Útoky na šifru 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}.

Klíčová slova: Hiji-bij-bij, proudová šifra, asynchronní šifra, kryptoanalýza, ekvivalentní klíče, KPA.

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

{

$(KS_i, LC_i, NLC_i) = Round(LC_{i-1}, NLC_{i-1})$,

$C_i = M_i \oplus KS_i$,

},

b) v modu SS podle vztahů:

for i = 0 to n - 1 do

{

$(KS_i, LC_i, NLC_i) = Round(LC_{i-1}, NLC_{i-1})$, poznamenejme, že LC_{i-1} a NLC_{i-1} jsou v následujících krocích přepsány jinou hodnotou

$C_i = M_i \oplus KS_i$,

$LC_i = Exp(KEY) \oplus (C_i, C_{i-1}, C_{i-2}, C_{i-3})$,

$NLC_i = Fold(KEY, 128) \oplus C_i \oplus C_{i-1} \oplus C_{i-2} \oplus C_{i-3}$,

}.

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

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

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

Ze vztahu LC_i = Evolve(KEY ⊕ (C_{i-1}, C_{i-2}), R_0) || Evolve(nonKEY ⊕ (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 ⊕ 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) ⊕ (C_{i-1} ⊕ C_{i-2} ⊕ C_{i-3} ⊕ 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 ⊕ LC_j = NextState(Exp(KEY) ⊕ (C_{i-1}, C_{i-2}, C_{i-3}, C_{i-4})) ⊕ NextState(Exp(KEY) ⊕ (C_{j-1}, C_{j-2}, C_{j-3}, C_{j-4})) = NextState((Exp(KEY) ⊕ (Exp(KEY) ⊕ (C_{i-1}, C_{i-2}, C_{i-3}, C_{i-4}) ⊕ (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}) ⊕ (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) ⊕ (C_{j-1} ⊕ C_{j-2} ⊕ C_{j-3} ⊕ C_{j-4}),
- NLC_j = F(NLC_{j-1}),
- a konečně KS_j = NLC_j ⊕ 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 128bitový 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

$LC_i = \text{NextState}(LC_{i-1}) = \text{NextState}(\text{Exp}(KEY) \oplus (C_{i-1}, C_{i-2}, C_{i-3}, C_{i-4})) = \text{NextState}((KEY \| nonKEY \| nonKEY \| KEY) \oplus (C_{i-1}, C_{i-2}, C_{i-3}, C_{i-4})) = \text{NextState}((KEY \| KEY \| KEY \| KEY) \oplus (C_{i-1} \| nonC_{i-2} \| nonC_{i-3} \| C_{i-4})) = \text{NextState}(KEY \| KEY \| KEY \| KEY) \oplus \text{NextState}(C_{i-1} \| nonC_{i-2} \| nonC_{i-3} \| C_{i-4})$. Podobně $LC_j = \text{NextState}(KEY \| KEY \| KEY \| KEY) \oplus \text{NextState}(C_{j-1} \| nonC_{j-2} \| nonC_{j-3} \| C_{j-4})$. Odtud je vidět, že libovolné bloky LC_i a LC_j se liší o známou hodnotu nezávislou na klíči.

Dále máme $LC_i = \text{NextState}(KEY \| KEY \| KEY \| KEY) \oplus \text{NextState}(C_{i-1} \| nonC_{i-2} \| nonC_{i-3} \| C_{i-4}) = (\text{Evolve}(KEY[0] \| ... \| KEY[3] \| KEY[0] \| ... \| KEY[3], R_0) \| \text{Evolve}(KEY[0] \| ... \| KEY[3] \| KEY[0] \| ... \| KEY[3], R_1)) \oplus \text{NextState}(C_{i-1} \| nonC_{i-2} \| nonC_{i-3} \| C_{i-4})$. 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. $\text{SideW}(LC_i)$. Dále $\text{Fold}(KEY, 128) = NLC_{i-1} \oplus C_{i-1} \oplus C_{i-2} \oplus C_{i-3} \oplus C_{i-4} = F^{-1}(NLC_i) \oplus C_{i-1} \oplus C_{i-2} \oplus C_{i-3} \oplus C_{i-4} = F^{-1}(KS_i \oplus \text{SideW}(LC_i)) \oplus C_{i-1} \oplus C_{i-2} \oplus C_{i-3} \oplus C_{i-4}$. 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. $LC_i = \text{NextState}(LC_{i-1})$,
2. $NLC_i = F(NLC_{i-1}) \oplus \text{MiddleW}(LC_i)$,
3. $KS_i = F(NLC_{i-1}) \oplus \text{SideW}(LC_i)$,
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 použita bloková šifra se 128bitovým blokem v modu CFB místo. 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.

7 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 Delhi, India, December 2003, Springer-Verlag, Lecture Notes in Computer Science, Vol. 2904, pp. 36 - 51, 2003.