una C1 38 senza Nano 82 e con un P2 tale da richiedere una key inesistente, si osserva lo stesso comportamento del punto 5. La signature e' calcolata senza inizializzare il key-buffer.

Nel caso fallisca un qualsiasi altro controllo, la signature non viene verificata.

#### Power Analysis - La fase di de-SSE (o de-envelope)

All'interno dell'impulso ad alto consumo si possono osservare, filtrando opportunamente il segnale letto, delle cose piuttosto interessanti : sono presenti 66 iterazioni di durata pari a 550  $\mu$ s (il cui assorbimento dipende dal valore assunto dai dati in elaborazione) intervallati da 8 ms a corrente piu' bassa. Ingrandendo una di queste iterazioni si osservano 24 sotto-cicli diversi tra loro (e diversi per ciascuna delle 66 iterazioni "principali")

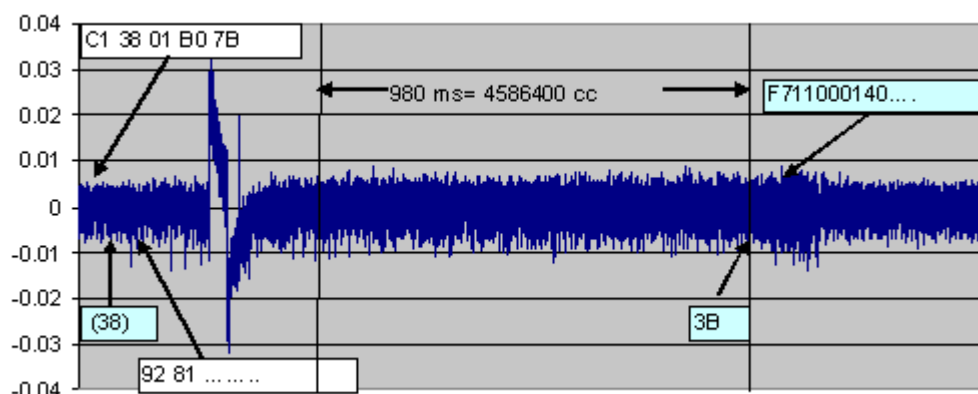
#### Power Analysis - Analisi del bug 9600

Inviando una C1 38 con zero dati da decriptare, del tipo :

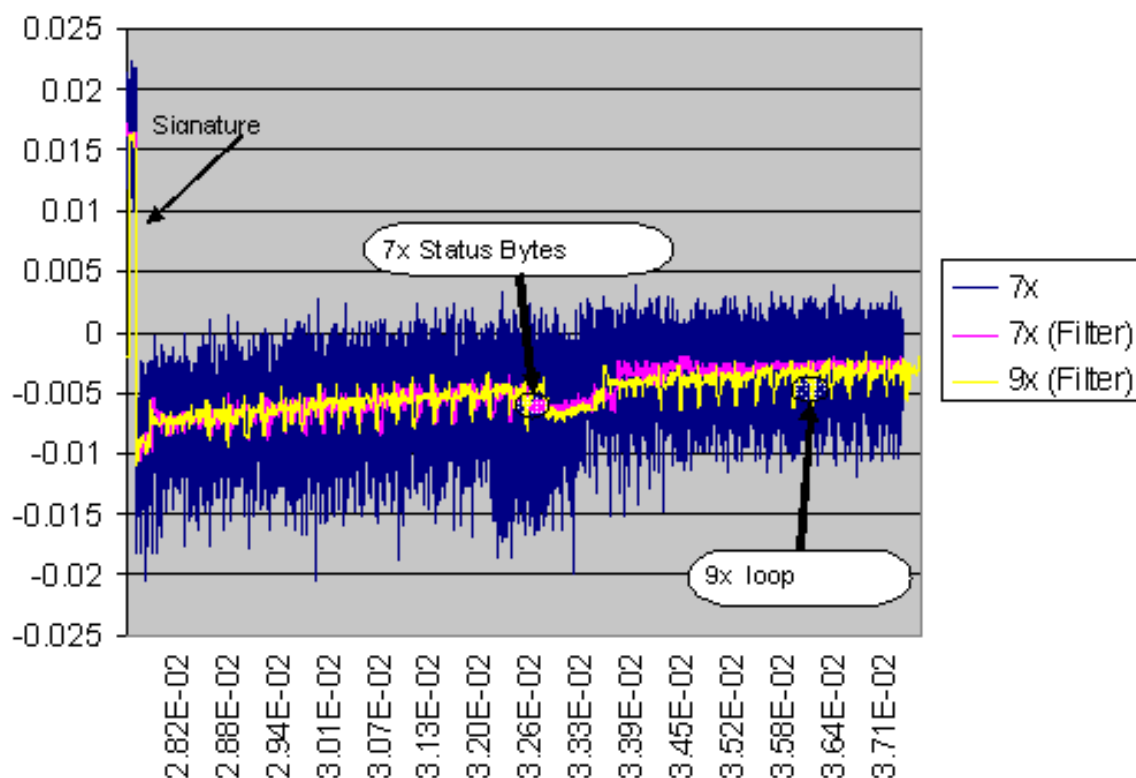
**C1 38 P1 P2 LEN 9x y1 .....**

avremo come risposta l'ATR. L'esecuzione dell'INS avra' evidentemente una parte comune alle C1 38 "normali" ed una parte comune alla routine di invio dell'ATR. Cio' varra' anche per l'andamento della corrente assorbita.

La figura seguente rappresenta un andamento tipico per un'INS del tipo indicato. Si nota il lungo intervallo di tempo che intercorre tra l'invio dell'INS e la restituzione dell'ATR (intorno ai 1400 ms).



Proviamo ad ingrandire una parte del grafico :



Confrontando il caso P3 = 9x col caso P3 = 7x (che risponde normalmente), si osserva che per entrambe viene eseguito il decrypt di un blocco (i 16 cicli operati dal decrypt sono chiaramente visibili nelle tracce "filtrate"), dopodichè la card entra in un loop piuttosto lungo che si conclude con la restituzione dell'ATR.

Il loop ha una durata superiore ai 4500000 cicli di clock (IPOTESI : *potrebbero* essere sufficienti a leggere tutta la RAM, ROM ed EEPROM della card, compresa una possibile area protetta, accedendo alla quale *potrebbe* prodursi il reset).

La routine dell'ATR è eseguita a partire dal suo *reale* punto d'inizio, ma senza i test sul cryptoprocessore e sul generatore di numeri casuali : non si vedono i relativi assorbimenti di corrente, visibili, invece, in seguito ad un normale reset della card.

## Last but not least

*Icon of Coil ... chi era costui ?*

*Parafrasando le parole di un celebre autore, si potrebbe dire che "Icon of Coil e' chi lo si crede"...*

*Un ringraziamento a coloro che hanno contribuito alla diffusione di questo documento, in particolare ai traduttori in altre lingue. 😊*

## Per chi ha gia' stampato la v3.02

*Quelle che seguono sono le pagine aggiunte e/o modificate rispetto alla vecchia edizione ( alcune sono soltanto modifiche estetiche o correzioni lessico-grammaticali ) :*

*Pagg. 1, 5, 7, 8, 9, 10, 11, 12, 19, 20, 22, 26, 27, 33, 33 bis, 36, 37, 42, 43 e da pagina 59 in poi.*

## Coming Soon

- *Il metodo delle chiavi gemelle (teoria)*
- *Calcolo Signature : ricerca di un possibile modello*
- *Masking : ricostruzione di XOR e permutazioni*
- *Crypt, Decrypt & collisioni : ricostruzione delle Tabelle Hash*

***That's all folks !***

***Icon of Coil***

***E2 82 AC 20 AC A4 ...***