Útoky na a přes API: PIN Recovery Attacks

Jan Krhovják
xkrhovj@fi.muni.cz

Fakulta informatiky
Masarykova univerzita v Brně
Brno, Česká republika

Daniel Cvrček
Daniel.Cvrcek@cl.cam.ac.uk

Computer Laboratory
University of Cambridge
United Kingdom

Abstrakt

Bezpečná zařízení byla navržena především pro zajištění důvěrnosti citlivých kryptografických dat. Data samotná neopouští příslušné zařízení, a kryptografické operace, při kterých jsou data vyžadována, jsou tímto zařízením prováděny. Aby libovolná aplikace mohla bezpečné zařízení používat, je definováno aplikační programovací rozhraní (API). Tento příspěvek pojednává o útocích na a přes tato API a zaměřuje se na útoky vedoucí k získání PINů – *PIN Recovery Attacks*. Hlavním cílem je vytvořit přehled typických útoků. V první části se věnujeme útokům, které využívají nedostatečné kontroly parametrů funkcí. Druhá část obsahuje útoky na bankovní API zneužívající špatného návrhu formátů PIN-bloků, do nichž jsou PINy před zašifrováním formátovány.

Klíčová slova: ANSI X9.8, decimalizační tabulka, PIN, útoky vedoucí k získání PINů.

1 Úvod

Pro realizaci bezpečné komunikace v elektronických aplikacích je zajištění správnosti kryptografických funkcí (integrita kryptografických funkcí a důvěrnost příslušných kryptografických klíčů) bezpodmínečnou nutností. U běžně používaných výpočetních systémů toho lze dosáhnout jen velmi obtížně a důvodem jsou dva okruhy problémů. Především je to hardware, který je typicky zcela fyzicky nezabezpečen a umožňuje kompromitaci systému, pokud k němu útočník získá fyzický přístup. Dále je to obrovský rozsah programového vybavení, což implikuje velké množství chyb, které mohou (bez ohledu na jejich původ) ovlivnit funkčnost libovolné části tohoto programového vybavení. Snaha odstranit tyto problémy je hlavním důvodem pro návrh a používání hardwarových bezpečnostních zařízení (HSM – Hardware Security Module). Ta mají fungovat jako důvěryhodná výpočetní báze (TCB – Trusted Computing Base), která poskytuje veškeré potřebné kryptografické operace.

Aplikační oblastí, jež odstartovala vývoj HSM, bylo bankovnictví. Hlavní skupinou aplikací pak elektronické transakce v bankovních sítích (např. VISA, MasterCard, American Express) zahrnující komunikaci jednotlivých bank a komunikaci s jejich peněžními bankomaty (ATM). Tyto rozsáhlé ATM sítě[1], do kterých je v dnešní době sdružováno stále více bank, umožňují zákazníkům provádět finanční transakce téměř z kteréhokoliv místa na světě. Jednotlivé banky si však nedůvěřují a HSM byly zvoleny jako mechanizmus, který umožní efektivně provádět bankovní transakce. Mezi problémy, které bylo třeba vyřešit, patří zabezpečení důvěrného přenosu dat či bezpečná správa kryptografických klíčů a jiných citlivých dat (např. PINů) tak, aby se předešlo podvodům ze strany klientů i zaměstnanců bank.

HSM poskytují prostředí pro bezpečné provádění citlivých operací. Přístup k těmto operacím je možný pouze přes předem definované softwarové rozhraní – tzv. aplikační programovací rozhraní (API).

Předpokládalo se, že takováto abstrakce citlivých operací a přístup aplikací pouze k „hlavičkám funkcí“ poskytne dostatečné záruky bezpečnosti. Bohužel ekonomie postupně převážila bezpečnost. Snaha o co nejflexibilnější návrh HSM, a zpětná (i současná) podpora konkurenčních standardů a norem, zapříčinila takový nárůst ve složitosti programového vybavení HSM, že bezpečnost citlivých dat uvnitř HSM již nelze dále zaručit.

Důkazem je stále narůstající počet útoků na API různých HSM, jejichž přehledem a analýzou se budeme v tomto příspěvku zabývat.

[1] V tomto příspěvku bude pojem *ATM síť* značit vždy síť peněžních bankomatů, a význam zkratky ATM je tedy nutno interpretovat nikoliv jako Asynchronous Transfer Mode, ale jako Automatic Teller Machine.

2 Nedostatečná kontrola parametrů funkcí

U bankovních HSM je jednou z nejcitlivějších oblastí práce s PINy bankovních karet – tím je také vymezena pravděpodobně nejvíce používaná část bankovního API. Ověřování PINů bylo hlavní motivací pro zavádění HSM v bankovnictví, protože umožňovaly klientům bank používat své bankovní karty kdekoliv po světě. Je pozoruhodné, jak špatně jsou chráněny citlivé parametry právě funkcí zajišťujících operace s PINy.

Útoky vedoucí k získání PINů – *PIN Recovery Attacks* tvoří jednu z největších tříd útoků na současné HSM. Tyto útoky demonstrují techniky, s jejichž pomocí lze ze zašifrovaného PIN-bloku bez znalosti klíče získat hodnotu PINu. K jejich provedení stačí v některých případech pouze přístup k API daného zařízení, které je součástí bankovního systému (s řádně nainstalovanými klíči) a jeden zašifrovaný PIN-blok. Kromě toho, že jsou útoky extrémně rychlé a snadno implementovatelné, lze je většinou aplikovat i na více druhů běžně používaných API[2], čímž postihnou mnohem více HSM. V tomto příspěvku vycházíme především z [2, 3, 4].

Na úvod bychom jen rádi zmínili způsob práce s PINy. PINy jsou formátovány do tzv. PIN-bloků – CPB (clear PIN block), což jsou 8bajtové struktury. Tyto se po zašifrování nazývají EPB (encrypted PIN block). Tuto terminologii budeme dále používat.

3 Útoky na decimalizační tabulku

Typická verifikační funkce pro ověřování PINu na základě validačních dat (obvykle je to číslo účtu) vygeneruje PIN a porovná jej s PINem získaným z EPB[3]. Bezpečnostním problémem těchto funkcí jsou metody generování PINů. Ty vycházejí ze starých metod používaných bankomatem IBM 3624 (vyráběným od poloviny sedmdesátých let). Jedním z parametrů těchto funkcí je decimalizační tabulka umožňující převod čísel ze šestnáctkové (formát výstupu šifrovací funkce) na desítkovou soustavu (bankomaty používají numerickou klávesnici). S touto převodní tabulkou lze dělat zajímavé věci. Nejdříve se ale podívejme, jak se PIN generuje.

3.1 Techniky generování a verifikace PINů

Existuje mnoho metod používaných pro generování a verifikaci PINů. Typickými příklady jsou metody IBM 3624 a IBM 3624 Offset. IBM 3624 generování PINů je založeno na validačních datech (obvykle je to číslo účtu – PAN), která jsou zašifrována PIN generujícím klíčem a požadovaný počet číslic je převeden do desítkové soustavy (decimalizován) a zvolen jako PIN.

Verifikace pak probíhá analogicky, avšak PIN generující klíč se nazývá PIN verifikující klíč a vypočítaný PIN je nakonec porovnán s PINem získaným z EPB. Metoda IBM 3624 Offset navíc použitím offsetů umožňuje změnu PINu zákazníkem. Generování zde probíhá stejně jako v předchozím případě, ale výsledek se nazývá IPIN (intermediate PIN) a offset je získán odečtením IPINu od zvoleného PINu. Všechny operace sčítání a odčítání se provádějí na jednotlivých číslicích (modulo 10) a k decimalizaci se používá decimalizační tabulka. Názorný příklad výpočtu zákazníkem zvoleného PINu při verifikaci metodou IBM 3624 Offset je uveden níže.

Číslo účtu je reprezentováno pomocí ASCII číslic v dekadické soustavě a poté interpretováno jako hexadecimální vstup blokové šifry DES (resp. 3DES). Po zašifrování PIN generujícím klíčem je výsledek opět převeden do hexadecimální soustavy a zkrácen na první čtyři cifry. Ty však mohou obsahovat hodnoty 'A'–'F', které nejsou obsaženy na numerické klávesnici bankomatu, a proto jsou pomocí decimalizační tabulky převedeny na číslice dekadické soustavy. K výsledku je přičten offset, čímž se získá hodnota pro porovnání s PINem, který byl zadán bankomatu (viz obr. 1).

```
          číslo účtu (PAN): 4556 2385 7753 2239
           Zašifrovaný PAN: 3F7C 2201 00CA 8AB3
Zkrácený zašifrovaný PAN: 3F7C
                    IPIN: 3972            decimalizační tabulka
Veřejně přístupný offset: 4344          vstup 0123456789ABCDEF
              Zvolený PIN: 7216         výstup 0123456789012789
```

Obr.1: Postup výpočtu PINu metodou IBM3624 Offset.

[2] Například IBM CCA API, HP(dříve Compaq)-Atalla API a Thales-Zaxus-Racal API.

[3] Vstupem funkce nemůže být přímo PIN, protože bankovní programátor by mohl snadno vyzkoušet všechny PINy (10^4 možností) a určit hodnotu PINu v EPB.

Tyto metody byly použity v nejstarších typech bankomatů, a jsou stále rozšířeny a implementovány i v nových HSM (podporuje je například i CCA API v IBM 4758). Jedinou změnou je, že validační data byla původně uložena společně s offsetem na kartě (na magnetickém proužku) a jediné, co musel bankomat chránit, byl PIN generující klíč. Dnes již verifikace PINů neprobíhá v bankomatu, ale ve vydávající bance, takže EPB již není na kartě potřeba a PIN generující klíč je uložen bezpečně v bance.

V dalším textu bude decimalizační tabulka zadávána jen jako posloupnost výstupních hodnot (viz obr. 1), s tím, že příslušné vstupní hodnoty budou implicitně '0'–'F'. Bez újmy na obecnosti budeme také (pokud nestanovíme jinak) pracovat pouze se čtyřmístnými PINy.

3.2 Útoky využívající známých zašifrovaných PINů

Než se pustíme do popisu útoků, jen krátce uvedeme základní možné parametry funkce pro verifikaci PINů (dále jen VerifyPIN) a shrneme jak pracuje:

1. klíč pro šifrování PINu;
2. klíč pro verifikaci PINu – pro výpočet IPIN;
3. formát PIN bloku;
4. validační data – pro získání PINu z EPB, obvykle číslo účtu (PAN);
5. zašifrovaný PIN – EPB (encrypted PIN block);
6. verifikační metoda;
7. data – pole obsahující decimalizační tabulku, validační data a offset.

Verifikace pak probíhá tak, že EPB (5) se dešifruje klíčem (1) a podle formátu (3) se z něj extrahuje PIN (v případě některých formátů ještě pomocí validačních dat (4)). Současně se vezmou validační data (7), která se zašifrují klíčem (2) a pomocí decimalizační tabulky a offsetu (7) se vytvoří PIN (tento postup se v závislosti na použité verifikační metodě (6) může v praxi mírně lišit). Oba PINy se pak porovnají.

Nejjednodušším útokem je využití decimalizační tabulky ke zjištění číslic, které se vyskytují v PINu. Nastavíme-li například decimalizační tabulku na samé nuly, bude PIN vygenerovaný metodou IBM 3624 roven čtyřem nulám (všechny číslice jsou decimalizovány na '0').

Tímto trikem můžeme pomocí funkce generující zašifrované PINy (pro jednoduchost dále jen GenEncPIN, která v bankovním API vždy existuje a má podobné vstupní parametry jako VerifyPIN) získat EPB obsahující PIN `0000`. Jestliže nyní při verifikaci použijeme jako parametry tento EPB a „nulovou“ decimalizační tabulku, proběhne verifikace úspěšně (tj. z EPB dešifrovaný PIN `0000` je roven vygenerovanému PINu).

Nechť D_{orig} je korektní decimalizační tabulka a D_i jsou nové binární decimalizační tabulky, které mají jedničku právě na těch pozicích, kde D_{orig} měla hodnotu *i*. Je-li například D_{orig} = `0123 4567 8901 2345`, pak D_5 = `0000 0100 0000 0001` a D_9 = `0000 0000 0100 0000`. Nyní stačí, aby útočník pro *i* od 0 do 9 zavolal VerifyPIN s EPB nulového PINu a decimalizační tabulkou D_i. Není-li v hledaném PINu číslice *i* obsažena, změna v decimalizační tabulce se neprojeví a verifikace proběhne úspěšně. V opačném případě je *i* jedna z hledaných číslic PINu. K určení všech číslic vyskytujících se v PINu je potřeba provést verifikaci maximálně desetkrát.

Celkově se tak počet možných PINů omezí z 10 000 na (v případě čtyřmístného PINu složeného z tří různých číslic) nejvýše $_3C_2 \cdot _4C_2 \cdot _2C_1 \cdot _1C_1 = 3 \cdot 6 \cdot 2 \cdot 1 = 36$, kde první kombinační číslo je počet různých možností dvou výskytů stejné číslice, a zbylá tři vyjadřují možné rozložení číslic (dvě místa ze čtyř pozic pro první číslici a jedno místo ze dvou/jedné pozice pro druhou a třetí číslici). Pro čtyři různé číslice je to $1 \cdot _4C_1 \cdot _3C_1 \cdot _2C_1 \cdot _1C_1 = 1 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 24$ a pro dvě $2 \cdot _4C_3 \cdot _1C_1 + 1 \cdot _4C_2 \cdot _2C_2 = 2 \cdot 4 \cdot 1 + 1 \cdot 6 \cdot 1 = 14$.

Jiná varianta útoku je efektivnější a umožňuje přesně určit pořadí číslic v PINu, je ovšem třeba získat EPB pro pět známých PINů: `0000`, `0001`, `0010`, `0100` a `1000`. Protože bankovní systémy většinou neumožňují vkládání nezašifrovaných PINů a metodu z předchozího útoku nelze k vygenerování všech pěti hodnot PINů použít, je nutno získat je jinou cestou. Asi nejjednodušší je zadat tyto PINy bankomatu a zachytit je zašifrované poté, co dorazí do banky[4]. Kromě zachycení komunikace s bankou je také možné EPB získat z log souborů, které jsou v bankomatu a archivovány bankou. Jestliže se samotné hledání PINu implementuje, např. pomocí

[4] To je zároveň nejjistější metoda, jak známý zašifrovaný PIN získat, protože použití funkce generující PINy bývá do značné míry omezeno (a její využití v prvním útoku bylo spíše ilustrativní).

vhodně zkonstruovaného binárního stromu, který určí jakou decimalizační tabulku použít v dalším kroku, je možné nalézt PIN nejhůře na 24 pokusů, ale průměrně již na 15 pokusů [2].

3.3 Útok bez známého zašifrovaného PINu

Nutnou podmínkou předchozího útoku byla znalost EPB pro vybrané PINy. Následující útoky již toto nevyžadují. Předpokládejme, že se nám podařilo zachytit zákazníkův EPB obsahující správný PIN a že hodnota tohoto PINu ještě nikdy nebyla změněna (tj. offset je stále `0000`). Nechť D_{orig} je původní decimalizační tabulka a D_i jsou upravené decimalizační tabulky. Platí, že D_i mají hodnotu *i-1* právě na těch pozicích, kde D_{orig} měla hodnotu *i*. Je-li například D_{orig} = `0123 4567 8901 2345`, pak D_5 = `0123 44567 8901 2344` a D_9 = `0123 4567 8801 2345`. Nyní stačí, aby útočník pro každou číslici *i* zavolal VerifyPIN se zachyceným EPB, správným offsetem (tj. `0000`) a decimalizační tabulkou D_i. Tímto způsobem, podobně jako u prvního útoku, zjistí číslice obsažené v zákazníkově PINu. Jejich pořadí pak dokáže určit vhodnou manipulací s offsety.

Uvažme běžný případ, kdy má zákazníkův PIN všechny číslice odlišné. Jako příklad zvolme PIN s hodnotou `1492` a pokusme se určit pozici číslice `2`. Hodnota PINu v EPB je vždy `1492`, ale hodnotu generovaného PINu lze použitím decimalizační tabulky D_2 změnit na `1491` a verifikace pak neproběhne úspěšně. Postupným voláním VerifyPIN s offsety `1000`, `0100`, `0010`, `0001` pak zjistíme pozici, na které došlo ke změně – s offsetem `0001` se totiž hodnota PINu zvýší zpět na `1492` a verifikace proběhne úspěšně. Tím je jednoznačně určeno, která číslice PINu byla upravena. Ve skutečnosti se dokonce poslední verifikace ani nemusela provádět, protože nebyla-li hledaná číslice na předchozích třech pozicích, musela být na čtvrté.

Tímto způsobem může útočník určit pozice všech číslic, k čemuž v případě čtyřmístného PINu složeného z čtyř různých číslic potřebuje nejvýše 6 volání verifikační funkce (tři pro nalezení pozice první číslice, dvě pro nalezení pozice druhé číslice a jedno pro nalezení pozice třetí číslice). Poznamenejme, že tento útok je mírnou modifikací původního útoku z [2].

4 Útoky na ANSI X9.8

Existence PIN-bloků dává možnost definovat formátování. Důsledkem toho je, že existuje několik formátů PIN-bloků, které jsou široce používány. Pro výrobce HSM to ale znamená nutnost implementovat několik funkcí pro verifikaci PINů pro jednotlivé formáty a také překladové funkce mezi formáty. To je základem útoků, které zneužívají špatného návrhu a implementace těchto funkcí.

Předpokladem útoků je nízká entropie v PIN-blocích, jejímž důsledkem je nemožnost rozpoznat, který formát byl pro vytvoření daného PIN-bloku použit. Útočník se tak může zbavit např. validačních dat (PAN) v PIN bloku pomocí překladových funkcí (některé formáty je neobsahují), vložit validační data podle vlastního výběru (převodem zpátky), nebo se dostat k částem PIN-bloku, které by při použití jednoho formátu byly pro útočníka nedosažitelné.

4.1 Formáty PIN-bloků

Protože po zašifrování může samotný čtyřmístný PIN nabývat pouze 10 000 hodnot, hrozí nebezpečí *slovníkových útoků* (Code Book Attacks). Ty zneužívají malého počtu všech zašifrovaných a dešifrovaných PINů k snadnému vytvoření jednoznačného seznamu jejich dvojic – tzv. *kódové knihy*. Dešifrování je pak jen hledáním v tabulce. Z toho důvodu je vždy před zašifrováním PIN formátován do 8bajtové struktury (teoreticky 2^{64} možností) zvané PIN-blok, kde jsou k němu obvykle přidány náhodné doplňující hodnoty, které mají slovníkovým útokům zamezit[5]. Jako příklad uveďme formáty:

- IBM 3624;
- ISO-0 (stejný jako ANSI X9.8, VISA-1 a ECI-1);
- ISO-1 (stejný jako ECI-4);
- ISO-2;
- VISA-2, VISA-3, VISA-4.

[5] Formátování se samozřejmě provádí i pro PINy tvořené více než čtyřmi číslicemi.

Některé formáty však právě z důvodů proměnných délek PINů nepoužívají doplnění náhodnými hodnotami a snaží se vyřešit zvýšení entropie v PIN-bloku jinými způsoby.

Zaměřme se nyní na formáty VISA-3 a ANSI X9.8, které jsou nezbytné k aplikaci několika dalších útoků. Oba jsou určeny pro PINy délky 4–12 číslic, přičemž delší PIN může být z pravé strany zkrácen. U formátu VISA-3 začíná PIN vlevo a končí oddělovačem, za nimž následují doplňující číslice. Ty mají v rámci jednoho PIN-bloku vždy stejné hodnoty a ztíží[6] útočníkovi případné budování kódové knihy. Popis VISA-3 pro čtyřmístný PIN je uveden níže (viz obr. 2); použité symboly reprezentují 4bitové hexadecimální číslice.

```
p nabývá hodnot '0'-'9' a každý výskyt udává jednu číslici PINu.
F je číslice hodnoty 'F' a slouží jako oddělovač.
x je doplňující číslice, která má v rámci jednoho PIN-bloku vždy stejnou hodnotu.
VISA-3 Clear PIN Block (CPB) = ppppFxxxxxxxxxxx.
```

Obr. 2: Formátování CPB do formátu VISA-3.

U ANSI X9.8 je nejprve PIN formátován do bloku P1, PAN do bloku P2 a výsledný CPB vznikne následně jejich XORováním (operace $\oplus$). Použitím PANu se předejde[7] budování kódové knihy a jeho svázání s PINem poskytne dostatečnou ochranu i proti postupnému zkoušení falešných PANů. Obecný popis formátu ANSI X9.8 je uveden níže (viz obr. 3).

```
Z je číslice hodnoty '0'.
l je číslice nabývající hodnot '4'-'C' a udává délku PINu.
f je hodnota, která je v závislosti na délce PINu buď p nebo F[8].
a nabývá hodnot '0'-'9' a každý výskyt udává jednu číslici PANu.
P1 = Zlppppfffffffff.
P2 = ZZZZaaaaaaaaaaa.
ANSI X9.8 Clear PIN Block (CPB) = P1⊕P2.
```

Obr. 3: Formátování CPB do formátu ANSI X9.8.

4.2 Útok proti funkcím vyžadujícím PAN

Tento útok, objevený Jolyonem Clulowem [3, 4], lze aplikovat na všechny překladové a verifikační funkce, které k extrahování PINu z CPB využívají PAN. Ten je vyžadován především těmi funkcemi, které podporují formátování PINu podle ANSI X9.8. Významné vstupní parametry volaných funkcí jsou:

- Tajný klíč, kterým byl PIN-blok zašifrován (K);
- Zašifrovaný PIN-blok (EPB);
- Číslo účtu (PAN).

Základní myšlenka útoku pak spočívá ve sledování změn způsobených postupnými modifikacemi PANu. Předpokládejme, že se útočníkovi podařilo zachytit ANSI X9.8 EPB obsahující čtyřmístný PIN. Při správném volání některé z těchto funkcí je po dešifrování klíčem K tento PIN extrahován z P1=CPB$\oplus$P2= `04ppppFFFFFFFFFF` jako `pppp`. Během tohoto procesu je navíc proveden test, ověřující, zdali všechny jeho číslice nabývají hodnot '`0`' – '`9`'. Pokud tomu tak není, končí volání dané funkce s chybou. Podívejme se nyní, co se stane, zavolá-li útočník tutéž funkci s modifikovanou hodnotou první číslice PANu. Protože nyní P2' = P2$\oplus$`0000x00000000000`, je teď PIN z P1' = CPB $\oplus$ P2' = CPB $\oplus$ P2 $\oplus$ `0000x00000000000=` `04ppppFFFFFFFFFF`$\oplus$`0000x00000000000` extrahován jako `pppp`$\oplus$`00x0`. V závislosti na zvolené hodnotě `x` pak, pokud `p`$\oplus$`x<10`, funkce proběhne v pořádku, jinak skončí s chybou. Tohoto testu lze využít k vytvoření posloupnosti úspěšných a neúspěšných volání funkce, která umožní částečně identifikovat `p` jako číslici z množiny[9] {`p`, `p`$\oplus$`1`}. Protože však první čtyři číslice bloku P2 jsou vždy nuly, není možné tímto způsobem blíže identifikovat hodnoty prvních dvou číslic PINu. Útok sníží množství možných kombinací PINu z 10^4 na 400.

---

[6] Počet položek kódové knihy se z 10^4 zvýší pouze na 10^5.

[7] V tomto případě zvýšení entropie plně závisí na číslu účtu, pokud ho jsme schopni získat, tak se entropie nezvýší.

[8] Minimální délka PINu je čtyři, při delším PINu jsou příslušné pozice `f` použity pro PIN, ostatní mají hodnotu `F`.

[9] Nechť $x,n \in Z$ a n je sudé. Pak platí, že $x<n \Leftrightarrow x \oplus 1<n$.

4.3 Útok proti funkcím překládajícím PINy

Tento útok, publikovaný opět v [3, 4], je sice rozšířením útoku předcházejícího, ale lze jej již aplikovat pouze na překladové funkce. Ty navíc umožňují kromě PANů modifikovat i formát vstupního či výstupního PIN-bloku, čímž dávají útočníkovi mnohem větší prostor ke zneužití. Pozorujme například, co se stane při přeformátování zachyceného ANSI X9.8 EPB, je-li vstupní formátování specifikováno jako VISA-3 a výstupní jako ANSI X9.8. Pro jednoduchost nechť je PAN použitý v EPB roven nulám, čehož lze přeformátováním vždy dosáhnout. Stejně tak nechť je i výstupní PAN roven nulám.

Po dešifrování je tedy CPB = `04ppppFFFFFFFFFF` ⊕ `0000000000000000=04ppppFFFFFFFFFF`, ale PIN je extrahován podle pravidel VISA-3 jako `04pppp`. Poté je formátován do nového ANSI X9.8 PIN-bloku jako CPB=`0604ppppFFFFFFFF` ⊕ `0000000000000000=0604ppppFFFFFFFF` a znova zašifrován. Tímto se uvnitř EPB rozšířil původní čtyřmístný PIN tvaru `pppp` na šestimístný PIN tvaru `04pppp`. Aplikací předchozího útoku z 4.2 nyní již útočník může částečně identifikovat všechny číslice původního PINu a prostor možných PINů tak snížit z 10^4 na 16.

K jejich jednoznačnému určení je však nezbytné zároveň s výše uvedeným přeformátováním modifikovat i PAN použitý v zachyceném EPB. To při vstupním formátování VISA-3 není možné (a verifikační funkce tento vstup ignoruje). Požadované modifikace PANu je tedy nutno provést předem pomocí přeformátování EPB s vstupním i výstupním formátem ANSI X9.8.

Použijeme-li k přeformátování z VISA-3 do ANSI X9.8 například EPB s takto předem změněnou druhou číslicí PANu, pak podle pravidel VISA-3 je PIN z CPB extrahován jako `04pppp` ⊕ `00000x`. V případě, že `x=p` ⊕ `F` (tj. `x` ⊕ `p=F`) je však extrahován pouze jako `04ppp`, což lze detekovat převedením zpět do původního formátu a porovnáním obou zašifrovaných PIN-bloků. Tato metoda již umožňuje jednoznačně identifikovat všechny číslice PINu jako `p=x` ⊕ `F`, přičemž k odhalení prvních dvou číslic je opět nutno nejprve PIN rozšířit.

4.4 Kolizní útok (Collision Attack)

Tento útok byl objeven Mikem Bondem. V době psaní tohoto příspěvku však ještě nebyl v plném rozsahu veřejně publikován (nějaké informace o něm však lze nalézt v [1]), a vycházíme zde tedy především z osobní e-mailové korespondence.

Předpokládejme, že použité API je již navrženo tak, aby odolávalo všem předchozím útokům vedoucím k získání PINů (tj. například decimalizační tabulkou již nelze manipulovat apod.). Tento útok i přesto umožňuje pomocí funkce GenEncPIN (např. metodou IBM 3624 Offset) částečné identifikování posledních dvou číslic čtyřmístného PINu uloženého ve formátu ANSI X9.8. V původní verzi útoku sice Bond uvedl, že tyto dvě číslice PINu lze určit jednoznačně, ale po podrobnější analýze jsme dospěli k názoru, že celý útok umožňuje pouze jejich částečnou identifikaci (podobně jako jeden z předchozích Clulowových ANSI X9.8 útoků).

Uvažme nejprve pro jednoduchost PIN skládající se z jedné cifry[10]. Níže (viz obr. 4) jsou pro dvě odlišná čísla účtu uvedeny dvě množiny všech deseti EPB, které byly vygenerovány například opět pomocí offsetů.

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**

Obr. 4: Příklad kolizního útoku.

[10] Předpokládejme, pro účely příkladu, že se tato cifra XORuje s číslem účtu, což ve skutečnosti není vždy pravda.

Jediné, co pro dané číslo účtu útočník vidí, je EPB. Sledováním obou množin však může navíc pozorovat, že v levé množině chybí EPB s hodnotou `AC42` a `9A91`. Jednoduchým výpočtem pak snadno zjistí, že jim odpovídá hodnota PINu `8` nebo `9`.

Základní myšlenka útoku tedy spočívá ve využití malého počtu hodnot vzniklých XORováním číslic '`0`' – '`9`'. Hledáním kolizí mezi výstupy funkce GenEncPIN pak lze pomocí offsetu a čísla účtu částečně odhalit číslice PINu v EPB. Připomeňme, že GenEncPIN nejprve vypočítá z validačních dat IPIN a přičte k němu offset (modulo 10). Tím je vytvořen PIN, který je společně s PANem formátován do ANSI X9.8 CPB a zašifrován. Během tohoto procesu se poslední dvě cifry PINu XORují s prvními dvěma ciframi PANu. Nyní se pokusme GenEncPIN více formalizovat. Většina parametrů funkce GenEncPIN bude mít během útoku stejnou hodnotu a neovlivní tedy generovaný PIN. Díky tomu na ni můžeme nahlížet jako na čtyři pseudonáhodné funkce (F_a, F_b, F_c, F_d), z nichž každá bude mít jako parametry první dvě cifry PANu (označme je e a f). To je dáno tím, že poslední dvě cifry čtyřmístného PINu (uloženého ve formátu ANSI X9.8) se XORují pouze s prvními dvěmi ciframi PANu, jejichž jakákoliv změna však ovlivní i celý vygenerovaný IPIN. Výsledkem těchto funkcí je tedy vždy jedna dekadická číslice IPINu, který má tvar $F_a(e, f) \parallel F_b(e, f) \parallel F_c(e, f) \parallel F_d(e, f)$, kde symbol $\parallel$ označuje zřetězení. K IPINu je dále přičten offset (a, b, c, d) tvořený číslicemi '`0`' – '`9`' a poslední dvě číslice právě vytvořeného PINu jsou XORovány s prvními dvěmi číslicemi PANu.

$$U_a = (F_a(e, f)+a) \bmod 10$$
$$U_b = (F_b(e, f)+b) \bmod 10$$
$$U_c = ((F_c(e, f)+c) \bmod 10) \oplus e$$
$$U_d = ((F_d(e, f)+d) \bmod 10) \oplus f$$

EPB se pak vypočítá jako $Encrypt(Pad(U_a, U_b, U_c, U_d))$. Celkově tedy obdržíme funkci $Generate(a, b, c, d, e, f)$, která ze vstupních čtyř číslic offsetu a prvních dvou číslic PANu vrací EPB. Její pomocí je pak útočník schopen k danému PANu částečně identifikovat dvě číslice IPINu odpovídající hodnotám $F_c(e, f)$ a $F_d(e, f)$. K získání $F_c(e, f)$ si nejprve zvolí hodnotu DELTA a modifikací offsetu hledá kolize tak, aby platilo $Generate(a, b, c, d, e, f) = Generate(a', b', c', d', e \oplus \text{DELTA}, f)$. Je-li kolize nalezena (tj. oba EPB se rovnají), tak platí $(U_a, U_b, U_c, U_d) = (U_{a'}, U_{b'}, U_{c'}, U_{d'})$ a zejména $U_c = U_{c'}$. Z této rovnosti dále dostáváme, že $\text{DELTA} = ((F_c(e, f)+c) \bmod 10) \oplus ((F_c(e \oplus \text{DELTA}, f)+c') \bmod 10)$. Dané hodnoty DELTA však lze dosáhnout pouze pomocí XORování omezeného počtu dekadických číslic, jejichž seznam je uveden níže (viz obr. 5).

```
DELTA=1: {0,1} {2,3} {4,5} {6,7} {8,9}
DELTA=2: {0,2} {1,3} {4,6} {5,7}
DELTA=3: {0,3} {1,2} {4,7} {5,6}
DELTA=4: {0,4} {1,5} {2,6} {3,7}
DELTA=5: {0,5} {1,4} {2,7} {3,6}
DELTA=6: {0,6} {1,7} {2,4} {3,5}
DELTA=7: {0,7} {1,6} {2,5} {3,4}
DELTA=8: {0,8} {1,9}
DELTA=9: {0,9} {1,8}
DELTA=A: {2,8} {3,9}
DELTA=B: {2,9} {3,8}
DELTA=C: {4,8} {5,9}
DELTA=D: {4,9} {5,8}
DELTA=E: {6,8} {7,9}
DELTA=F: {6,9} {7,8}
```

Obr. 5: Kombinace číslic pro dané DELTA.

Řekněme například, že byla detekována kolize pro DELTA=`F`. To znamená, že $(F_c(e, f)+c) \bmod 10$ má hodnotu `6`, `7`, `8` nebo `9`. Další kolize pro DELTA=`7` tuto množinu přípustných hodnot omezí na `6` a `7` a protože hodnota c je známá, lze již snadno určit i hodnotu $F_c(e, f)$. Tímto způsobem dokáže útočník pro dobře volené DELTA částečně identifikovat $F_c(e, f)$ pomocí průměrně dvou kolizí. Určení hodnoty $F_d(e, f)$ pak provede analogicky.

5 Závěr

Příspěvek vznikl na základě (a podrobnější informace lze získat v) [8] – diplomové práce J. Krhovjáka, vedené V. Matyášem a oponované D. Cvrčkem.

Hlavním cílem celého příspěvku byla prezentace útoků, které vedou k získání PINů (tzv. *PIN Recovery Attacks*). Ukázali jsme, že tyto útoky má na svědomí nedostatečná kontrola parametrů funkcí, která společně se špatným návrhem formátů PIN-bloků poskytuje útočníkovi velmi velký prostor ke zneužití.

Při návrhu nové generace API by již bylo vhodné (spolu se symetrickým algoritmem AES) použít alespoň 128bitové PIN-bloky, které by obsahovaly dostatečné množství entropie. Ani to však stále nevyřeší problémy HSM způsobené existencí a podporou mnoha standardů a norem, což v důsledku činí jednotlivá API příliš složitá.

Radikálnější změnou by pak bylo využití technik/metod asymetrické kryptografie založených na použití dat s menší entropií (tj. například hesel či PINů) [7]. Pomocí těchto metod lze v nedůvěryhodném prostředí provést autentizaci zcela bezpečně, a navíc není ani potřeba mít ustanoveny předem žádné důvěryhodné komunikační kanály – teoreticky. Z praktického hlediska skrývá ovšem i tento přístup velké množství problémů. Asymetrická kryptografie není podporována staršími HSM, které jsou stále ve velké míře používány. Je třeba používat nové formáty pro uložení klíčů a přenos dat, což přináší nové problémy – viz část týkající se PKCS#11 [8]. Banky by se musely dohodnout na jedné kořenové certifikační autoritě, nebo vytvořit mechanizmy pro uznávání vzájemných podpisů, což opět zvyšuje složitost celého řešení. V tomto kontextu je zajímavé porovnání symetrické a asymetrické kryptografie v [5], případně v [6].

Reference

[1] M. Bond, J. S. Clulow. Encrypted? Randomised? Compromised? (When Cryptographically Secured Data is Not Secure). In *Cryptographic Algorithms and Their Uses*, Eracom Workshop 2004, Queensland Australia.

[2] M. Bond, P. Zieliński. Decimalisation Table Attacks for PIN Cracking. Technical Report 560, University of Cambridge, Computer Laboratory, February 2003.

[3] J. S. Clulow. PIN Recovery Attacks. Technical Report 0520 00296, Prism, October 2001. Revised, October 2002.

[4] J. S. Clulow. The Design and Analysis of Cryptographic Application Programming Interfaces for Security Devices. Master's thesis, University of Natal, January 2003.

[5] B. Christianson, B. Crispo, and J. A. Malcolm. Public-Key Crypto-systems Using Symmetric-Key Crypto-algorithms. Security Protocols, 8th International Workshops Cambridge, UK, April 3-5, 2000.

[6] D. Cvrček. Vytvoření lokální klíčové infrastruktury. Mikulášská kryptobesídka, pp. 19–25, 2002.

[7] IEEE. IEEE P1363/D18 (Draft version 18) – Standard Specifications for Password-based Public Key Cryptographic Techniques, November 2004.

[8] J. Krhovják. Analýza útoků na aplikační programovací rozhraní pro hardwarová bezpečnostní zařízení. Master's thesis, Masaryk University Brno, 2004. Available at `http://www.fi.muni.cz/~xkrhovj/apinf/sdipr/DP_upravena_v1.pdf`.