# Crypto-World

**Informační sešit GCUCMP**

**Ročník 7, číslo 2/2005** **15. únor 2005**


# 2/2005

**Připravil: Mgr. Pavel Vondruška**
**Sešit je přednostně distribuován registrovaným čtenářům.**
**Starší sešity jsou dostupné na adrese**
**http://crypto-world.info**
**(805 registrovaných odběratelů)**


**Obsah :** **str.**

# A. Mikulášská kryptobesídka 2004

**Václav Matyáš ml., FI MU (matyas(at)fi.muni.cz), Dan Cvrček, FIT VUT (dc@workshop.tns.cz)**


Loňská *Mikulášská kryptobesídka*, tradiční předvánoční workshop věnovaný kryptografii a souvisejícím oblastem bezpečnosti informačních technologií, se konal ve dnech 6.-7. prosince 2004 v Praze. První den workshopu byl ve znamení formální verifikace protokolů. Karthik Bhargavan (Microsoft Research) jej zahájil svým příspěvkem na téma Verifying Security of Web Service Configurations, ve kterém se věnoval verifikaci protokolů pro webové aplikace, které jsou budované na bázi XML. Přednáška byla následována minipanelem moderovaným Tomášem Rosou, s diskuzí nad otázkou, zda je dokazatelná bezpečnost budoucností, nebo pouhým mýtem, ve který se snažíme věřit.

Druhý den workshopu začal úvodem do generování náhodných čísel, který přednesl Peter Hellekalek (Universita Salzburg). Velká část byla věnována testování náhodnosti a byly ukázány příklady, kdy se zdánlivě dobrý generátor ukázal až v dobře zvoleném testu jako velice špatný. Další příspěvky se věnovaly problémům a možným řešením kontroly dat při zachování anonymity, mobilní kryptografii a možnému způsobu implementace šifrovacích algoritmů. Vlastimil Klíma představil útok na jednu z nových šifer a Honza Krhovják se věnoval některým z útoků na bezpečnostní moduly, které používají banky pro veškeré operace s našimi platebními kartami.

Po přednáškách následovala druhá panelová diskuse, moderovaná Pavlem Vondruškou, o kryptografii a archivaci dokumentů. Na závěr diskuse byla předána tekutá cena úspěsnému luštiteli zprávy z děrné pásky a celý workshop zakončil Vašek Matyáš v roli Mikuláše losováním zodpovězených dotazníků a předáváním cen z tomboly.

Velké díky patří všem, kteří zodpověděli otázky organizátorů – vaše postřehy a hodnocení umožní připravit v dalších letech workshop takový, který bude odpovídat lépe vašim představám (tedy přesněji představám těch, kdo zodpověděli anketní otázky ☺). Následující řádky přináší několik zajímavých statistik – a mimo to nemůžeme nezmínit, že jako tři nejlepší příspěvky byly hodnoceny příspěvky Vl. Klímy, P. Hellekalka a dvojice J. Krhovják, D. Cvrček. Na dalších stránkách tohoto čísla jsou publikovány tři nejúspěšnější příspěvky podle hodnocení účastníků.

## Hodnocení Mikulášské kryptobesídky 2004

| *Otázka* | *Možnosti* | *Počet* | *Průměr/ procento* |
|---|---|---|---|
| **Splnil workshop Vaše očekávání?** | 1 = rozhodně ano, 5 = rozhodně ne | | **1.66** |
| **Odkud jste se dozvěděl(a) o workshopu?** | CryptoWorld | 17 | 39.53% |
| | DSM | 4 | 9.30% |
| | Kolegové | 30 | 69.77% |
| | Internet | 5 | 11.63% |
| | Jinde | 5 | 11.63% |

| | | |
|---|---|---|
| **Považujete cenu workshopu za přijatelnou?** | 1 = rozhodně ano, 5 = rozhodně ne | 1.68 |
| **Průměr hodnocení přednášek** | 1 = nejlepší hodnota, 5 = nejhorší | 2.00 |
| **Hodnocení organizace workshopu** : | 1 = nejlepší hodnota | |
| organizační zabezpečení | 5 = nejhorší hodnota | 1.44 |
| stravování a občerstvení | | 1.86 |
| místo workshopu | | 1.85 |
| panelová diskuse | | 1.63 |
| **Plánujete se zúčastnit příštího workshopu?** | 1 = rozhodně ano, 5 = rozhodně ne | 1.79 |
| Celkem odevzdaných dotazníků | **43** | |

**Sponzoři workshopu:**

**Organizováno:**

**Mediální partneři:**

Crypto-World

Pokud bychom měli celý workshop shrnout jednou myšlenkou, která zaznívala nejsilněji, tak to asi bude nutnost neustále zlepšovat naše znalosti a sledovat nejnovější vývoj, protože absolutní bezpečnost je nesplnitelným ideálem.

*Za sponzorskou podporu workshopu musíme poděkovat firmám RSA, ICZ a GiTy, společnosti TNS za organizační zázemí, dále pak časopisu Data Security Management a e-zinu Crypto-World za mediální partnerství. Příspěvky, prezentace a další informace je možné najít na našem webu [http://www.tns.cz/kryptobesidka](http://www.tns.cz/kryptobesidka) .*

# B. Útoky na šifru Hiji-bij-bij (HBB)

**Vlastimil Klíma, v.klima@volny.cz, http://cryptography.hyperlink.cz**
**LEC s.r.o., Národní 9, 110 00, Praha 1**

## Abstrakt

Hiji-bij-bij byla navržena na konferenci INDOCRYPT 2003. Je to proudová šifra se 128bitovým nebo 256bitovým klíčem a dvěma mody šifrování. V základním modu (B) generuje 128bitové bloky hesla $KS_0$, $KS_1$, $KS_2$, ..., které jsou xorovány na otevřený text. V samosynchronizačním modu (SS) je to asynchronní proudová šifra, která pracuje s celistvými 128bitovými bloky hesla KS ($KS_0$, $KS_1$, $KS_2$, ...), otevřeného textu M ($M_0$, $M_1$, $M_2$, ...) a šifrového textu C ($C_0$, $C_1$, $C_2$, ... ).

V příspěvku odhalujeme závažné slabiny v obou modech.

V modu SS pro 128bitový klíč ukazujeme útok se znalostí libovolného 1 bloku otevřeného textu $M_i$ pro $i \geq 4$. Se složitostí $2^{66}$ je možné dešifrovat celou zprávu M a odvodit celý klíč.

V modu SS pro 256bitový klíč ukazujeme útok se znalostí libovolného 1 bloku otevřeného textu $M_i$ pro $i \geq 4$ a dalších cca 66 libovolných bitů otevřeného textu, které mohou být rozprostřeny v několika různých blocích $M_j$ pro $j \geq 4$, $j \neq i$. Na základě této znalosti můžeme se složitostí $2^{67}$ dešifrovat celou zprávu M kromě prvních čtyř bloků. Známe-li navíc dalších libovolných 62 bitů otevřeného textu z prvních čtyř bloků $M_0 \| M_1 \| M_2 \| M_3$, pak se složitostí $2^{67}$ můžeme dešifrovat celou zprávu M a odvodit celý klíč.

V modu B s 256bitovým klíčem ukazujeme, že se znalostí libovolných 34 za sebou jdoucích bloků otevřeného textu lze dešifrovat celou zprávu M se složitostí $2^{140}$.

## 1 Úvod

Šifra Hiji-bij-bij [1] byla navržena ve velmi nedávné době jako jedna z mála proudových šifer společně s modem umožňujícím samosynchronizaci. V takovém modu při výpadku části šifrového textu dojde na dešifrovací straně po určité době k synchronizaci a dešifruje se opět správný souvislý text. Šifra by měla větší uplatnění, pokud by tato synchronizace probíhala na bitové úrovni, tj. byl umožněn výpadek 1 nebo několika bitů, ale HBB požaduje synchronizaci na úrovni celých neporušených 128bitových bloků. Z tohoto hlediska je její přínos jako asynchronní proudové šifry sporný, neboť místo ní lze využít 128bitovou blokovou šifru v modu CFB. V [1] se uvádí, že základním principem HBB je kombinace lineární a nelineární části, přičemž přínos HBB je spatřován v návrhu obou částí. Útoky, které popisujeme, však ukazují, že slabinou HBB je

- způsob kombinace obou částí,
- nevhodný návrh nelineární části, která vytváří velké třídy slabě ekvivalentních klíčů,
- nevhodné zpracování lineární části, která poskytuje malé množství klíčové entropie pro kombinaci s nelineární částí.

Příspěvek má následující strukturu. V druhé kapitole uvádíme nezbytný popis HBB. V kapitole 3 se zabýváme útoky na HBB v modu SS, v kapitole 4 útoky na HBB v modu B. V kapitole 5 uvádíme některé poznámky a v kapitole 6 je závěr.

## 2 Popis šifry

V tomto odstavci uvedeme nezbytný popis HBB. HBB používá lineární registr LC (512 bitů) a nelineární registr NLC (128 bitů). Oba jsou v inicializačním procesu naplněny pomocí šifrovacího klíče (KEY, 128 bitů nebo 256 bitů) a poté se pravidelně aktualizuje jejich stav v jednotlivých krocích. V modu SS závisí tato aktualizace na šifrovém textu. Obsah registrů LC a NLC je pak směšován a vytváří bloky hesla. V následujícím pracujeme s řetězci různých délek, jedná se ale vždy o celistvé násobky 32 bitů, které nazýváme slova. Pokud proměnnou indexujeme hranatými závorkami, jedná se o její odpovídající (32bitové) slovo. Například u 512bitového řetězce LC je LC[15] jeho poslední slovo. Pro zřetězení dílčích řetězců A a B používáme zápis A || B nebo (A, B) podle kontextu. Dále používáme označení SideW(LC) a MiddleW(LC) pro krajní a střední slova 512bitového řetězce LC. Konkrétně pro LC = LC[0] || LC[1] || ... || LC[15] definujeme

- SideW(LC) = LC[0] || LC[7] || LC[8] || LC[15],
- MiddleW(LC) = LC[3] || LC[4] || LC[11] || LC[12].

Negaci řetězce X bit po bitu označujeme nonX a počet jeho bitů značíme |X|. X <<< n znamená cyklický bitový posun řetězce X o n bitů doleva. Jednotlivé bity řetězců číslujeme od jedné výše zleva doprava a označujeme pomocí složených závorek. Například posloupnost prvních tří bitů řetězce LC označujeme LC{1, 2, 3}, zatímco LC{512} je poslední bit řetězce LC. Nyní zavedeme dílčí funkce a procedury Exp(), Fold(), NextState(), Evolve(), FastTranspose(), Round(), NLSub(), NLF() a celkovou funkci šifrování.

### 2.1 EXP() a FOLD()

Tyto funkce expandují nebo komprimují šifrovací klíč KEY v závislosti na jeho délce.
Pro |KEY| = 256, KEY = KEY[0] || KEY[1] || ... || KEY[7], definujeme

- Fold(KEY, 128) = (KEY[0], KEY[1], KEY[2], KEY[3]) ⊕ (KEY[4], KEY[5], KEY[6], KEY[7]),
- Fold(KEY, 64) = (KEY[0], KEY[1]) ⊕ (KEY[2], KEY[3]) ⊕ (KEY[4], KEY[5]) ⊕ (KEY[6], KEY[7]),
- Exp(KEY) = KEY || nonKEY.

Pro |KEY| = 128, KEY = KEY[0] || KEY[1] || KEY[2] || KEY[3], definujeme

- Fold(KEY, 128) = KEY,
- Fold(KEY, 64) = (KEY[0], KEY[1]) ⊕ (KEY[2], KEY[3]),
- Exp(KEY) = KEY || nonKEY || nonKEY || KEY.

### 2.2 Evolve(s, C)

Vstupem Evolve(s, C) je 256bitový řetězec $s = (s_1, s_2, ..., s_{256})$, zatímco C je 256bitový konstantní řetězec $C = (c_1, c_2, ..., c_{256})$. Evolve(s, C) je 256bitový stav celulárního automatu CA, následující po stavu s. CA je definován třídiagonální konstantní maticí A 256 x 256 tak, že Evolve(s, C) = A*s. Hlavní diagonála matice A je tvořena bity 256bitového řetězce C, tj. $A_{1,1} = c_1$, $A_{2,2} = c_2$, ..., $A_{256,256} = c_{256}$ a obě vedlejší diagonály obsahují jedničky. Operace probíhají v binární aritmetice. Označíme-li S nový stav automatu, máme

S = Evolve(s, C) = A*s, tj.

$S_1 = c_1 s_1 \oplus s_2$,
$S_2 = s_1 \oplus c_2 s_2 \oplus s_3$,
$S_3 = s_2 \oplus c_3 s_3 \oplus s_4$,
atd. ...

$S_{255} = s_{254} \oplus c_{255}s_{255} \oplus s_{256}$,
$S_{256} = s_{255} \oplus c_{256}s_{256}$.

**2.3 NextState()**
NextState(LC) pracuje se vstupním 512bitovým řetězcem LC = LC[0] || LC[1] || ... || LC[15], který je chápán jako dva 256bitové řetězce LC[0] || LC[1] || ... || LC[7] a LC[8] || LC[9] || ... || LC[15]. NextState dále používá dvě 256bitové konstanty $R_0$ a $R_1$, jejichž konkrétní hodnota je definována v [1]. S první polovinou LC je provedena lineární transformace Evolve s konstantou $R_0$, s druhou polovinou lineární transformace Evolve s konstantou $R_1$. Obě poloviny jsou poté spojeny a výsledkem je 512 bitový řetězec

NextState(LC) = Evolve(LC[0] || LC[1] || ... || LC[7], $R_0$) || Evolve(LC[8] || LC[9] || ... || LC[15], $R_1$).

NextState je lineární zobrazení. Používá se k obnově lineárního registru jednoduše takto: LC = NextState(LC). Útoky nevyužívají žádnou vlastnost konstant $R_0$ a $R_1$, ale mnohokrát se využívá fakt, že každý bit {i} nového stavu LC závisí pouze na třech bitech {i-1, i, i+1} předchozího stavu LC. Je to důsledek použití celulárního automatu CA v lineární části schématu.

**2.4 FastTranspose()**
FastTranspose(NLC) je pevně definovaná permutací 128 vstupních bitů NLC. Konkrétní hodnota permutace je definována v [1]. Útoky nevyužívají žádnou speciální vlastnost této permutace.

**2.5 NLSub() a F()**
NLSub(NLC) pracuje se 128bitovým vstupem NLC jako s 16 bajty. Každý bajt substituuje za bajt pomocí pevného substitučního boxu S, který je definován v [1]. Výsledkem je 16 bajtů, chápaných opět jako 128bitový řetězec. Naše útoky nevyužívají žádnou speciální vlastnost substitučního boxu.

Zobrazení F(NLC) pracuje se 128bitovým vstupem NLC, výstupem je 128bitový řetězec Y = F(NLC), který vznikne následujícím postupem:

1. NLC = NLCSub(NLC),
2. temp = NLC[0] ⊕ NLC[1] ⊕ NLC[2] ⊕ NLC[3],
3. for i = 0 to 3 do NLC[i] = (temp ⊕ NLC[i]) <<< (8*i + 4),
4. NLC = FastTranspose(NLC),
5. Y = NLSub(NLC).

Snadno se lze přesvědčit, že F je bijekcí na množině $\{0,1\}^{128}$. Kromě toho využijeme vlastnost, že F je silně nelineární zobrazení, které nemá žádnou užitečnou lineární aproximaci. Tato vlastnost je dokázána v [1] (Věta 5).

**2.6 Round()**
Vstupem Round(LC, NLC) je 512bitový lineární registr LC a 128bitový nelineární registr NLC, výstupem je 128bitové heslo KS a nový obsah obou registrů LC i NLC. Výstup (KS, LC, NLC) = Round(LC, NLC)definujeme následovně:

1. LC = NextState(LC),
2. NLC = F(NLC),
3. KS = NLC ⊕ SideW(LC),

4. NLC = NLC ⊕ MiddleW(LC).

### 2.7 Zašifrování s HBB v modu B a SS

V obou modech nejprve proběhne inicializační proces, definující počáteční nastavení registrů takto:

1. LC = Exp(KEY), F = Fold(KEY, 64), NLC = F || nonF,
2. for i = 0 to 15 do ($T_{i \bmod 4}$, LC, NLC) = Round(LC, NLC),
3. $LC_{-1} = T_3 \| T_2 \| T_1 \| T_0$, $NLC_{-1}$ = NLC, $C_{-3} = T_3$, $C_{-2} = T_2$, $C_{-1} = T_1$.

Zašifrování zprávy $M = M_0 \| M_1 \| ... \| M_{n-1}$ klíčem KEY na šifrový text $C = C_0 \| C_1 \| ... \| C_{n-1}$ probíhá

a) v modu B podle vztahů:

```
for i = 0 to n - 1 do
{
      (KSi, LCi, NLCi) = Round(LCi-1, NLCi-1),
      Ci = Mi ⊕ KSi,
},
```

b) v modu SS podle vztahů:

```
for i = 0 to n - 1 do
{
      (KSi, LCi, NLCi) = Round(LCi-1, NLCi-1),  poznamenejme, že LCi-1 a NLCi-1
                                                jsou v následujících krocích přepsány jinou hodnotou
      Ci = Mi ⊕ KSi,
      LCi = Exp(KEY) ⊕ (Ci, Ci-1, Ci-2, Ci-3),
      NLCi = Fold(KEY, 128) ⊕ Ci ⊕ Ci-1 ⊕ Ci-2 ⊕ Ci-3,
}.
```

## 3 Útoky na HBB v modu SS

Ukážeme útoky pro obě délky klíče. Budeme je definovat jako tvrzení a konkrétní postup útoků bude vidět z důkazů.

### 3.1 256bitový klíč

**Tvrzení 1.** *Uvažujme HBB v modu SS s 256bitovým klíčem. Předpokládejme znalost jednoho bloku otevřeného textu $M_i$ pro nějaké $i \geq 4$ a dalších 66 bitů otevřeného textu, které mohou být rozprostřeny v několika různých blocích $M_j$ pro $j \geq 4$, $j \neq i$. Potom se složitostí řádově $2^{67}$ můžeme dešifrovat celý otevřený text M kromě prvních čtyř bloků. Pokud známe navíc dalších 62 bitů z prvních čtyř bloků otevřeného textu, můžeme dešifrovat celý otevřený text a určit celý klíč se složitostí řádově $2^{67}$.*

*Důkaz.* Důkaz provedeme v následujících odstavcích. Nejprve určíme část klíče a část otevřeného textu.

#### 3.1.1 Rekonstrukce části klíče a části otevřeného textu

Ze znalosti bloku otevřeného a šifrového textu $M_i$ a $C_i$ určíme odpovídající blok hesla $KS_i = M_i \oplus C_i$. Protože $i \geq 4$, na tvorbě $KS_i$ se podílí pouze známé bloky šifrového textu. Poznamenejme, že na tvorbě prvních čtyř bloků hesla $KS_0$ až $KS_3$ se podílí i vnitřní proměnné schématu $T_0$, $T_1$, $T_2$ a $T_3$, které se nepředávají na komunikačním kanálu.

Podle definice zašifrování bloku $M_i$ pro $i \geq 4$ máme:

1. $LC_{i-1} = Exp(KEY) \oplus (C_{i-1}, C_{i-2}, C_{i-3}, C_{i-4})$,
2. $NLC_{i-1} = Fold(KEY, 128) \oplus (C_{i-1} \oplus C_{i-2} \oplus C_{i-3} \oplus C_{i-4})$,

3. $NLC_i = F(NLC_{i-1})$,
4. $LC_i = NextState(LC_{i-1})$,
5. $KS_i = NLC_i \oplus SideW(LC_i)$,
6. $C_i = M_i \oplus KS_i$.

Pro 256bitový klíč KEY máme

- $Fold(KEY, 128) = (KEY[0] \oplus KEY[4]) \,||\, (KEY[1] \oplus KEY[5]) \,||\, (KEY[2] \oplus KEY[6]) \,||\, (KEY[3] \oplus KEY[7])$,
- $Exp(KEY) = KEY \,||\, nonKEY$.

Z definice funkce NextState() máme $LC_i = NextState(LC_{i-1}) = NextState(Exp(KEY) \oplus (C_{i-1}, C_{i-2}, C_{i-3}, C_{i-4})) = NextState((KEY \,||\, nonKEY) \oplus (C_{i-1}, C_{i-2}, C_{i-3}, C_{i-4})) = Evolve(KEY \oplus (C_{i-1}, C_{i-2}), R_0) \,||\, Evolve(nonKEY \oplus (C_{i-3}, C_{i-4}), R_1)$.

Ze vztahu $LC_i = Evolve(KEY \oplus (C_{i-1}, C_{i-2}), R_0) \,||\, Evolve(nonKEY \oplus (C_{i-3}, C_{i-4}), R_1)$ a definice funkce Evolve plyne, že pokud se týká závislosti na klíči, pak $LC_i\{1, ..., 32\}$ závisí pouze na bitech $KEY\{1, ..., 32, 33\}$ a $LC_i\{225, ..., 256\}$ závisí pouze na bitech $KEY\{224, 225, ..., 256\}$.

Těchto 66 bitů klíče, tj. KEY[0], KEY[7], KEY{33} a KEY{224} určíme hrubou silou. Zvolme tedy nějakou jejich hodnotu. Z předchozího vztahu vypočítáme $LC_i\{1, ..., 32\}$ a $LC_i\{225, ..., 256\}$, tj. $LC_i[0]$ a $LC_i[7]$. Na stejných bitech klíče je však závislá i část druhé poloviny řetězce $LC_i$, proto vypočítáme také slova $LC_i[8]$ a $LC_i[15]$. Takže známe $SideW(LC_i) = LC_i[0] \,||\, LC_i[7] \,||\, LC_i[8] \,||\, LC_i[15]$.

Pomocí $SideW(LC_i)$, známého bloku hesla $KS_i$ a vztahu $KS_i = NLC_i \oplus SideW(LC_i)$ určíme blok $NLC_i$. Protože F je bijekce, vypočítáme $NLC_{i-1} = F^{-1}(NLC_i)$. Z výrazu $NLC_{i-1} = Fold(KEY, 128) \oplus (C_{i-1} \oplus C_{i-2} \oplus C_{i-3} \oplus C_{i-4})$ pak obdržíme $Fold(KEY, 128)$. Poznamenejme, že $Fold(KEY, 128)$ obsahuje značnou informaci o klíči.

Nyní ukážeme, jak vypočítat $KS_j$ pro libovolné $j \geq 4$, $j \neq i$, a tím dešifrovat celou zprávu M kromě prvních čtyř bloků.

Díky linearitě zobrazení NextState máme $LC_i \oplus LC_j = NextState(Exp(KEY) \oplus (C_{i-1}, C_{i-2}, C_{i-3}, C_{i-4})) \oplus NextState(Exp(KEY) \oplus (C_{j-1}, C_{j-2}, C_{j-3}, C_{j-4})) = NextState( (Exp(KEY) \oplus (Exp(KEY) \oplus (C_{i-1}, C_{i-2}, C_{i-3}, C_{i-4}) \oplus (C_{j-1}, C_{j-2}, C_{j-3}, C_{j-4})) = NextState((C_{i-1}, C_{i-2}, C_{i-3}, C_{i-4}) \oplus (C_{j-1}, C_{j-2}, C_{j-3}, C_{j-4}))$. Bloky $LC_i$ a $LC_j$ se tak liší o známou hodnotu. Protože známe $SideW(LC_i)$, odvodíme odtud i $SideW(LC_j)$. Ze znalosti $Fold(KEY, 128)$ dále obdržíme

- $NLC_{j-1} = Fold(KEY, 128) \oplus (C_{j-1} \oplus C_{j-2} \oplus C_{j-3} \oplus C_{j-4})$,
- $NLC_j = F(NLC_{j-1})$,
- a konečně $KS_j = NLC_j \oplus SideW(LC_j)$.

Tímto způsobem jsme schopni odšifrovat všechny bloky $M_j$, kde jsou nám známé hodnoty bitů otevřeného textu. Pokud se vypočítané a známé hodnoty bitů otevřeného textu nerovnají, klíč $KEY\{1, ..., 33, 225, ..., 256\}$ byl nesprávný a zkoušíme jeho další hodnotu. K určení jediné hodnoty klíče postačí přibližně 66 bitů informace o otevřeném textu. Složitost útoku je tak maximálně $2^{66}$ *66 operací šifrování HBB, střední hodnota je přibližně $2^{67}$ operací, neboť falešný klíč je vyloučen dříve než po 66 pokusech.

V důkazu budeme pokračovat po krátké vsuvce o slabě ekvivalentních klíčích.

### 3.1.2 Slabě ekvivalentní klíče

Poznamenejme, že ze znalosti 66 bitů klíče KEY[0], KEY[7], KEY{33} a KEY{224} a jednoho bloku otevřeného textu jsme získali dalších 128 bitů klíčové informace Fold(KEY, 128). Protože Fold(KEY, 128) = (KEY[0] ⊕ KEY[4]) || (KEY[1] ⊕ KEY[5]) || (KEY[2] ⊕ KEY[6]) || (KEY[3] ⊕ KEY[7]), známe těchto 128 + 66 = 194 hodnot:

- 66 bitů KEY{1}, KEY{2}, ... , KEY{32}, KEY{33} a KEY{96}, KEY{97}, ..., KEY{128},
- 66 bitů KEY{129}, KEY{130}, ... , KEY{160}, KEY{161} a KEY{224}, KEY{225}, ..., KEY{256},
- 62 součtů KEY{34} ⊕ KEY{162}, KEY{35} ⊕ KEY{163}, ..., KEY{95} ⊕ KEY{223}.

Viděli jsme také, že k dešifrování celé zprávy M kromě prvních čtyř bloků nám postačí tato informace, přičemž není nutné určovat dílčí bity klíče v 62 součtech KEY{34} ⊕ KEY{162}, KEY{35} ⊕ KEY{163}, ..., KEY{95} ⊕ KEY{223}. Existuje tedy vždy minimálně $2^{62}$ klíčů, které šifrují zprávu M (kromě prvních čtyř bloků) stejně. Tyto klíče proto nazýváme slabě ekvivalentní.

### 3.1.3 Rekonstrukce zbytku klíče a zprávy

Nyní hrubou silou zrekonstruujeme i zbývající bity klíče KEY{34, ..., 95}. Využijeme předpokládanou znalost dalších 62 bitů otevřeného textu z prvních čtyř bloků zprávy M.

Zvolme hodnotu KEY{34, ..., 95}. Ze součtů KEY{34} ⊕ KEY{162}, KEY{35} ⊕ KEY{163}, ..., KEY{95} ⊕ KEY{223} dopočítáme KEY{162, ..., 223}, čímž máme určeny všechny bity klíče. Nyní odšifrujeme první čtyři bloky zprávy M a výsledek porovnáme se známými 62 bity otevřeného textu. Pokud nedojde ke shodě, zkoušíme další klíč až nalezneme správný. Tato část útoku navyšuje složitost pouze aditivně, a to o zhruba $2^{62}$ šifrování HBB, takže celková složitost rekonstrukce klíče zůstává řádově $2^{67}$ operací šifrování HBB, qed.

### 3.2 128b klíč

Útok na 128bitový klíč je podobný jako na 256bitový. Odlišnost vyplývá z jiné definice funkce Fold(KEY, 128) a Exp(KEY) pro tuto délku klíče.

**Tvrzení 2.** *Uvažujme HBB v modu SS se 128bitovým klíčem. Předpokládejme znalost jednoho bloku otevřeného textu $M_i$ pro $i \geq 4$. Potom je možné dešifrovat celý otevřený text se složitostí řádově $2^{66}$.*

*Důkaz.* V tomto případě máme

- Fold(KEY, 128) = KEY,
- Exp(KEY) = KEY || nonKEY || nonKEY || KEY.

Z linearity funkcí Evolve() a NextState() vyplývá

$LC_i$ = NextState($LC_{i-1}$) = NextState(Exp(KEY) ⊕ ($C_{i-1}$, $C_{i-2}$, $C_{i-3}$, $C_{i-4}$)) = NextState((KEY || nonKEY || nonKEY || KEY) ⊕ ($C_{i-1}$, $C_{i-2}$, $C_{i-3}$, $C_{i-4}$)) = NextState( (KEY || KEY || KEY || KEY) ⊕ ($C_{i-1}$ || $nonC_{i-2}$ || $nonC_{i-3}$ || $C_{i-4}$)) = NextState(KEY || KEY || KEY || KEY) ⊕ NextState($C_{i-1}$ || $nonC_{i-2}$ || $nonC_{i-3}$ || $C_{i-4}$). Podobně $LC_j$ = NextState(KEY || KEY ||

KEY || KEY) ⊕ NextState($C_{j-1}$ || $nonC_{j-2}$ || $nonC_{j-3}$ || $C_{j-4}$). Odtud je vidět, že libovolné bloky $LC_i$ a $LC_j$ se liší o známou hodnotu nezávislou na klíči.

Dále máme $LC_i$ = NextState(KEY || KEY || KEY || KEY) ⊕ NextState($C_{i-1}$ || $nonC_{i-2}$ || $nonC_{i-3}$ || $C_{i-4}$) = ( Evolve(KEY[0] || ... || KEY[3] || KEY[0] || ... || KEY[3], $R_0$) || Evolve(KEY[0] || ... || KEY[3] || KEY[0] || ... || KEY[3], $R_1$)) ⊕ NextState($C_{i-1}$ || $nonC_{i-2}$ || $nonC_{i-3}$ || $C_{i-4}$). Proto $LC_i$ [0] a $LC_i$ [8] jsou (vzhledem ke klíči) pouze závislé na bitech KEY{1, ..., 33} a $LC_i$ [7] a $LC_i$ [15] na bitech KEY{96, ..., 128}.

Nyní určíme bity klíče KEY{1, ..., 33} a KEY{96, ..., 128} hrubou silou. Zvolme proto jejich hodnotu. Potom určíme $LC_i$ [0], $LC_i$ [7], $LC_i$ [8], $LC_i$ [15], tj. SideW($LC_i$ ). Dále Fold(KEY, 128) = $NLC_{i-1} \oplus C_{i-1} \oplus C_{i-2} \oplus C_{i-3} \oplus C_{i-4} = F^{-1}(NLC_i) \oplus C_{i-1} \oplus C_{i-2} \oplus C_{i-3} \oplus C_{i-4} = F^{-1}(KS_i \oplus SideW(LC_i)) \oplus C_{i-1} \oplus C_{i-2} \oplus C_{i-3} \oplus C_{i-4}$. Protože Fold(KEY, 128) = KEY, právě jsme určili celý klíč KEY.

Nyní porovnáme vypočtené hodnoty bitů klíče KEY s hodnotami KEY{1, ..., 33} a KEY{96, ..., 128}, zvolenými na počátku útoku. Pokud nenastane shoda, testujeme další hodnotu klíče. Pokud nalezneme více než jeden klíč, lze falešné klíče vyloučit pomocí několika dalších bitů otevřeného textu. Nakonec obdržíme pouze jediný správný klíč. Proto složitost útoku je řádově $2^{66}$ operací šifrování HBB.

Na základě získané hodnoty KEY můžeme v tomto případě dešifrovat celou zprávu M včetně prvních čtyř bloků, qed.

## 4 Útok na HBB v modu B

Následující postup platí pro obě délky klíče, ale uvažujeme pouze 256bitový klíč, kde tento útok významně snižuje sílu šifry.

**Tvrzení 3.** *Uvažujme HBB v modu B s 256bitovým klíčem. Dále předpokládejme znalost libovolných 34 následných bloků otevřeného textu. Potom je možné dešifrovat celý zbytek otevřeného textu se složitostí řádově* $2^{140}$.

*Důkaz.* V inicializačním procesu se z klíče KEY o délce 256 bitů vytvoří počáteční nastavení registrů $NLC_{-1}$ a $LC_{-1}$, viz odstavec 2.7. Z tohoto nastavení se začíná generovat heslo a současně se průběžně aktualizuje kontext z ($NLC_{i-1}$, $LC_{i-1}$) na ($NLC_i$, $LC_i$) pro i = 0, 2, ..., n-1 takto:

1. $LC_i$ = NextState($LC_{i-1}$),
2. $NLC_i$ = F($NLC_{i-1}$) ⊕ MiddleW($LC_i$),
3. $KS_i$ = F($NLC_{i-1}$) ⊕ SideW($LC_i$),
4. $C_i = M_i \oplus KS_i$.

Nejprve pro jednoduchost uvažujme, že známe 34 následných bloků otevřeného textu $M_0$ až $M_{33}$ od počátku zprávy, tj. známe bloky hesla $KS_0$ až $KS_{33}$. Ukážeme, že se složitostí cca $2^{140}$ určíme $NLC_{-1}$ a $LC_{-1}$, což postačuje k dešifrování celé zprávy M.

Nyní budeme určovat 140 bitů $LC_0${1, ..., 32, 33, 34, 35}, $LC_0${222, 223, 224, 225, ..., 256}, $LC_0${257, ..., 288, 289, 290, 291} a $LC_0${478, 479, 480, 481, ..., 512} hrubou silou. Zvolme tedy jejich hodnotu.

Potom pomocí NextState() řádek po řádku dostáváme
$LC_1${1, ..., 32, 33, 34},$LC_1${223, 224, 225, ..., 256},$LC_1${257, ..., 288, 289, 290},$LC_1${479, 480, 481, ..., 512},

$LC_2\{1, ..., 32, 33\}$, $LC_2\{224, 225, ..., 256\}$, $LC_2\{257, ..., 288, 289\}$, $LC_2\{480, 481, ..., 512\}$,
$LC_3\{1, ..., 32\}$, $LC_3\{225, ..., 256\}$, $LC_3\{257, ..., 288\}$, $LC_3\{481, ..., 512\}$,

tj. obdržíme hodnoty $SideW(LC_0)$, $SideW(LC_1)$, $SideW(LC_2)$ a $SideW(LC_3)$.

Nyní podle vztahu $NLC_{i-1} = F^{-1}(KS_i \oplus SideW(LC_i))$ pro i = 0, 1, 2 a 3 vypočítáme $NLC_{-1}$, $NLC_0$, $NLC_1$ a $NLC_2$. Dále vypočteme

- $MiddleW(LC_0) = NLC_0 \oplus F(NLC_{-1}) = NLC_0 \oplus KS_0 \oplus SideW(LC_0) = F^{-1}(KS_1 \oplus SideW(LC_1)) \oplus KS_0 \oplus SideW(LC_0)$
- $MiddleW(LC_1) = NLC_1 \oplus F(NLC_0) = NLC_1 \oplus KS_1 \oplus SideW(LC_1) = F^{-1}(KS_2 \oplus SideW(LC_2)) \oplus KS_1 \oplus SideW(LC_1)$
- $MiddleW(LC_2) = NLC_2 \oplus F(NLC_1) = NLC_2 \oplus KS_2 \oplus SideW(LC_2) = F^{-1}(KS_3 \oplus SideW(LC_3)) \oplus KS_2 \oplus SideW(LC_2)$

Mezi vypočítanými hodnotami $MiddleW(LC_0)$ a $MiddleW(LC_1)$ však musí platit 124 lineárních vztahů, neboť 124 bitů $MiddleW(LC_1)$ se vypočítá pomocí $MiddleW(LC_0)$. Podobně mezi hodnotami $MiddleW(LC_1)$ a $MiddleW(LC_2)$ platí dalších 124 lineárních vztahů. Tyto vztahy jsou vzájemně lineárně nezávislé, za předpokladu, že funkce $F^{-1}$ je kvalitní nelineární funkce. Skutečně, jestliže $F^{-1}$ nepropaguje lineární vztahy mezi vstupy ($KS_1 \oplus SideW(LC_1)$, $KS_2 \oplus SideW(LC_2)$ a $KS_3 \oplus SideW(LC_3)$) a výstupy, pak podle výše uvedených rovnic jsou $MiddleW(LC_0)$, $MiddleW(LC_1)$ a $MiddleW(LC_2)$ lineárně nezávislé. Mezi vypočítanými hodnotami musí tedy platit 248 vztahů. Pokud neplatí, hodnoty $LC_0\{1, ..., 32, 33, 34, 35\}$, $LC_0\{222, 223, 224, 225, ..., 256\}$, $LC_0\{257, ..., 288, 289, 290, 291\}$ a $LC_0\{478, 479, 480, 481, ..., 512\}$ byly zvoleny na počátku chybně. Zkoušíme další, až zůstane pouze jedna správná hodnota těchto 140 bitů. Proto složitost tohoto postupu je zhruba $2^{140}$ šifrování HBB.

Nyní využijeme znalosti dalších 30 slov $KS_4$ až $KS_{33}$ a budeme postupně rozšiřovat znalost bitů $LC_0$ v krocích pro i = 4 až 33 podle následujícího postupu.
Na počátku i-tého kroku (i = 4 až 33) známe

- $LC_0\{1, ..., 32 + (i-1)\}$, $LC_0\{225 - (i-1), ..., 256\}$, $LC_0\{257, ..., 288 + (i-1)\}$ a $LC_0\{481 - (i-1), ..., 512\}$, tedy známe také $SideW(LC_0)$,
- $LC_1\{1, ..., 32 + (i-2)\}$, $LC_1\{225 - (i-2), ..., 256\}$, $LC_1\{257, ..., 288 + (i-2)\}$ a $LC_1\{481 - (i-2), ..., 512\}$, tedy známe také $SideW(LC_1)$,
- ....
- $LC_{i-1}\{1, ..., 32\}$, $LC_{i-1}\{225, ..., 256\}$, $LC_{i-1}\{257, ..., 288\}$ a $LC_{i-1}\{481, ..., 512\}$, tedy známe také $SideW(LC_{i-1})$,
- $MiddleW(LC_0)$, $MiddleW(LC_1)$, ..., $MiddleW(LC_{i-2})$,
- $NLC_{-1}$, $NLC_0$, ..., $NLC_{i-2}$.

Nyní zkoušíme všechny možné hodnoty čtyř bitů $LC_0\{32 + i\}$, $LC_0\{225 - i\}$, $LC_0\{288 + i\}$ a $LC_0\{481 - i\}$. Z nich a z ostatních známých bitů proměnných $LC_0$ až $LC_{i-1}$ postupně pomocí NextState() vypočteme

$LC_1\{32 + (i-1)\}$, $LC_1\{225 - (i-1)\}$, $LC_1\{288 + (i-1)\}$ a $LC_1\{481 - (i-1)\}$,
$LC_2\{32 + (i-2)\}$, $LC_2\{225 - (i-2)\}$, $LC_2\{288 + (i-2)\}$ a $LC_2\{481 - (i-2)\}$,
$LC_3\{32 + (i-3)\}$, $LC_3\{225 - (i-3)\}$, $LC_3\{288 + (i-3)\}$ a $LC_3\{481 - (i-3)\}$,
...
$LC_{i-1}\{32 + 1\}$, $LC_{i-1}\{225 - 1\}$, $LC_{i-1}\{288 + 1\}$ a $LC_{i-1}\{481 - 1\}$.

Z hodnoty $SideW(LC_{i-1})$ a nově určených bitů $LC_{i-1}$ vypočítáme celé $SideW(LC_i)$. Dále vypočítáme $NLC_{i-1} = F^{-1}(KS_i \oplus SideW(LC_i))$ a odtud $MiddleW(LC_{i-1}) = NLC_{i-1} \oplus F(NLC_{i-2})$.

Mezi vypočítanou hodnotou MiddleW($LC_{i-1}$) a nám známou hodnotou MiddleW($LC_{i-2}$) však musí platit 124 lineárních vztahů, neboť 124 bitů MiddleW($LC_{i-1}$) se počítá z bitů MiddleW($LC_{i-2}$). Pokud uvedené vztahy neplatí, zkoušíme jiné hodnoty $LC_0\{32 + i\}$, $LC_0\{225 - i\}$, $LC_0\{288 + i\}$ a $LC_0\{481 - i\}$ dokud nedojde ke shodě. Po maximálně $2^4$ pokusech nalezneme jejich pravou hodnotu. Složitost určení 4 uvedených bitů je maximálně $2^4$ šifrování HBB.

Na konci i-tého kroku tedy známe

- $LC_0\{1, ..., 32 + i\}$, $LC_0\{225 - i, ..., 256\}$, $LC_0\{257, ..., 288 + i\}$ a $LC_0\{481 - i, ..., 512\}$, obsahuje SideW($LC_0$),
- $LC_1\{1, ..., 32 + (i-1)\}$, $LC_1\{225 - (i-1), ..., 256\}$, $LC_1\{257, ..., 288 + (i-1)\}$ a $LC_1\{481 - (i-1), ..., 512\}$, obsahuje SideW($LC_1$),
- ....
- $LC_i\{1, ..., 32\}$, $LC_i\{225, ..., 256\}$, $LC_i\{257, ..., 288\}$ a $LC_i\{481, ..., 512\}$, obsahuje SideW($LC_i$),
- MiddleW($LC_0$), MiddleW($LC_1$), ..., MiddleW($LC_{i-1}$),
- $NLC_{-1}$, $NLC_0$, ..., $NLC_{i-1}$.

Předpoklad pro následující krok je tedy splněn a můžeme induktivně pokračovat dále.

Složitost všech 30 kroků je maximálně $30*2^4$ šifrování HBB.

Po ukončení 33. kroku známe

- $LC_0\{1, ..., 32, 33, 34, ..., 64, 65\}$, $LC_0\{192, 193, ..., 224, 225, ..., 256\}$, $LC_0\{257, ..., 288, 289, ..., 320, 321\}$ a $LC_0\{448, 449, ..., 480, 481, ..., 512\}$,
- MiddleW($LC_{32}$), MiddleW($LC_{31}$), ..., MiddleW($LC_0$).

Z řetězce $LC_0$ tedy neznáme už jen slova $LC_0[2]$, $LC_0[5]$, $LC_0[10]$, $LC_0[13]$.

Nyní ukážeme jak určit $LC_0[2]$ na základě znalosti slov $LC_{32}[3]$, $LC_{31}[3]$, ..., $LC_0[3]$ z MiddleW($LC_{32}$), MiddleW($LC_{31}$), ..., MiddleW($LC_0$).

Postup má 32 kroků:

- Ze znalosti $LC_{32}\{97\}$, $LC_{31}\{97, 98\}$, $LC_{30}\{97, 98\}$, ..., $LC_1\{97, 98\}$, $LC_0\{97, 98\}$ určíme postupně $LC_{31}\{96\}$, $LC_{30}\{96\}$, ..., $LC_0\{96\}$. Například $LC_{31}\{96\}$ určíme ze vztahu $LC_{31}\{96\} \oplus R_0\{97\}*LC_{31}\{97\} \oplus LC_{31}\{98\} = LC_{32}\{97\}$, kde je jedinou neznámou.
- Ze znalosti $LC_{31}\{96\}$, $LC_{30}\{96, 97\}$, $LC_{29}\{96, 97\}$, ..., $LC_0\{96, 97\}$ určíme postupně $LC_{30}\{95\}$, $LC_{29}\{95\}$, ..., $LC_0\{95\}$.
- Ze znalosti $LC_{30}\{95\}$, $LC_{29}\{95, 96\}$, $LC_{28}\{95, 96\}$, ..., $LC_0\{95, 96\}$ určíme postupně $LC_{29}\{94\}$, $LC_{28}\{94\}$, ..., $LC_0\{94\}$.
- ...
- Ze znalosti $LC_1\{66\}$, $LC_0\{66, 67\}$ určíme $LC_0\{65\}$.

Tím jsme určili bity $LC_0\{65, 66, ..., 96\}$, tedy celé slovo $LC_0[2]$.

Podobně jako $LC_0[2]$ bychom ze znalosti MiddleW($LC_{32}$), MiddleW($LC_{31}$), ..., MiddleW($LC_0$) určili i zbývající slova $LC_0[5]$, $LC_0[10]$, $LC_0[13]$ ze slov 4, 11 a 12 řetězců MiddleW($LC_{32}$), MiddleW($LC_{31}$), ..., MiddleW($LC_0$). Tím jsme určili celý řetězec $LC_0$.

Nyní z $LC_0$ vypočteme $LC_{-1}$, přičemž $NLC_{-1}$ už známe. Pomocí hodnot $LC_{-1}$ a $NLC_{-1}$ dešifrujeme celý zbytek zprávy.

Na počátku útoku jsme uvažovali, že známe bloky $M_0$ až $M_{33}$. V případě, že známe bloky $M_i$ až $M_{i+33}$ pro libovolné $i > 0$, určili bychom stejným postupem jako výše hodnoty $NLC_{i-1}$ a $LC_{i-1}$. Z nich pak zpětným výpočtem určíme $NLC_{i-2}$ a $LC_{i-2}$ a dále až $NLC_{-1}$ a $LC_{-1}$. Je tedy lhostejné, v jakém místě se známé bloky otevřené zprávy nachází.

Složitost útoku je tak $2^{140} + 30*2^4$ šifrování HBB, tj. cca $2^{140}$ šifrování HBB.

Uvedený útok pro 256bitový klíč významně redukuje sílu šifry, což ukazuje na chybu v jejím návrhu.

## 5 Další poznámky k bezpečnosti a vlastnostem HBB

V modu SS má tvorba proměnných T během prvních 16 rund velmi omezený význam, neboť tyto proměnné se využijí jen pro šifrování prvních čtyř bloků otevřeného textu. Útočník může proto tuto fázi ignorovat a zaměřit se na luštění dalších bloků, které nejsou na proměnných T závislé.

Koncepce HBB jako proudové asynchronní šifry je sporná, protože by mohla být místo ní použita bloková šifra se 128bitovým blokem v modu CFB. Garance přenosu celistvých 128bitových bloků nebývá běžná.

Délka heslové posloupnosti je v popisu HBB [1] omezena na $2^{64}$ bloků, měla by však být menší. Z narozeninového paradoxu vyplývá, že se v této posloupnosti s přibližně 50% pravděpodobností nachází dva 512bitové "superbloky" šifrového textu ($C_{i+1}$, $C_{i+2}$, $C_{i+3}$, $C_{i+4}$) a ($C_{j+1}$, $C_{j+2}$, $C_{j+3}$, $C_{j+4}$) mající shodnou 128bitovou hodnotu $C_{i+1} \oplus C_{i+2} \oplus C_{i+3} \oplus C_{i+4} = C_{j+1} \oplus C_{j+2} \oplus C_{j+3} \oplus C_{j+4}$. Následující blok hesla se v těchto případech (v modu SS) liší pouze o konstantu SideW[NextState($C_{i+1}$, $C_{i+2}$, $C_{i+3}$, $C_{i+4}$)] $\oplus$ SideW[NextState($C_{j+1}$, $C_{j+2}$, $C_{j+3}$, $C_{j+4}$)]. Odtud vyzařuje 128 bitová informace o odpovídajících blocích otevřeného textu, což není žádoucí vlastnost.

## 6 Závěr

V příspěvku jsme ukázali několik závažných slabin šifry HBB, které vyplývají z nevhodného návrhu a kombinace její lineární a nelineární části. V modu SS můžeme 128bitový klíč rekonstruovat se složitostí $2^{66}$ a 256bitový klíč se složitostí $2^{67}$. V modu B s 256bitovým klíčem lze se složitostí $2^{140}$ dešifrovat celou zprávu pouze na základě znalosti 34 následných bloků otevřeného textu. Odtud vyplývá, že HBB má vážné slabiny a neměla by být používána.

## Literatura

[1] Sarkar, P.: Hiji-bij-bij: A New Stream Cipher with a Self-synchronizing Mode of Operation, in *Proc. Of Progress in Cryptology - INDOCRYPT 2003*, 4th International Conference on Cryptology in India, New Delphi, India, December 2003, Springer-Verlag, Lecture Notes in Computer Science, Vol. 2904, pp. 36 - 51, 2003.

# C. A Concise Introduction to Random Number Generators

**Peter Hellekalek (pLab, University of Salzburg)**

## 1 Introduction

Since John von Neumann we know that anyone who generates random numbers (or random bits) on a deterministic machine „is in a state of sin“ (see Knuth[7]).

It is difficult to define or measure the extent of this sin by mathematical means.

History shows that the generation and the application of random numbers is more difficult than one might expect. We refer to the story on randomness and the Netscape browser, (see *http://www.cs.berkeley.edu/~daw/rnd* for details), and to a tragic historical example, called the Rosenberg story and the VENONA files, see Kahn[6] and *http://www.nsa.gov/venona*.

### 1.1 What is our goal?

We want a device (hardware or algorithm) whose output is computationally indistinguishable from truly random sources. We would like to generate bits (or numbers) that appear like being sampled from a uniform distribution on {0; 1} (or [0; 1]), independently of each other.

What we get in practice from algorithms like AES, TT800, … and hardware generators are finite bit strings that pass many tests of randomness.

So, what is *randomness*?

### 1.2 What is this talk about?

In this talk, we will discuss criteria for random number generators. We will present some of the main theoretical concepts and we will also have a closer look at statistical testing.

## 2 Quotes

When generating random numbers or bits on a computer, we will be restricted to finite bit strings. What is a finite random sequence? In the words of Kolmogoroff,

> *„A finite sequence is random if there is no short sequence that describes it fully, in some unambigous mathematical notation.“*

In the very same spirit, Calude[1] defines

> *„A string is random if it cannot be algorithmically compressed.“*

Hence, in the context of Kolmogorov complexity,

Randomness = Incompressibility.

For practitioners it will be interesting (and amusing) to know what theoreticians think of their theoretical findings:

> *„In the previous section, we recommended new definitions of pseudorandomness in terms of strong unpredictability and*

> *Kolmogoroff complexity. In practice, we may not need such stronger definition.*
> *… it is practically su-cient to consider the sequences which withstand the following five basic tests: frequency test, serial test, polar test, runs test, autocorrelation test.“*

Randomness of finite sequences is in the eye of the beholder. As Kendall says,

> *„Finite sequences can only be random with respect to the tests of randomness used.“*

Therefore, we will have to study tests for randomness a little bit closer.

## 3 Types of Random Number Generators

We may distinguish random number generators (RNGs) by their construction principle:

- hardware-based RNGs,
- software-based (algorithmic) RNGs,

and by the intended application:

- RNGs for stochastic simulation (keyword: Monte Carlo method),
- RNGs for cryptographic applications.

Software-based RNGs, i.e., algorithms to produce random numbers, may be divided into:

- linear algorithms, and
- nonlinear algorithms.

Deterministic algorithms to produce random numbers are often called *pseudo*-random number generators. We will not use this notion here.
Every RNG has its advantages and its de□ciencies. No RNG is perfect. RNGs are tools and the appropriate range of applications for a given RNG has to be chosen with care.
Hardware RNGs rely upon physical phenomena like electronic noise or radioactive decay. Their properties are the following:

- slow,
- not portable,
- unpredictable output,
- theoretical analysis is impossible.

A typical example of a hardware RNGs is SG100, which is based on electronic noise.
Software RNGs employ deterministic algorithms to stretch a short seed

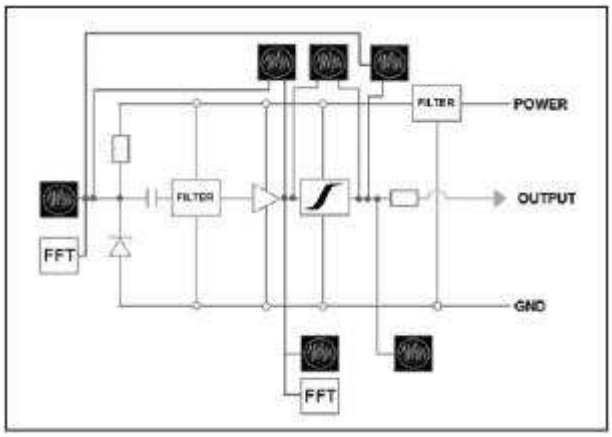



***Figure 1: SG100 (Protego SA, Sweden)***

into a long sequence. They have the following properties:

- (usually) very fast,
- portable,
- generate reproducible output,
- theoretical analysis is possible.

These algorithms may be divided into two classes, linear and nonlinear RNGs. Linear RNGs are based on a linear arithmetical operation, like a linear congruence, a linear shift register, etc. These RNGs are well-known in simulation, but insecure for cryptography.

Nonlinear RNGs employ nonlinear arithmetical operations like modular inversion in a finite field, etc. Some of these generators are quite efficient, but most are much slower than their linear counterparts. Nonlinearity does not necessarily mean cryptographic security (see Menezes et al.[11] for references). Well-known examples of nonlinear RNGs are inversive-congruential generators (ICGs; see Hellekalek[4] for a survey), AES, the advanced encryption standard (see Daemen and Rijmen[3] for the algorithm and Hellekalek and Wegenkittl[5] for statistical results), and the stream ciphers of modern applied cryptography.

## 4 Criteria

What is a good RNG?

The answer to this question will depend on the application. Three aspects will have to be studied in the assessment of an RNG:

- Is there some kind of theoretical analysis of the algorithm available?
- What statistical test results are available?
- What are the practical aspects of our RNG?

### 4.1 Theoretical Analysis

Theoretical analysis of a software-based RNG refers to the mathematical study of properties like the period length, the inner structure, and correlations within output streams.

If we cannot say anything about the period length of the output streams, an important information will be missing . For most applications, such RNGs should be avoided. If we have no idea about the existence or absence of simple (geometric) structures or simple bit patterns in the output stream then, again, we should have second thoughts about using this

RNG. If such intrinsic structures have not been studied at all, then not only the security of our RNG is at risk, but also stochastic simulations may go wrong (see Compagner[2]):

> *„Monte Carlo results are misleading when correlations hidden in the random numbers and in the simulated system interfere constructively.“*

There is an important difference between cryptographic RNGs and simulation RNGs. The algorithms for simulation RNGs are much simpler than their counterparts in cryptography. As a consequence, it is much easier to derive theoretical results for a simulation RNG than for a cryptographic RNG. In cryptography linear algorithms are avoided for reasons of security. One uses highly nonlinear algorithms. Hence, in most cases, there is no chance to carry out the type of number-theoretic correlation analysis that is so common among simulation RNGs (for the latter, see Niederreiter[13]).

### 4.2 Statistical Testing

Statistical testing of RNGs means that the RNG is treated as a black box. No use is made of the knowledge of the algorithm.

Given observations $x_1;\ldots; x_n$ of the random variables $X_1; \ldots ; X_n$ , we want to estimate some parameter or figure of merit $\theta$ or the distribution of the $X_i$'s.

We employ a function $\Theta$, the so-called test statistic, which assigns to the sample $x_1; \ldots, x_n$ the value $\Theta (x_1; \ldots; x_n)$. The function $\Theta$ should be chosen such that $\Theta(x_1; \ldots; x_n)$ is as close as possible to the target $\theta$. Statistical test designs differ by the choice of the target distribution and, if the same distribution or parameter $\theta$ is considered, by the choice of the test statistic $\Theta$. Some test statistics are numerically more effcient than others, and some have better theoretical properties or better convergence rates (if the sample size $n$ tends to infinity).

In order to illustrate such a situation, we will compare entropy estimators like the approximate entropy of Pincus and Singer[14, 15], Maurer's[10] Universal Test, and the well-known overlapping serial test (see Wegenkittl[17] for the relationship between these statistics).

For random number generation, there are two well-known batteries of statistical tests, DIEHARD of Marsaglia (see *http://www.cs.hku.hk/~diehard* for a recent version), and the NIST test suite [12]. The extensive new test bench TestU01 of L'Ecuyer and Simard[9] will provide much more extensive possibilities for statistical testing (see *http://www.iro.umontreal.ca/~simardr/* ).

## 5 Examples and Links

In the final part of my talk, I will point out an interesting concept called „HAVEGE“ by Seznec and Sendrier[16] to gather entropy that appears much more effective (and secure) than, for example, *dev/random*.

As starting points to the topics of RNGs for simulation, I would like to recommend the excellent survey of L'Ecuyer[8]. David Wagner has collected numerous links to randomness for crypto, see *http://www.cs.berkeley.edu/~daw/rnd/* .

## References

[1] C. Calude. Information and Randomness. Springer Verlag, 1994.
[2] A. Compagner. Operational conditions for random-number generation. Phys. Review E, 52:5634-5645, 1995.
[3] J. Daemen and V. Rijmen. The Design of Rijndael. Springer Verlag, New York, 2002.
[4] P. Hellekalek. Inversive pseudorandom number generators: concepts, results, and links. In C. Alexopoulos, K. Kang, W.R. Lilegdon, and D. Goldsman, editors, Proceedings of the 1995 Winter Simulation Conference, pages 255-262, 1995.
[5] P. Hellekalek and S. Wegenkittl. Empirical evidence concerning AES. ACM TOMACS, 13:322-333, 2003.
[6] D. Kahn. The Code Breakers. Macmillian, New York, 1967.
[7] D.E. Knuth. The Art of Computer Programming, Vol. 2. Addison-Wesley, Reading, Mass., third edition, 1998.
[8] P. L'Ecuyer. Random number generation. In J.E. Gentle, W. Haerdle, and Y. Mori, editors, Handbook of Computational Statistics, pages 35-70. Springer, New York, 2004.
[9] P. L'Ecuyer and R. Simard. TestU01: A Software Library in ANSI C for Empirical Testing of Random Number Generators, 2002. Software user's guide.
[10] U. Maurer. A universal statistical test for random bit generators. J. Cryptology, 5:89-105, 1992.
[11] A. J. Menezes, P. C. van Oorschot, and S.A. Vanstone. Handbook of Applied Cryptography. CRC Press, Boca Raton, 1997.
[12] National Institute of Standards and Technology (NIST) Special Publication 800-22. A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications, 2001. Available from http://csrc.nist.gov/rng .
[13] Harald Niederreiter and Igor E. Shparlinski. Recent advances in the theory of nonlinear pseudorandom number generators. In Fang, Kai-Tai (ed.) et al., Monte Carlo and quasi-Monte Carlo methods 2000. Proceedings of a conference, held at Hong Kong Baptist Univ., Hong Kong SAR, China, November 27 - December 1, 2000, pages 86-102. Springer, Berlin, 2002.
[14] S. Pincus and B. H. Singer. Randomness and degrees of irregularity. Proc. Natl. Acad. Sci. USA, 93:2083-2088, 1998.
[15] S. Pincus and B. H. Singer. A recipe for randomness. Proc. Natl. Acad. Sci. USA, 95:10367-10372, 1998.
[16] A. Seznec and N. Sendrier. HAVEGE: A user-level software heuristic for generating empirically strong random numbers. ACM TOMACS, 13:334-346, 2003.
[17] S. Wegenkittl. Monkeys, gambling, and return times: Assessing pseudo-randomness. In P.A. Farrington, H.B. Nembhard, D.T. Sturrock, and G.W. Evans, editors, Proceedings of the 1999 Winter Simulation Conference, pages 625-631, Piscataway, N.J., 1999. IEEE Press.

# Selection of slides from presentation

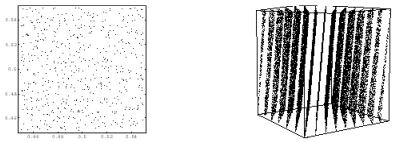
## *Phenomena*

Setup

- RNG: LCG($2^{31}$, 65539, 0, 1), i.e. RANDU
- Dimension: $d = 2, 3$
- Sample size: $N = 2^{16}$
- Plot nonoverlapping pairs $(x_{2n}, x_{2n+1})$ and triples $(x_{3n}, x_{3n+1}, x_{3n+2})$, $0 \le n < N$.

## *Phenomena: Increasing the Dimension*

We increase the dimension from $d = 2$ to $d = 3$:



Question
How to prevent such unpleasant surprises?

Answer
Theoretical correlation analysis and/or statistical testing.

## *Linear Congruential Generator (LCG)*

Parameters

- $m$ ... modulus
- $a$ ... multiplier
- $b$ ... additive constant
- $y_0$ ... initial value

Defining congruence

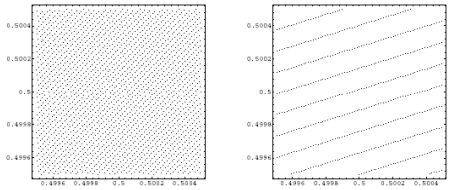
$$y_{n+1} \equiv a \cdot y_n + b \pmod{m}, \quad n \ge 0$$

... LCG$(m, a, b, y_0)$

Output stream

$$x_n := \frac{y_n}{m} \in [0, 1[, \quad n = 0, 1, \ldots$$

## *LCGs: Two Examples*



LCG($2^{31} - 1$,630360016,0,1) LCG($2^{32}$, 69069, 0,1)

## *Inversive Congruential Generator (ICG)*

Parameters

- $m$ ... modulus (usually a big prime)
- $a$ ... multiplier
- $b$ ... additive constant
- $y_0$ ... initial value

Defining congruence

$$y_{n+1} \equiv a \cdot \overline{y_n} + b \pmod{m}, \quad n \ge 0$$

($\overline{c} = c^{-1}$ for $c \ne 0$, $\overline{c} = 0$ if $c = 0$.)

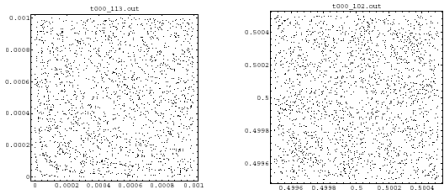
... ICG$(m, a, b, y_0)$

Output stream

$$x_n := \frac{y_n}{m} \in [0, 1[, \quad n = 0, 1, \ldots$$

## *ICG: Point Structure*

ICG($2^{31} - 1$, 1288490188, 1, 1)



lower left corner middle section

# D. Útoky na a přes API: PIN Recovery Attack

**Jan Krhovják, FI MU , xkrhovj@fi.muni.cz**
**Daniel Cvrček, Computer Laboratory University of Cambridge, daniel.cvrcek@cl.cam.ac.uk**

## 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.

## 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).

[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.

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.

## 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.

[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.

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.

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 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 4467 8901 2344` a $D_9$ = `0123 4567`

[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).

8**8**01 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) = ppppFxxxxxxxxxx.
```

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`'

[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'. 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 ⊕ 0000x00000000000, je teď PIN z P1' = CPB ⊕ P2' = CPB ⊕ P2 ⊕ 0000x00000000000= 04ppppFFFFFFFFFF ⊕ 0000x00000000000 extrahován jako pppp ⊕ 00x0. V závislosti na zvolené hodnotě x pak, pokud p ⊕ 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 ⊕ 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.

### 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 0 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 první 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.

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

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.

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 || 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.

$$U_a = (F_a(e, f)+a) \bmod 10$$

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

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

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⊕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 DELTA = (($F_c$(e, f)+c) mod 10) ⊕ (($F_c$(e⊕DELTA, f)+c') mod 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 .

# E. Call for Papers – Announcement

## MoraviaCrypt´05

The 5th Central European Conference on Cryptology
Brno, June 15 - 17, 2005

*The Faculty of Informatics, Masaryk University Brno, Czech Republic*
*with*
*The Institute of Computer Science of the Academy of Sciences of the Czech Republic, Prague*
*and*
*The Faculty of Information Technology, Brno University of Technology, Czech Republic*

are pleased to announce that the series of the Central European Conferences on Cryptology will continue with its 5th conference in Brno, Czech Republic. The conference will be held from June 15 to 17, 2005.

The aim of the conference is to cover all aspects of cryptology, including teaching relevant subjects. All parties interested to discuss recent developments in cryptography, cryptanalysis, and computational number theory are encouraged to contribute to the program of the conference.

Please submit the extended abstracts (within the strict limit of 1000-2000 words, as a PDF or PostScript file suitable for anonymous review process), by e-mail to both porubsky@cs.cas.cz and matyas@fi.muni.cz, along with a separate ASCII text file containing the paper title, names of all authors and contact details of at least one of the authors.
The deadline for submissions is March 31st, and notification of acceptance/rejection will be sent no later than April 22nd.

Visit http://www.moraviacrypt.fit.vutbr.cz for more details.

**Conference Program Committee**:

| | |
|---|---|
| Otokar Grošek (Bratislava) | Attila Pethoe (Debrecen) |
| Petr Hanáček (Brno) | **Štefan Porubský (Prague) - chair** |
| Mieczyslaw Kula (Katowice) | Tomáš Rosa (Prague) |
| Spyros Magliveras (Boca Raton) | Ladislav Satko (Bratislava) |
| Vašek Matyáš (Brno) | Tran van Trung (Essen) |
| Ján Mináč (London, Ontario) | Jerzy Urbanowicz (Warszawa) |

# F. O čem jsme psali v únoru 2000 – 2004

**Crypto-World 2/2000**



**Crypto-World 2/2001**



**Crypto-World 2/2002**



Příloha: Program pro naše čtenáře : "Hašák ver. 0.9" (viz. letem šifrovým světem) hasak.

**Crypto-World 2/2003**



Příloha : Crypto_p2.pdf
Přehled dokumentů ETSI, které se zabývají elektronickým podpisem
(ETSI - European Telecommunication Standards Institute) 10 stran

**Crypto-World 2/2004**

# G. Závěrečné informace

## 1. Sešit

Crypto-World je oficiální informační sešit "Kryptologické sekce Jednoty českých matematiků a fyziků" (GCUCMP). Obsahuje články podepsané autory. Případné chyby a nepřesnosti jsou dílem P.Vondrušky a autorů jednotlivých podepsaných článků, GCUCMP za ně nemá odbornou ani jinou zodpovědnost.

Adresa URL, na níž můžete najít tento sešit (zpravidla 3 týdny po jeho rozeslání) a předchozí sešity GCUCMP, denně aktualizované novinky z kryptologie a informační bezpečnosti, normy, standardy, stránky některých členů a další související materiály:

http://crypto-world.info

**2. Registrace / zrušení registrace**

Zájemci o e-zin se mohou zaregistrovat pomocí e-mailu na adrese pavel.vondruska@crypto-world.info (předmět: Crypto-World) nebo použít k odeslání žádosti o registraci elektronický formulář na http://crypto-world.info. Při registraci vyžadujeme pouze **jméno a příjmení, titul**, pracoviště (není podmínkou) a **e-mail adresu** určenou k zasílání kódů ke stažení sešitu.

Ke **zrušení registrace** stačí zaslat krátkou zprávu na e-mail pavel.vondruska@crypto-world.info (předmět: ruším odběr Crypto-Worldu!) nebo opět použít formulář na http://crypto-world.info . Ve zprávě prosím uveďte jméno a příjmení a e-mail adresu, na kterou byly kódy zasílány.

## 3. Redakce

**E-zin Crypto-World**

| | |
|---|---|
| Redakční práce: | Pavel Vondruška |
| Stálí přispěvatelé: | Pavel Vondruška |
| | Jaroslav Pinkava |
| Jazyková úprava: | Jakub Vrána |
| Přehled autorů: | http://crypto-world.info/obsah/autori.pdf |

| | |
|---|---|
| **NEWS** | Vlastimil Klíma |
| (výběr příspěvků, | Jaroslav Pinkava |
| komentáře a | Tomáš Rosa |
| vkládání na web) | Pavel Vondruška |

**Webmaster** Pavel Vondruška, jr.

## 4. Spojení (abecedně)

| | | |
|---|---|---|
| **redakce e-zinu** | ezin@crypto-world.info , | http://crypto-world.info |
| Vlastimil Klíma | v.klima@volny.cz , | http://cryptography.hyperlink.cz/ |
| Jaroslav Pinkava | jaroslav.pinkava@pvt.cz , | http://crypto-world.info/pinkava/ |
| Tomáš Rosa | t_rosa@volny.cz , | http://crypto.hyperlink.cz/ |
| Pavel Vondruška | pavel.vondruska@crypto-world.info , | http://crypto-world.info/vondruska/index.php |
| Pavel Vondruška,jr. | pavel@crypto-world.info , | http://webdesign.crypto-world.info |
| Jakub Vrána | jakub@vrana.cz , | http://www.vrana.cz/ |