Men who stare at bits

Die abenteuerliche Analyse des Datenformats eines real existierenden RFID-basierten bargeldosen Bezahlsystems

@nv1t, murx

SIGINT 2012, Cologne

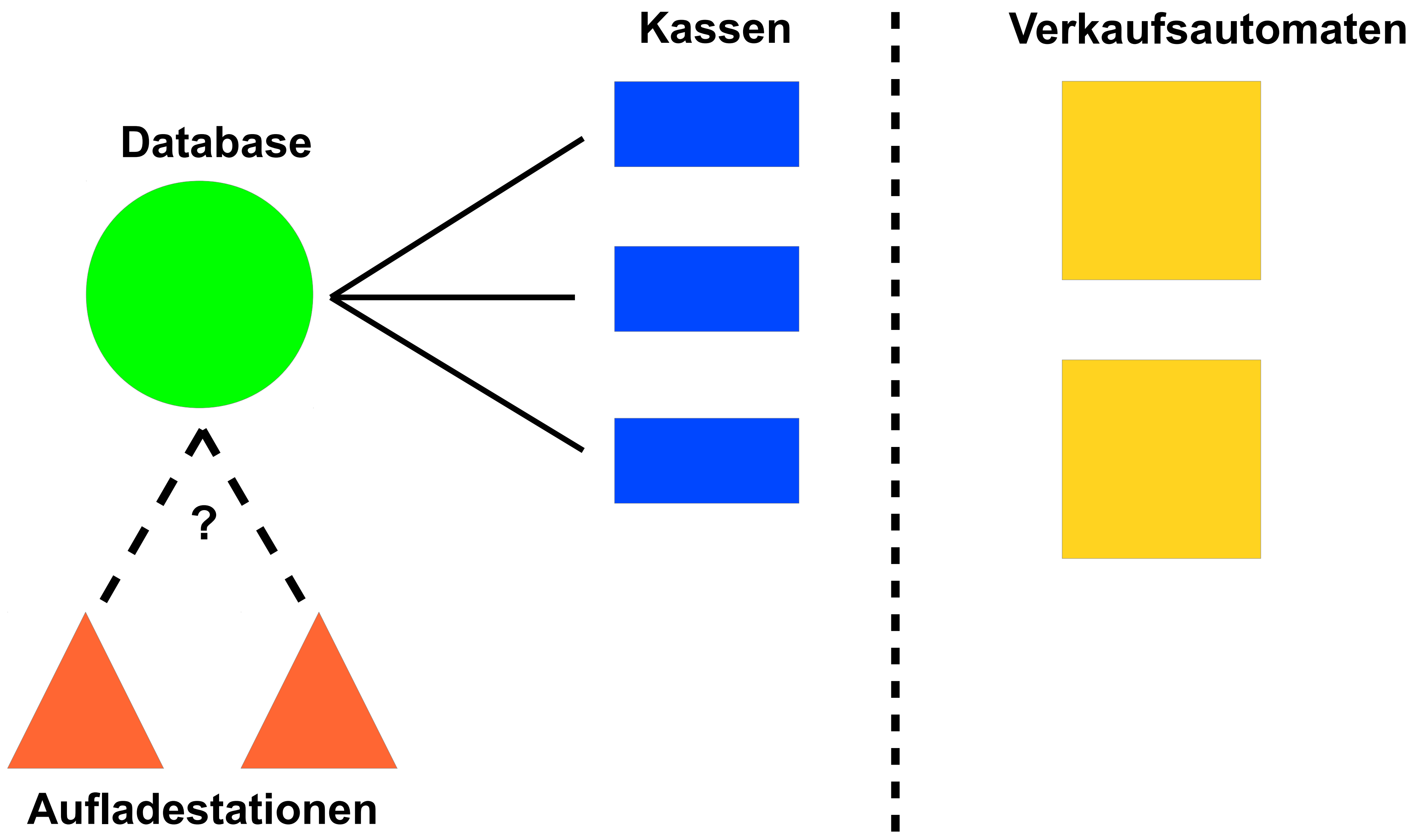
Infrastructure (1)

Verkaufsautomaten

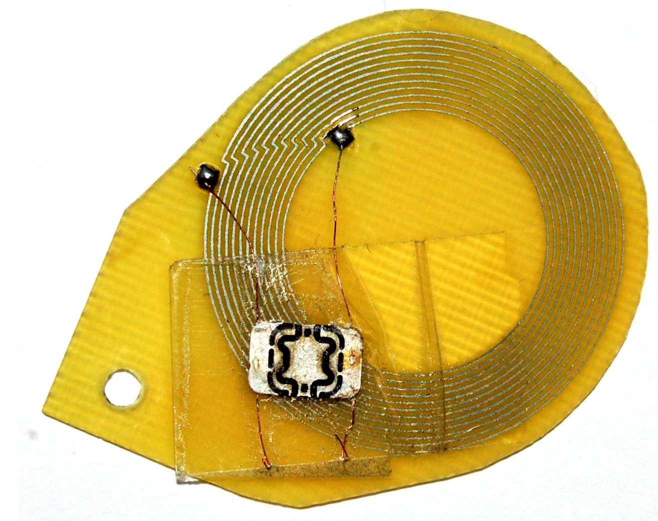
Kasse

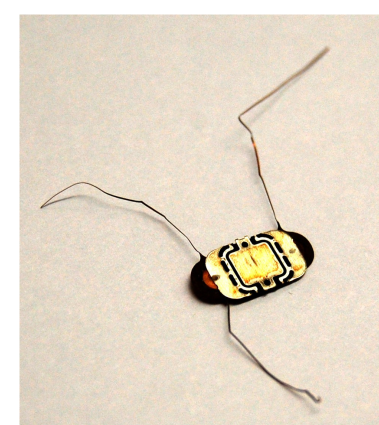
Aufladestation

Infrastructure (2)



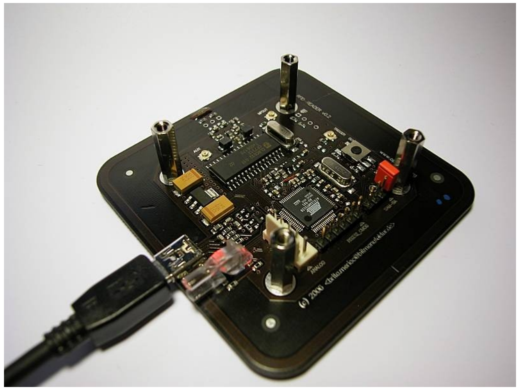
Looking inside the card





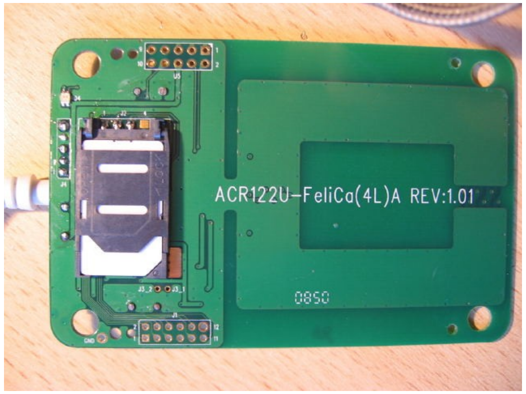
Card Reader

Proxmark3



OpenPCD

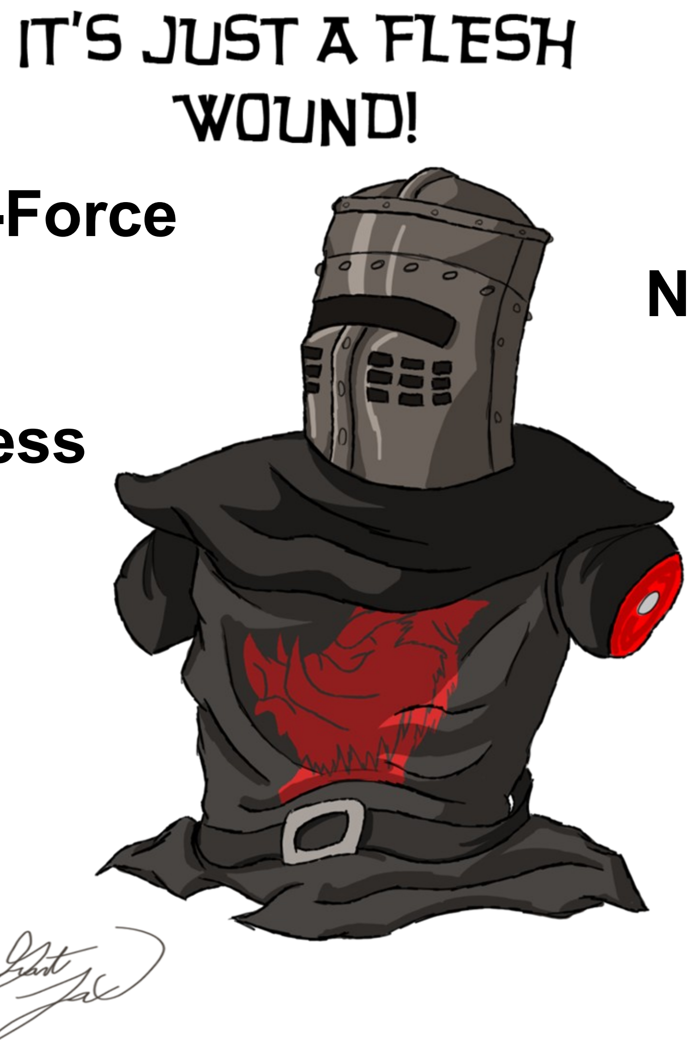
ACS ACR122



Touchatag

MIFARE Classic

Kontaktlose Chipkarte von NXP
Semiconductors



Brute-Force

Parity Weakness

Timing

Nijmegen

Curtious

Nested Authentication

Replay

Keystream Recovery

Playing around...

„klonbar“

„backupbar“

Dataformat

Sector		Block A	Block B
0	000	92 18 a7 10 3d 88 04 00	46 7a a8 05 32 31 3a 31
	010	32 02 00 00 01 38 01 38	01 38 00 00 00 00 00 00
	020	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
	*030	[redacted] 78 77	88 c1 [redacted]
1	040	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
	050	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
	060	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
	070	[redacted] ff 07	80 69 [redacted]
2	080	04 50 00 ff 00 dc 99 00	00 00 00 00 00 00 00 00
	090	00 03 03 00 00 00 00 00	00 00 00 00 00 00 c0 3a
	0a0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
	*0b0	[redacted] 78 77	88 69 [redacted]
3	0c0	ff 98 f7 1d ee 06 73 83	bd 47 d5 45 51 e7 fd dd
	0d0	2d cf 92 67 5b 1d 03 75	00 00 00 00 00 00 00 00
	0e0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
	*0f0	[redacted] 7f 07	88 69 [redacted]
4	100	c5 f7 1d ec bb ae b0 42	14 a0 4e 8e e0 32 6b 47
	110	f7 4d a3 0c 98 58 17 8c	26 0b 79 be 29 72 37 c7
	120	c6 49 ed 85 f4 fc c1 30	00 00 00 00 00 00 00 00
	*130	[redacted] 7f 07	88 69 [redacted]

- Manufacturer data
- M.A.D.
- Access Flags
- UID
- Institution ID (?)
- Cardnumber
- Cardtype
- Credit

Credit

1. Automaten ändern Daten *<u>auch ohne</u>* Preisänderung

0x100	0x110
c1 c0 18 ec 38 97 ee 37	f7 6f e6 0c bb 72 de c4
c1 c0 18 ec 38 97 ee 37	f7 6d e6 0c bb 72 9c e4
c1 c2 1c ec 38 97 ac 17	f7 6d e6 0c bb 72 9c e4
c1 c2 1c ec 38 97 ac 17	f7 6b e2 0c bb 72 9e c4
c1 c4 1c ec 38 97 ae 37	f7 6b e2 0c bb 72 9e c4
c1 c4 1c ec 38 97 ae 37	77 69 e2 0c ba 62 dc e4
41 c6 18 ec 39 87 f4 17	77 69 e2 0c ba 62 dc e4
41 c6 18 ec 39 87 f4 17	77 1f e6 0c ba 6a c6 e4
41 b0 18 ec 39 8f f6 37	77 1f e6 0c ba 6a c6 e4

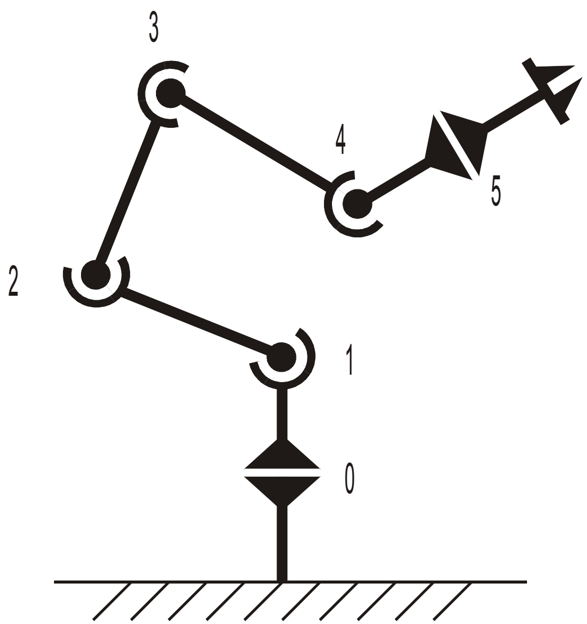
Abwechseln beim Schreiben der Daten

=> Counter oder RTC

Robot - Attack

[0x100:0x108]

```
c1 c0 18 ec 38 97 ee 37
c1 c2 1c ec 38 97 ac 17
c1 c4 1c ec 38 97 ae 37
41 c6 18 ec 39 87 f4 17
41 b0 18 ec 39 8f f6 37
41 b2 1c ec 39 8f b4 17
41 b4 1c ec 39 8f b6 37
c1 b6 18 ec 39 8f fc 17
c1 b0 18 ec 39 87 fe 37
c1 b2 1c ec 39 87 bc 17
c1 b4 1c ec 39 87 be 37
c1 b6 18 ec 39 87 a4 37
43 a0 58 ec 38 bf e6 17
43 a2 5c ec 38 bf e4 37
43 a4 5c ec 38 bf a6 17
43 a6 58 ec 38 bf ac 37
c3 a0 58 ec 38 b7 ee 17
c3 a2 5c ec 38 b7 ec 37
c3 a4 5c ec 38 b7 ae 17
```



Robot - Attack

3.5 h → 1150 Datensätze

```
[0x100:0x108]               [0x110:0x118]
c1 c0 18 ec 38 97 ee 37   f7 6f e6 0c bb 72 de c4
c1 c2 1c ec 38 97 ac 17   f7 6d e6 0c bb 72 9c e4
c1 c4 1c ec 38 97 ae 37   f7 6b e2 0c bb 72 9e c4
41 c6 18 ec 39 87 f4 17   77 69 e2 0c ba 62 dc e4
41 b0 18 ec 39 8f f6 37   77 1f e6 0c ba 6a c6 e4
41 b2 1c ec 39 8f b4 17   77 1d e6 0c ba 6a c4 c4
41 b4 1c ec 39 8f b6 37   77 1b e2 0c ba 6a 86 e4
c1 b6 18 ec 39 8f fc 17   77 19 e2 0c ba 6a 84 c4
c1 b0 18 ec 39 87 fe 37   f7 1f e6 0c ba 62 ce e4
c1 b2 1c ec 39 87 bc 17   f7 1d e6 0c ba 62 cc c4
c1 b4 1c ec 39 87 be 37   f7 1b e2 0c ba 62 8e e4
c1 b6 18 ec 39 87 a4 37   f7 19 e2 0c ba 62 8c c4
43 a0 58 ec 38 bf e6 17   75 0f a6 0c bb 5a d6 e4
43 a2 5c ec 38 bf e4 37   75 0d a6 0c bb 5a d4 c4
43 a4 5c ec 38 bf a6 17   75 0b a2 0c bb 5a 96 e4
43 a6 58 ec 38 bf ac 37   75 09 a2 0c bb 5a 94 c4
c3 a0 58 ec 38 b7 ee 17   f5 0f a6 0c bb 52 de e4
c3 a2 5c ec 38 b7 ec 37   f5 0d a6 0c bb 52 dc c4
c3 a4 5c ec 38 b7 ae 17   f5 0b a2 0c bb 52 9e e4
c3 a6 58 ec 38 b7 b4 37   f5 09 a2 0c bb 52 9c c4
43 90 58 ec 39 af f6 17   f5 3f a6 0c bb 5a 86 c4
43 92 5c ec 39 af f4 37   75 3d a6 0c ba 4a c4 e4
43 94 5c ec 39 af b6 17   75 3b a2 0c ba 4a c6 c4
43 96 58 ec 39 af bc 37   75 39 a2 0c ba 4a 84 e4
c3 90 58 ec 39 a7 fe 17   75 3f a6 0c ba 42 8e c4
c3 92 5c ec 39 a7 fc 37   f5 3d a6 0c ba 42 cc e4
c3 94 5c ec 39 a7 be 17   f5 3b a2 0c ba 42 ce c4
```

Robot - Attack

[0x100]

```
byte: 0        1        2        3        4        5        6        7
 bit: 76543210 76543210 76543210 76543210 76543210 76543210 76543210 76543210
      -----------------------------------------------------------------------
      11000001 11000000 00011000 11101100 00111000 10010111 11101110 00110111
      11000001 11000010 00011100 11101100 00111000 10010111 10101100 00010111
      11000001 11000100 00011100 11101100 00111000 10010111 10101110 00110111
      01000001 11000110 00011000 11101100 00111001 10000111 11110100 00010111
      01000001 10110000 00011000 11101100 00111001 10001111 11110110 00110111
      01000001 10110010 00011100 11101100 00111001 10001111 10110100 00010111
      01000001 10110100 00011100 11101100 00111001 10001111 10110110 00110111
      11000001 10110110 00011000 11101100 00111001 10001111 11111100 00010111
      11000001 10110000 00011000 11101100 00111001 10000111 11111110 00110111
      11000001 10110010 00011100 11101100 00111001 10000111 10111100 00010111
      11000001 10110100 00011100 11101100 00111001 10000111 10111110 00110111
      11000001 10110110 00011000 11101100 00111001 10000111 10100100 00110111
      01000011 10100000 01011000 11101100 00111000 10111111 11100110 00010111
      01000011 10100010 01011100 11101100 00111000 10111111 11100100 00110111
      01000011 10100100 01011100 11101100 00111000 10111111 10100110 00010111
      01000011 10100110 01011000 11101100 00111000 10111111 10101100 00110111
      11000011 10100000 01011000 11101100 00111000 10110111 11101110 00010111
      11000011 10100010 01011100 11101100 00111000 10110111 11101100 00110111
      11000011 10100100 01011100 11101100 00111000 101101
      11000011 10100110 01011000 11101100 00111000 101101
      01000011 10010000 01011000 11101100 00111001 101011
      01000011 10010010 01011100 11101100 00111001 101011
      01000011 10010100 01011100 11101100 00111001 101011
      01000011 10010110 01011000 11101100 00111001 10101111 10111100 00110
      11000011 10010000 01011000 11101100 00111001 10100111 11111110 000101
      11000011 10010010 01011100 11101100 00111001 10100111 11111100 00110111
```

Synchron

Asynchron

inc-cnt.txt

Counter

```
            [0x100]                                [0x110]

    byte:   3  5  4 2 6 1 1 1 5 1 1 1              3  5  4 2 6 1 1 1 5 1 1 1
     bit:   3  2  1 0 7 6 5 4 3 2 1 0              3  2  1 0 7 6 5 4 3 2 1 0

 cnt-bit:   11 10 9 8 7 6 5 4 3 2 1 0              11 10 9 8 7 6 5 4 3 2 1 0
     inv:   i  i  i i   i i i i                    i        i   i   i i i i
            -------------------------              -------------------------
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1
            0  0  1 1 1 1 0 1 1 1 1 0              0  0  1 1 1 1 0 1 1 1 1 1
            0  0  1 1 1 1 1 0 0 0 0 0              0  0  1 1 1 1 1 0 0 0 0 1
            0  0  1 1 1 1 1 0 0 0 1 0              0  0  1 1 1 1 1 0 0 0 1 1
            0  0  1 1 1 1 1 0 0 1 0 0              0  0  1 1 1 1 1 0 0 1 0 1
```

~~Verschlüsselung~~

***Verwurschtelung*:**

- Inversionsmaske (unterschiedlich für A/B)
- Bits vertauscht (Perm. identisch für A/B)
- **ABER:** Bitposition innerhalb eines Bytes unverändert

cnt.txt

Counter

Höhere Counter bits:

- Setze Counter auf 2**n-1
- Stecke Karte
  → Überlauf auf 2**n
- Counter >= 2**14-1
  → Error E.7E

Asynchrone Counter bits:

- Real time clock?
- Checksum

- counter // 256
- counter % 256
- checksum // 256
- checksum % 256

```
byte:   00000000 11111111 22222222 33333333 44444444 55555555 66666666 77777777
bit:    76543210 76543210 76543210 76543210 76543210 76543210 76543210 76543210
-------------------------------------------------------------------------------
xor A:            111 00          1   101          1      11   0
xor B:            101 11          1   001          0      10   0
```

Credit?

Pepsi - Attack

= 1.00 EUR

[0x100]

	0	1	2	3	4	5	6	7
EUR	76543210	76543210	76543210	76543210	76543210	76543210	76543210	76543210
16	01000101	10000011	01001001	11101100	00111011	10110100	01101000	00000011
17	11000101	10000001	00001101	11101100	00111010	10000100	00100001	00100010
18	11001101	10000111	00001001	11101100	00111010	10001110	00100010	00000111
19	11001101	10000101	00001101	11101100	00111010	10001110	00111011	00000110
20	11100101	10000011	00001101	11101100	00111010	10001110	01110010	00110111
21	11100101	10000001	00001001	11101100	00111010	10001110	01110011	00010110
22	11100101	10000011	00001101	11101100	00111010	10001110	01110001	00110110
23	01100101	10010101	00001101	11101100	00111010	10000100	00101001	00010110
24	01100101	10010011	00001001	11101100	00111010	10000110	01101010	00110011
25	01100101	10010001	00001101	11101100	00111010	10000110	01100011	00010010
26	11100111	10010111	00001101	11101100	00111011	10011100	00100100	00110011

Pepsi – Attack (2)

[0x100]

```
        07 0757      07 0757
EUR     54 3210      54 3210
                      i  iii
------------------------------------
16      00 0001      01 0110
17      00 0000      01 0111
18      00 1111      01 1000
19      00 1110      01 1001
20      11 0111      10 0000
21      11 0110      10 0001
21      11 0110      10 0001
23      11 0100      10 0011
24      11 0011      10 0100
25      11 0010      10 0101
26      11 0001      10 0110
```

→ BCD Code

- 10 EUR
- 1 EUR
- counter // 256
- counter % 256
- checksum // 256
- checksum % 256

```
byte:   00000000 11111111 22222222 33333333 44444444 55555555 66666666 77777777
bit:    76543210 76543210 76543210 76543210 76543210 76543210 76543210 76543210
-------------------------------------------------------------------------------
xor A:    0 0      111 00        1    101        1      111   0            1 1 1
xor B:    1 0      101 11        1    001        0      101   0            0 1 0
```

Chocolate - Attack

= 1.1 EUR

10 EUR
1 EUR
0.1 EUR
counter // 256
counter % 256
checksum // 256
checksum % 256

```
byte:   00000000 11111111 22222222 33333333 44444444 55555555 66666666 77777777
bit:    76543210 76543210 76543210 76543210 76543210 76543210 76543210 76543210
-------------------------------------------------------------------------------
xor A:    0 0    1111 00     0   1   101       1   1      111  0         0 1 1 1
xor B:    1 0    0101 11     1   1   001       1   0      101  0         1 0 1 0
```

Salad - Attack

$$= m \cdot 0.01 \, \frac{\text{EUR}}{\text{g}}$$

- 10 EUR
- 1 EUR
- 0.10 EUR
- 0.01 EUR
- counter // 256
- counter % 256
- checksum // 256
- checksum % 256

```
byte:    00000000 11111111 22222222 33333333 44444444 55555555 66666666 77777777
bit:     76543210 76543210 76543210 76543210 76543210 76543210 76543210 76543210
--------------------------------------------------------------------------------
xor A:     0 01   1111000     0   1   101       1   1      1110 0         0 1 111
xor B:     1 00   0101111     1   1   001       1   0      1011 0         1 0 100
```

Rich man's - Attack

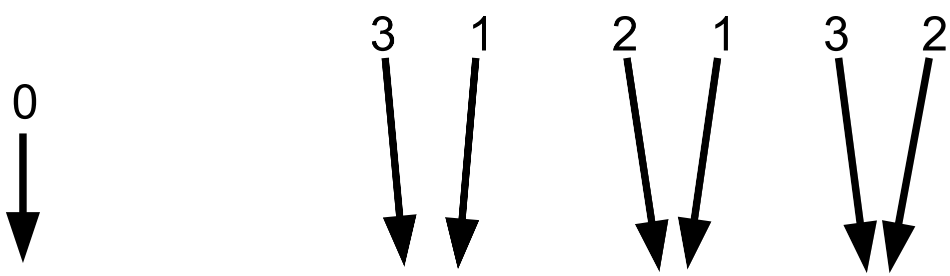
$$\sum_{1}^{10} 10\,\mathrm{EUR} = 100\ \mathrm{EUR}$$

100	EUR
10	EUR
1	EUR
0.10	EUR
0.01	EUR
counter // 256	
counter % 256	
checksum // 256	
checksum % 256	

```
byte:     00000000 11111111 22222222 33333333 44444444 55555555 66666666 77777777
bit:      76543210 76543210 76543210 76543210 76543210 76543210 76543210 76543210
--------------------------------------------------------------------------------
xor A:      0 01 1 1111000     0   1  1101      1   1      1110 0        00 1 111
xor B:      1 00 1 0101111     1   1  0001      1   0      1011 0        11 0 100
```

Nostradamus - Attack

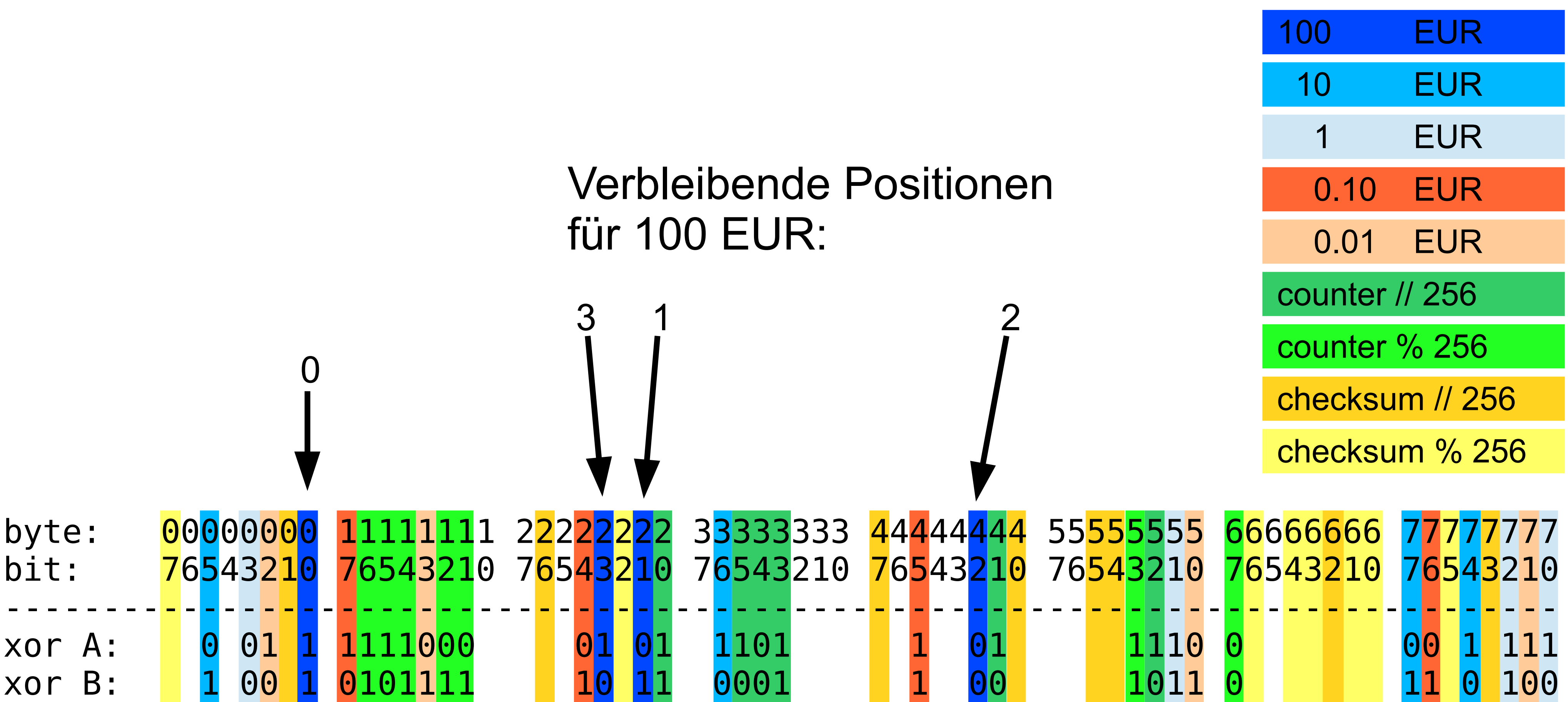
Verbleibende Positionen
für 100 EUR:




```
byte:    00000000 11111111 22222222 33333333 44444444 55555555 66666666 77777777
bit:     76543210 76543210 76543210 76543210 76543210 76543210 76543210 76543210
-----------------------------------------------------------------------------------
xor A:     0 01 1 1111000     0   1  1101      1   10     1110 0        00 1 111
xor B:     1 00 1 0101111     1   1  0001      1   0      1011 0        11 0 100
```

100	EUR
10	EUR
1	EUR
0.10	EUR
0.01	EUR
counter // 256	
counter % 256	
checksum // 256	
checksum % 256	

Nostradamus - Attack



Nostradamus - Attack

Maximale erlaubtes Guthaben: 150 EUR

100	EUR
10	EUR
1	EUR
0.10	EUR
0.01	EUR
counter // 256	
counter % 256	
checksum // 256	
checksum % 256	

Verbleibende Positionen
für 100 EUR:

3 1 2

```
byte:     00000000 11111111 22222222 33333333 44444444 55555555 66666666 77777777
bit:      76543210 76543210 76543210 76543210 76543210 76543210 76543210 76543210
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
xor A:      0 01 1 1111000     01 01  1101      1  01      1110 0        00 1 111
xor B:      1 00 1 0101111     10 11  0001      1  00      1011 0        11 0 100
```

Checksum

+

Checksum

checksum // 256

checksum % 256

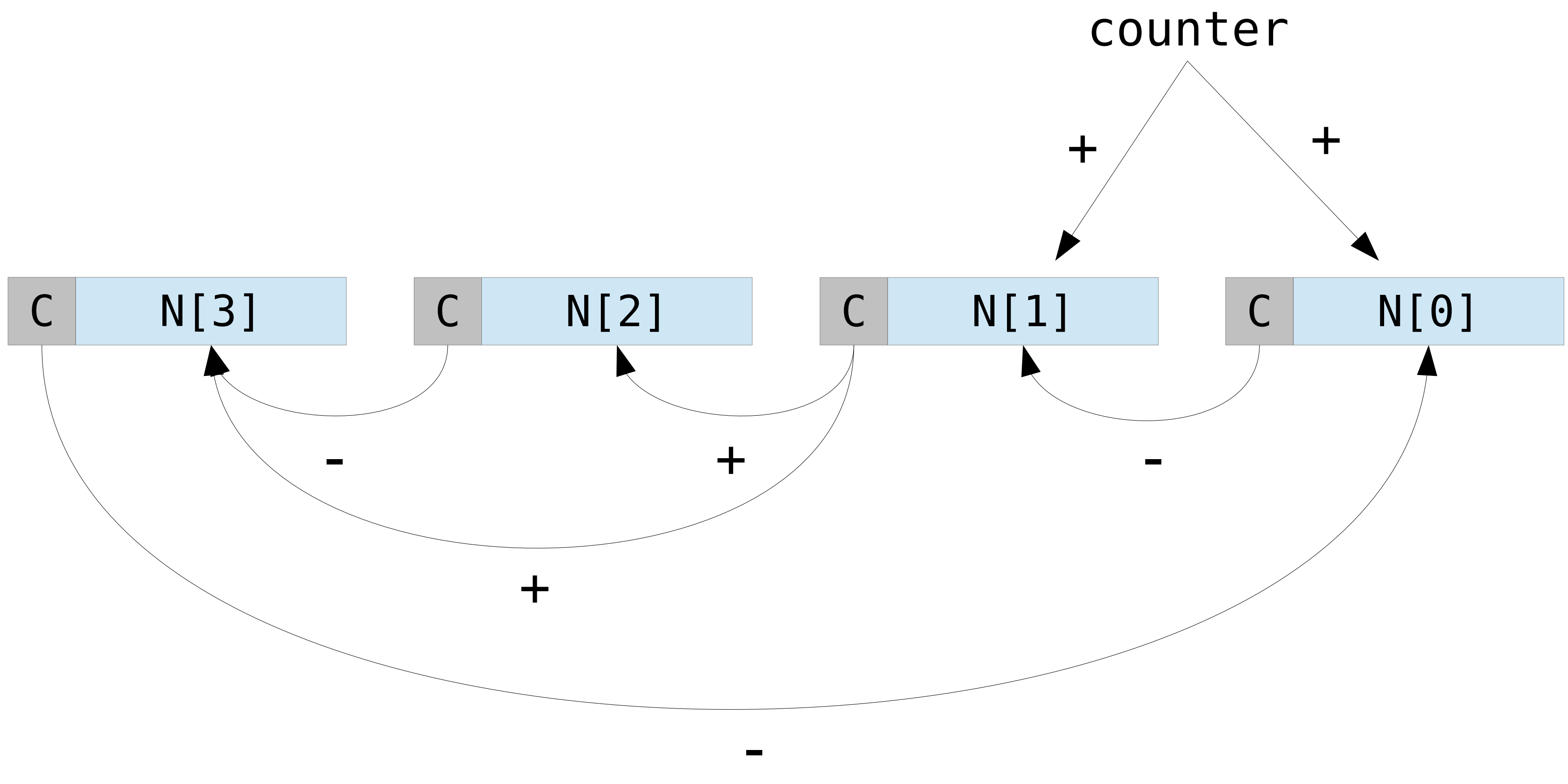
Staring at bits ...

```
byte:   00000000 11111111 22222222 33333333 44444444 55555555 66666666 77777777
bit:    76543210 76543210 76543210 76543210 76543210 76543210 76543210 76543210
```

```
byte:          4255  7604  0676  6266
bit:           7654  3210  7654  3210
-----------------------------------------------------
xor 0x100:     i ii  ii     i i
xor 0x110:      i i  iiii   iii   iii
-----------------------------------------------------
addr           N[3]  N[2]  N[1]  N[0]      counter
-----------------------------------------------------
0x100          1010  1000  1011  1010      001110111001
0x110          1010  1000  1100  1011      001110111010
0x100          1010  1000  1101  1100      001110111011
0x110          1010  1000  1110  1101      001110111100
0x100          1010  1000  1111  1110      001110111101
0x110          1011  1001  0000  1111      001110111110
0x100          1011  1001  0000  0000      001110111111
0x110          1011  1001  0001  0001      001111000000
0x100          1011  1001  0010  0010      001111000001
0x110          1011  1001  0011  0011      001111000010
0x100          1011  1001  0100  0100      001111000011
0x110          1011  1001  0101  0101      001111000100
```

chk-cnt.txt

Checksum



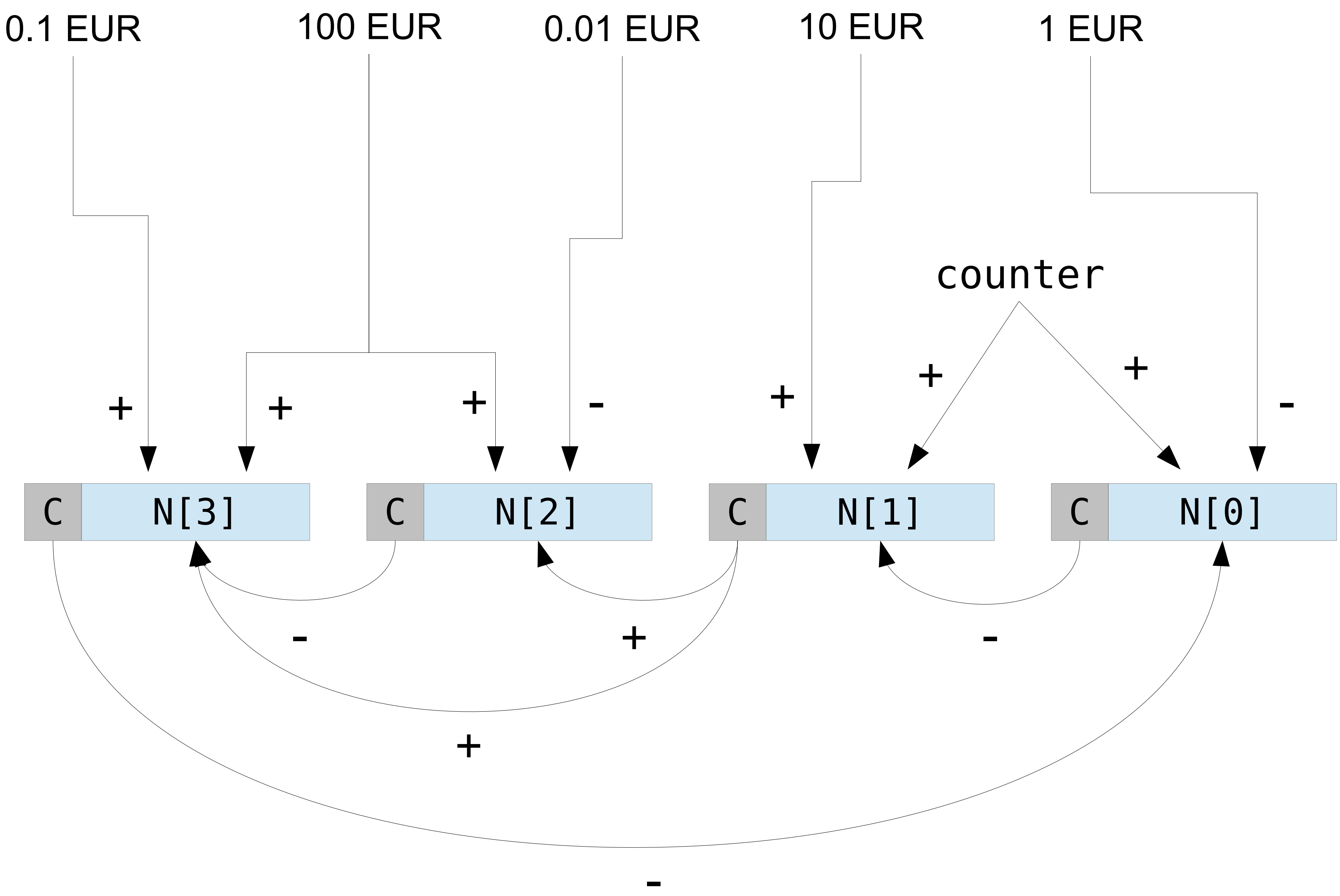
Checksum

```
def checksum_inc_cnt( chk ):

    n = [chk>>i*4&0xF for i in range(4)]

    n[0] += 1
    n[1] += 1

    n[1] -= n[0]>>4
    n[2] += n[1]>>4
    n[3] += n[1]>>4
    n[3] -= n[2]>>4
    n[0] -= n[3]>>4

    return sum( [(n[i]&0xF)<<i*4 for i in range(4)])
```

Checksum

Credit?

Checksum



Checksum

```
def checksum_adjust_credit(
      chk, eur100, eur10, eur1, eur01, eur001 ):

    n = [chk>>i*4&0xF for i in range(4)]

    n[2] -= eur001
    n[3] += eur01
    n[0] -= eur1
    n[1] += eur10
    n[2] += eur100
    n[3] += eur100

    n[1] -= n[0]>>4
    n[2] += n[1]>>4
    n[3] += n[1]>>4
    n[3] -= n[2]>>4
    n[0] -= n[3]>>4

    chk = sum( [(n[i]&0xF)<<i*4 for i in range(4)])
    return chk
```

Checksum

Bekannt:
Änderung der Checksumme bei Änderung von (counter, credit).
Gültige Checksumme für mind. ein (counter, credit).

z.B: counter = credit = 0

```
chk0 = 0b0100010111010100  # =17876
```

→ **Iterativ:**
Erzeugung der Checksumme für beliebige (counter, credit).

Unbekannt:
Geschlossener algebraischer Ausdruck für Checksumme?

$$chk = f\left(counter, credit\right)$$

Unbekannte Bitpositionen

```
byte:   3 4   5 5 6 4   2 0 2 0   4 3 3 3
bit:    7 6   7 6 5 4   7 6 5 4   3 2 1 0
```

Symmetrie:

- 2 weitere Counter Bits: <15:14>
- 4 weitere Credit Bits: 1000 EUR

Bleibt noch 1 unbekanntes Byte:

- 100k / 10k
- 0.001 / 0.0001

Unbekannte xor-Masken:

Unterstelle, daß alle unbekannten Position gleich 0.

???
100 EUR
10 EUR
1 EUR
0.10 EUR
0.01 EUR
counter // 256
counter % 256
checksum // 256
checksum % 256

```
byte:    00000000 11111111 22222222 33333333 44444444 55555555 66666666 77777777
bit:     76543210 76543210 76543210 76543210 76543210 76543210 76543210 76543210
-------------------------------------------------------------------------------
xor A:   01000101 11110001 00001001 11101100 10111010 10111110 01110100 00011111
xor B:   01110011 01011111 11110111 00001100 00111001 01011011 01010111 11101100
```

Zusammenfassung

- Giro Vend: MIFARE Classic
- Standardschlüssel für unbenutzte Sektoren
  → Anfällig für *Nested Authentication Attack*
- Guthaben auf Karte gespeichert
- Guthabenabgleich mit Datenbank
- Offline Lade/Verkaufsstationen → Kein Abgleich möglich.
- Guthaben gespeichert abwechselnd an Addr. 0x100 / 0x110.
- Counter für Kartenlesevorgänge
- <u>Verschlüsselung:</u>
  - xor-Schlüssel (spezifisch für 0x100 / 0x110)
  - Permutation von Bits (konst. pos. innerhalb eines Bytes)
  - Checksumme

```
Errorcodes:
--------------------
E.66  cnt == 0
E.76  credit > max
E.7E  cnt >= 2**14-1
E.49  B-code wrong
E.40  checksum wrong
```

100	EUR
10	EUR
1	EUR
0.10	EUR
0.01	EUR
counter // 256	
counter % 256	
checksum // 256	
checksum % 256	

```
byte:    00000000 11111111 22222222 33333333 44444444 55555555 66666666 77777777
bit:     76543210 76543210 76543210 76543210 76543210 76543210 76543210 76543210
-------------------------------------------------------------------------------
xor A:   01000101 11110001 00001001 11101100 10111010 10111110 01110100 00011111
xor B:   01110011 01011111 11110111 00001100 00111001 01011011 01010111 11101100
```