



Attacks On And With API: PIN Recovery Attacks

**Masaryk University in Brno
Faculty of Informatics**

Jan Krhovják
Daniel Cvrček

Roadmap

- Introduction
 - Basic terminology
 - Insufficient checking of function parameters
- Decimalisation table attacks
 - Techniques of PIN generation and verification
 - Attacks utilising known PINs
 - Extended attack without known PINs
- ANSI X9.8 attacks
 - PIN-block formats
 - Attacking PAN with translation&verification functions
 - Attacking PIN translation functions
 - Collision attack
- Conclusion

Introduction

- Basic terminology
 - Hardware Security Module (HSM)
 - Example: IBM 4758 (depicted below)
 - Host device
 - Application Programming Interface (API)
 - Attack
 - PIN Recovery Attacks
 - Clear PIN-block (CPB)
 - Encrypted PIN-block (EPB)
 - Personal Account Number (PAN)
- Insufficient checking of function parameters



PIN Generation and Verification

- Techniques of PIN generation and verification
 - IBM 3624 and IBM 3624 Offset
 - Based on validation data (e.g. account no. – PAN)
 - Validation data encrypted with *PIN derivation key*
 - The result truncated, decimalised => PIN

 - IBM 3624 Offset – decimalised result called IPIN (Intermediate PIN)
 - Customer selects PIN,
$$\text{Offset} = \text{PIN} - \text{IPIN} \text{ (digits mod 10)}$$
 - Verification process is the same
 - result is compared with decrypted EPB (encrypted PIN from cash-machine)

PIN Verification Function

- Simplified example of verification function and its parameters:
 1. PIN (CPB) encryption/decryption key
 2. PIN derivation key – for PIN generation process
 3. PIN-block format
 4. validation data – for PIN extraction from EPB (e.g. PAN)
 5. encrypted PIN-block
 6. verification method
 7. data array – contains decimalisation table, validation data and offset
- Clear PIN is not allowed to be a parameter of verification function!

PIN Verification – IBM 3624 Offset

- Inputs – (4-digit PIN)
 - PIN in EPB is 7216 (delivered by ATM)
 - Public offset (typically on card) – 4344
 - Decimalisation table – 0123 4567 8901 2789
 - Personal Account Number (PAN) is
4556 2385 7753 2239
- Verification process
 - PAN is encrypted => 3F7C 2201 00CA 8AB3
 - Truncated to four digits => 3F7C
 - Decimalised according to the table => 3972
 - Added offset 4344, generated PIN => 7216
 - Decrypt EPB and compare with the correct PIN

Decimalisation Table Attacks I

- Attacks utilising known PINs
 - Assume four-digit PINs and offset 0000
 - If decim. table (DT) is 0000 0000 0000 0000
generated PIN is always 0000
 - PIN generation function with *zero* DT outputs EPB with PIN 0000
 - Let $D_{orig} = 0123\ 4567\ 8901\ 2345$ is original DT
 - D_i is a *zero* DT with "1" where D_{orig} has i
e.g. $D_5 = 0000\ 0100\ 0000\ 0001$
 - The attacker calls 10x verification function with EPB of 0000 PIN and with D_0 to D_9
 - If i is not in PIN, the "1" will not be used and verification against 0000 will be successful

Decimalisation Table Attacks II

- Results
 - All PIN digits are discovered
 - PIN space reduced from 10^4 to 36 (worst case)
- Extended attack without known PINs
 - Assume, that we obtain customers EPB with correct PIN
 - D_i are DTs containing $i-1$ on positions, where D_{orig} has i e.g. $D_5 = 0123 \ 4\mathbf{4}67 \ 8901 \ 234\mathbf{4}$
 - Verification function is called with intercepted EPB and D_i
 - Position of PIN digits is discovered by using *offset* with digits incremented individually by "1"
 - Bold "4" changes to "5"

DT Attacks – Example

- Let PIN in EPB be 1492, offset is 1234
- We want to find position of "2"
- Verification function with D_2 results in $1491 \neq 1492 \Rightarrow$ fails
- Offsets 2234, 1334, 1244, 1235 increment resulting generated PIN (2491, 1591, ...)
- Eventually the verification is successful with the last offset \Rightarrow 2 is the last digit

- To determine four-digit PIN with different digits is needed at most 6 calls of verification function

Clear PIN Blocks

- Code Book Attacks and PIN-block formats

- => clear PIN blocks (CPB)

p – PIN digit
r – random digit
x – arbitrary,
all the same
F – 0xF digit

- ECI-2 format for 4 digits PINs

- ECI-2 CPB = pppprrrrrrrrrrrrrrrr

- Visa-3 format for 4–12 digits PINs

- Visa-3 CPB = ppppFxxxxxxxxxxxx

- **ANSI X9.8 format for 4–12 digits PINs**

- $P_1 = ZlppppffffffffffF$
- $P_2 = ZZZZaaaaaaaaaaaaa$
- ANSI X9.8 CPB = $P_1 \text{ xor } P_2$

Z – 0x0 digit
l – PIN length
f – either “p” or “F”
a – PAN digit

ANSI X9.8 Attacks I

- Attacking PAN with translation & verification functions – input parameters (key K, EPB, PAN)
 - Functions decrypt EPB & extract PIN
 $CPB \text{ xor } P_2 = 04ppppFFFFFFFFFFFF \Rightarrow PIN = pppp$
 - Extraction tests PIN digits to be 0–9!
 - If a digit of PAN is modified by x
 - $P_2' = P_2 \text{ xor } 0000x000000000000$
 - $CPB \text{ xor } P_2' = 04ppppFFFFFFFFFFFF \text{ xor } 0000x000000000000$
it means that $PIN = pppp \text{ xor } 00x0$
 - If $p \text{ xor } x < 10$ function ends successfully, otherwise function fails

ANSI X9.8 Attacks II

- The sequence of (un)successful function calls can be used by attacker to identify p as a digit from set $\{p, p \text{ xor } 1\}$
 - For example if PIN digit is 8 or 9, then this sequence will be PFFFFFFFFPPPPPPP, where P is PASS, F is FAIL and x is incremented from 0 to 15
-
- Only last two PIN digits can be attacked
 - PIN space is reduced from 10^4 to 400
 - This attack can be extended to all PIN digits

ANSI X9.8 Attacks III

- Attack against PIN translation functions
 - Input/output PIN-block format can be modified
 - Consider ANSI X9.8 EPB with null PAN (wlog)
 - Attacker specifies input format as VISA-3 and output as ANSI X9.8
 - PIN is then extracted from `04ppppFFFFFFFFFFFF` as `04pppp`
 - `04pppp` is formatted into ANSI X9.8 CPB as `0604ppppFFFFFFFF` and encrypted
 - Attacker has EPB with six-digit PIN and can use previous attack to determine all 4 digits of original PIN
- PIN space is reduced from 10^4 to 16

ANSI X9.8 Attacks IV

- PIN can be also determined exactly
- The attacker needs to be able to modify PAN
 - This is impossible if input format is Visa-3
 - PAN modification must be done earlier (in EPB)
- Let's modify second digit of PAN by x
 - Input format is VISA-3 and output ANSI X9.8
 - PIN is decrypted from ANSI X9.8 EPB and extracted as $04pppp \text{ xor } 00000x$
 - If $x = p \text{ xor } F$ (i.e. $x \text{ xor } p = F$) then PIN is extracted as $04ppp$ and formatted into ANSI X9.8
 - This can be detected by/during translation back to VISA-3 format EPB

ANSI X9.8 Attacks – Collision Attack

- Assuming well designed API (e.g. DT is fixed)
- Attack allows to partially identify last two PIN digits
 - Basic idea (simple example with one-digit PIN&PAN)

PAN	PIN	xor	EPB	PAN	PIN	xor	EPB
0	0	0	21A0	7	0	7	2F2C
0	1	1	73D2	7	1	6	345A
0	2	2	536A	7	2	5	0321
0	3	3	FA2A	7	3	4	FF3A
0	4	4	FF3A	7	4	3	FA2A
0	5	5	0321	7	5	2	536A
0	6	6	345A	7	6	1	73D2
0	7	7	2F2C	7	7	0	21A0
0	8	8	4D0D	7	8	F	AC42
0	9	9	21CC	7	9	E	9A91

- Attacker knows for each PAN only the set of EPBs

ANSI X9.8 Attacks – Collision Attack

- Looking collisions in output of PIN generation function
- Remember PIN generation & ANSI X9.8 CPB
- Formalizing PIN generation function
 - So $EPB = Encrypt(Pad(U_a, U_b, U_c, U_d))$, where
$$U_a = (F_a(e, f) + a) \bmod 10$$
$$U_b = (F_b(e, f) + b) \bmod 10$$
$$U_c = ((F_c(e, f) + c) \bmod 10) \text{ xor } e$$
$$U_d = ((F_d(e, f) + d) \bmod 10) \text{ xor } f$$
 - e, f are first two digits of PAN
 - $F_x(e, f)$ is respective digit of IPIN
 - a, b, c, d are digits of offset

ANSI X9.8 Attacks – Collision Attack

- The whole function is $Gen(a, b, c, d, e, f)$
- Desired IPIN digits are $F_c(e, f)$ and $F_d(e, f)$
 - To get $F_c(e, f)$, the attacker must choose a fixed value $DELTA$
 - She modifies offset and to get collisions:
$$Gen(a, b, c, d, e, f) = Gen(a', b', c', d', e \text{ xor } DELTA, f)$$
 - When a collision is found: $U_c = U_{c'}$, and $DELTA = ((F_c(e, f) + c) \bmod 10) \text{ xor } ((F_c(e \text{ xor } DELTA, f) + c) \bmod 10)$
 - Certain $DELTA$ can be obtained only by a few combinations (e.g $F=6 \text{ xor } 9$ or $7 \text{ xor } 8$)
$$\Rightarrow (F_c(e, f) + c) \bmod 10 \text{ is } 6, 7, 8 \text{ or } 9$$
 - Next collision for $DELTA=7$ leaves only 6 and 7
 - Because c is known, we simply get $F_c(e, f)$

Conclusion

- The security of current generation banking APIs is really bad with respect to **insider attacks**
- Function parameters can be arbitrarily changed – **controls not sufficient**
- PIN-block formats do not ensure sufficient **entropy**
- Number of standards implemented ensures **interoperability** but also **causes errors**

- Can asymmetric cryptography help? See an attack on Chrysalis Luna CA3 module!

- Other attacks 😊
 - Master's thesis (in czech):
http://www.fi.muni.cz/~xkrhovj/apinf/sdipr/DP_upravena_v1.pdf
 - Mike Bond's research:
<http://www.cl.cam.ac.uk/~mkb23/research.html>
 - Jolyon Clulow's research:
<http://www.cl.cam.ac.uk/~jc407/>