A Near-Practical Attack Against B Mode of HBB

Joydip Mitra

Managemant Development Institute,
Post Box 60, Mehrauli Road, Sukhrali,
Gurgaon, Haryana, India 122001
joydip@mdi.ac.in

Abstract. Stream cipher Hiji-Bij-Bij (HBB) was proposed by Sarkar at Indocrypt'03. This cipher uses cellular automata (CA). The algorithm has two modes: a basic mode (B) and a self-synchronizing mode (SS). This article presents the first attack on B mode of HBB using 128 bit secret key. This is a known-pliantext guess-then-determine attack. The main step in the attack guesses 512 bits of unknown out of the 640 bits of the initial internal state. The guesses are done sequentially and the attack uses a breadth-first-search-type algorithm so that the time complexity is 2^{50}.

Keywords: cryptanalysis, known-plaintext attack, HBB, stream cipher.

1 Introduction

A typical stream cipher generates a long sequence of pseudo-random numbers, known as key-streams, from a given seed (a secret key). The plaintext message M is then XORed with the key stream to generate the ciphertext C. Thus, a steam cipher handles each bit of plaintext separately.

In this article, we will concentrate on the stream cipher HBB, proposed by Sarkar in Indocrypt'03 [1]. This is the first stream cipher replacing LFSR by CA. This is a classical masking-type stream cipher, i.e. it evolves a linear and a non-linear generator and XORs selected portions of these to produce the key stream. Thus, the design methodology is classical and there are other ciphers like SNOW which use the same principle. The non-linear part has some nice provable properties. These are aimed at resisting correlation and low diffusion attacks. The linear portion ensures a sequence of vectors with long period. Again, there are ciphers like SNOW [4] and TURING [5] which use such sequence generators. So, weakness in HBB is possibly in the way CA is used. The design has certain flaws that are to be considered while suggesting new ciphers involving CA. Ours is a guess-then-determine known-plaintext attack. The FSE'05 attack [2] was an algebraic attack. The present attack exploits structural weaknesses in greater depth than previously done. Some salient features of our attack are as follows:

(1) Exploits weakness in the use of CA. (2) Exploits the linearity in the mixing of the linear part to the non-linear part. (3) Proves an interesting property of the

B. Roy (Ed.): ASIACRYPT 2005, LNCS 3788, pp. 412–424, 2005.
© International Association for Cryptologic Research 2005

nonlinear update function: Fixing the first 32 bits of the output of the nonlinear update function ensures that there are 2^{24} choices for each of the four 32-bit blocks of the input. While by itself this is not a weakness, this is combined with the first two properties to get an efficient attack.

The HBB cipher has two modes: basic mode (B) and self-synchronizing mode (SS). So far two articles have been found in the literature dealing with cryptanalysis of HBB. (Other articles on this topic are not known to the author.) Joux and Muller [2] have shown that the SS mode of HBB is not secured. They have also attacked the B mode. Their attack requires more than 2^{50} bits of known plaintext and more than 2^{142} time. Vlastimil Klima [3] has presented another attack, marginally faster than the one in [2], on B mode. His attack requires 34 blocks of known plaintext, i.e. 34×2^7 bits of known plaintext and its time complexity is 2^{140}. Thus, so far the B mode of HBB, using 128-bit secret key, seemed secured. The present work attacks only the B mode of HBB. This attack requires 225 blocks of consecutive plaintext to be known. It guesses 512 bits of internal state in a *sequential* manner, so that the time complexity does not exceed 2^{50}. Thus, the present attack is a near-practical one and shows that the B mode of HBB using even 128 bit secret key is also not secured.

The rest of the article is organized as follows: Section 2 describes one round of B mode of HBB. (Understanding of cellular automata (CA) is not required to follow this attack. Hence CA is not discussed.) Section 3 describes our first attack having time complexity 2^{61} for finding the initial internal state of 640 bits. Next, in Section 4, we improve this attack to get the unknown 640 bits of initial internal state in 2^{50} time. Finally we present our conclusion in Section 5, followed by references.

One reviewer has pointed out that some of the ideas used in the attack has earlier occurred in [7,8,9].

2 One Round of HBB

We start by describing one round of B mode of HBB encryption. We use two 256-bit constants given by:

$$\mathcal{R}_0 = (\ 80ffaf46977969e971553bb599be6b2b\ 4b3372952308c787b84c7cce36d501e6\ )_{16}$$
$$\mathcal{R}_1 = (\ dd18c62b153df31ac98e86c1910fee24\ 2942d51b4201eb3dc1d1a85f57b8919b\ )_{16}$$

And, a 128-bit string x will also be written as a 4×32 matrix

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

where, $x = x_0 \| x_1 \| x_2 \| x_3$ and each x_i is a 32-bit string.

One Round of HBB Encryption One round of HBB encryption, i.e. encryption of i-th message block M_i, $i \geq 0$, is described as follows:

Algorithm 1. HBB Encrypt
Input: Plaintext M_i
Output: Ciphertext C_i
Internal State at the beginning of encryption:
a) Non-linear core : $N_{i-1} = N_{i-1,0} \| \ldots \| N_{i-1,3}$
b) Linear core : $L_{i-1} = L_{i-1,0} \| \ldots \| L_{i-1,15}$
/* each substring is 32-bit long */
Update internal state and compute key stream K_i and ciphertext C_i
1. Update Linear Core /* $L_i = \text{NextState}(L_{i-1})$ */
   1.1 $LX_{i-1} = L_{i-1,0} \| \ldots \| L_{i-1,7}$; $LY_{i-1} = L_{i-1,8} \| \ldots \| L_{i-1,15}$;
   1.2 $LX_i = (LX_{i-1} \ll 1) \oplus (LX_{i-1} \gg 1) \oplus (LX_{i-1} \wedge \mathcal{R}_0)$;
   1.3 $LY_i = (LY_{i-1} \ll 1) \oplus (LY_{i-1} \gg 1) \oplus (LY_{i-1} \wedge \mathcal{R}_1)$;
   1.4 $L_i = LX_i \| LY_i$;
2. Half-update Non-Linear Core /* $NZ_i = \text{updateNLC}(N_{i-1})$ */
   2.1 $NV_i = \text{NLSub}(N_{i-1})$;/* replace each byte by its image */
   2.2 $NW_i = \text{Delta}(NV_i)$;
   /* replace each word by XOR of other three words */
   2.3 $NX_i = \text{RotateLeft}(NW_i)$; /* rotate j-th word by $8 * j + 4$ bits */
   2.4 $NY_i = \text{FastTranspose}(NX_i)$;
   /* replace each 4×4 sub-matrix by its transpose */
   2.5 $NZ_i = \text{NLSub}(NY_i)$;
3. Compute Key-Stream K_i
   $K_{i,0} = NZ_{i,0} \oplus L_{i,0}$; $K_{i,1} = NZ_{i,1} \oplus L_{i,7}$;
   $K_{i,2} = NZ_{i,2} \oplus L_{i,8}$; $K_{i,3} = NZ_{i,3} \oplus L_{i,15}$;
4. Compute N_i /* updated non-linear core */
   $N_{i,0} = NZ_{i,0} \oplus L_{i,3}$; $N_{i,1} = NZ_{i,1} \oplus L_{i,4}$;
   $N_{i,2} = NZ_{i,2} \oplus L_{i,11}$; $N_{i,3} = NZ_{i,3} \oplus L_{i,12}$;
5. Compute Ciphertext C_i
   $C_i = M_i \oplus K_i$;
Internal State at the end of encryption: N_i and L_i

3 A Simple Attack on HBB

Ours is a known-plaintext attack and we will assume that the key streams K_i for $0 \leq i \leq 224$ are known. (This is equivalent to knowing (M_i, C_i) pair for $0 \leq i \leq 224$.) From the knowledge of these key streams, using a guess-then-determine attack, we will determine the entire internal state (L_0, N_0) (related to encryption of first message block M_0). Sketch of our attack is given below.

Algorithm 2: Sketch of Attack against HBB
Assumption: Key streams K_i, for $0 \leq i \leq 224$, are known.
1. Determine LX_0 /* unknown 256 bits */
2. Determine LY_0 /* unknown 256 bits */
3. Compute $N_0 = K_0 \oplus (L_{0,0} \| L_{0,7} \| L_{0,8} \| L_{0,15}) \oplus (L_{0,3} \| L_{0,4} \| L_{0,11} \| L_{0,12})$
4. Proceed forward and break the rest of the cipher.

So, the complexity of attack is really the complexity of finding the unknown 512 bits of the linear core L_0. The method for determining LX_0 and LY_0 will be similar and will have same time complexities. So, we will only discuss attack against LX_0. Time complexity of our attack will be twice the time complexity of the attack against LX_0. Idea behind the attack against LX_0 is presented below.

Let us write $LX_0 = \ell \| b_0 b_1 \dots b_{223}$ where ℓ is a 32-bit string and each b_i is a bit. We first note that, knowing $\ell \| b_0 \dots b_{t-1}$ we can compute $L_{i,0}$ for $0 \le i \le t$ uniquely. (See Algorithm A1, Appendix A for pseudocode.) Since $K_{i,0}$ are known for $0 \le i \le t$, we also know $NZ_{i,0} = L_{i,0} \oplus K_{i,0}$ for $0 \le i \le t$. But if $NZ_{i,0}$ is fixed, then $N_{i-1,0}$ can have only 2^{24} possible choices. (This result is proved in Section 3.1.) For every fixed i, the set of all such possible choices of $N_{i,0}$ will be denoted by $\mathcal{N}_{i,0}$. For every $i = 0, \dots t-1$, from $NZ_{i,0}$ (unique) and $N_{i,0}$ (one of 2^{24} choices) we get $L_{i,3} = NZ_{i,0} \oplus N_{i,0}$ (2^{24} choices). The set of all possible (2^{24}) choices of $L_{i,3}$ will be denoted by $\mathcal{L}_{i,3}$. Thus, we have,

$$\mathcal{L}_{i,3} = \{NZ_{i,0} \oplus N_{i,0} \ : \ N_{i,0} \in \mathcal{N}_{i,0}\}$$

Next we consider the update function of the linear core. Given a choice x of $L_{i,3}$, we know the middle 30 bits of corresponding choice y of $L_{i+1,3}$. We will write $x \Rightarrow y$ to denote this. So, given $\mathcal{L}_{0,3}$, an x_1 is a *valid* choice of $L_{1,3}$ only if for some $x_0 \in \mathcal{L}_{0,3}$ we get $x_0 \Rightarrow x_1$. But $\mathcal{L}_{1,3}$ is already obtained and we know that $x_1 \notin \mathcal{L}_{1,3}$ is *not* a valid choice of $L_{1,3}$. Hence, given $\mathcal{L}_{0,3}$ and $\mathcal{L}_{1,3}$, the valid choices of $L_{1,3}$ are given by the set

$$\mathcal{L}^V_{1,3}|\mathcal{L}_{0,3} = \{x_1 \in \mathcal{L}_{1,3} \ : \ x_0 \Rightarrow x_1 \text{ for some } x_0 \in \mathcal{L}_{0,3}\}$$

The super-script "V" stands for "valid". Similarly, given $\mathcal{L}_{0,3}$, $\mathcal{L}_{1,3}$ and $\mathcal{L}_{2,3}$, the valid choices of $L_{2,3}$ will be given by the set

$$\begin{aligned}\mathcal{L}^V_{2,3}|\mathcal{L}_{0,3} &= \{x_2 \in \mathcal{L}_{2,3} \ : \ x_0 \Rightarrow x_1 \Rightarrow x_2 \text{ for some } x_0 \in \mathcal{L}_{0,3},\ x_1 \in \mathcal{L}_{1,3}\} \\ &= \left\{x_2 \in \mathcal{L}_{2,3} \ : \ x_1 \Rightarrow x_2 \text{ for some } x_1 \in \mathcal{L}^V_{1,3}|\mathcal{L}_{0,3}\right\}\end{aligned}$$

We now define the following sets

$$\mathcal{N}^V_{i,0} = \left\{x \oplus NZ_{i,0} \ : \ x \in \mathcal{L}^V_{i,3}|\mathcal{L}_{0,3}\right\}$$

As a convention, we take $\mathcal{N}^V_{0,0} = \mathcal{N}_{0,0}$. Proceeding this way we can find $\mathcal{N}^V_{i,0}$ for $0 \le i \le t-1$ and for a wrong choice of $\ell \| b_0 \dots b_{223}$, the set $\mathcal{N}^V_{223,0}$ will be empty. This constitutes an attack against LX_0. The idea is summarized below.

Algorithm 3: Idea behind first attack against LX_0

Guess $\ell \| b_0 \dots b_{t-1}$						
Compute	$L_{0,0}$	$L_{1,0}$	$\dots$	$L_{t-1,0}$	$L_{t,0}$	(unique choice)
Compute	$NZ_{0,0}$	$NZ_{1,0}$	$\dots$	$NZ_{t-1,0}$	$NZ_{t,0}$	(unique choice)
Compute	$\mathcal{N}_{0,0}$	$\mathcal{N}_{1,0}$	$\dots$	$\mathcal{N}_{t-1,0}$	$\mathcal{N}_{t,0}$	(2^{24} choices)
Compute	$\mathcal{N}^V_{0,0}$	$\mathcal{N}^V_{1,0}$	$\dots$	$\mathcal{N}^V_{t-1,0}$		(shrinking sets)

Certain finer points are to be noted. First, if at any stage $\mathcal{N}^V_{i,0} = \phi$ then $\mathcal{N}^V_{223,0} = \phi$. So, we need not compute $\mathcal{N}^V_{223,0}$ to declare a choice $\ell \| b_0 \dots b_{223}$ of LX_0 to be wrong, and we can guess these bits in LX_0 *sequentially*. Second, we can compute $\mathcal{N}^V_{i,0}$ without computing $\mathcal{L}^V_{i,3} | \mathcal{L}_{0,3}$ and without explicitly computing even $\mathcal{N}_{i,0}$. (This computation and reason for doing this are explained in section 3.2.) Third point is a more important one. Suppose ℓ in LX_0 is fixed and suppose for all possible choices of $b_0 \dots b_{t-1}$, we have computed the sets $\mathcal{N}^V_{t,0}$. These sets can be kept in a binary tree T. Root of T will be ℓ. For every other non-leaf node $y \in T$, its left (right) child will be the string $y \| 0$ ($y \| 1$). The node represented by x in T, with $|x| = 32 + t$ bits, will contain the set $\mathcal{N}^V_{t-1,0}$ for $x = \ell \| b_0 \dots b_{t-1}$ *only if* $\mathcal{N}^V_{t-1,0} \neq \phi$. Thus T may have 2^t nodes at level t. The actual number may be less if some of the sets $\mathcal{N}^V_{t-1,0}$ are empty. (Level of root is zero.) Now from each of the sets $\mathcal{N}^V_{t-1,0}$ at level t of T, and for each choice of $b_t = 0, 1$, we will compute $\mathcal{N}^V_{t,0}$. But the resulting set will be added to the tree only if it is non-empty. It will be argued that this breadth-first type processing of sets can be done in 2^{48} time giving LX_0.

Below the two sections (3.1 and 3.2) contain our results and algorithms to be used in the subsequently explained attack and its complexity (Section 3.3).

3.1 Determine $N_{i-1,0}$ from $NZ_{i,0}$

Suppose we know $NZ_{i,0}$ for some $i \geq 1$. Since,

$$NY_{i,0} = \text{NLSub}^{-1}(NZ_{i,0})$$

we can find out $NY_{i,0} = (y_{31} \dots y_0)$. Since FastTranspose transposes every 4×4 sub-matrix of its input, it is an idempotent function. Thus using $NY_i = \text{FastTranspose}(NX_i)$ we get $NX_i = \text{FastTranspose}(NY_i)$ and hence,

$$NX_i = \begin{bmatrix} y_{31} & y_{27} & y_{23} & y_{19} & y_{15} & y_{11} & y_7 & y_3 \\ y_{30} & * & y_{26} & * & y_{22} & * & y_{18} & * & y_{14} & * & y_{10} & * & y_6 & * & y_2 & * \\ y_{29} & y_{25} & y_{21} & y_{17} & y_{13} & y_9 & y_5 & y_1 \\ y_{28} & y_{24} & y_{20} & y_{16} & y_{12} & y_8 & y_4 & y_0 \end{bmatrix}$$

where every "$*$" represents an unknown 4×3 matrix of bits. But NX_i was calculated as RotateLeft(NW_i) and so, NW_i can be computed as

$$NW_i = \text{RotateRight}(NX_i)$$

where j-th word of NX_i is given a circular rotation by $8 * j + 4$ bits. Hence,

$$NW_i = \begin{bmatrix} y_3 & y_{31} & y_{27} & y_{23} & y_{19} & y_{15} & y_{11} & y_7 \\ y_{10} & * & y_6 & * & y_2 & * & y_{30} & * & y_{26} & * & y_{22} & * & y_{18} & * & y_{14} & * \\ y_{17} & y_{13} & y_9 & y_5 & y_1 & y_{29} & y_{25} & y_{21} \\ y_{24} & y_{20} & y_{16} & y_{12} & y_8 & y_4 & y_0 & y_{28} \end{bmatrix}$$

where every "$*$" represents an unknown 4×3 matrix of bits. Next, we note that "Delta" is also an idempotent operation, and hence, $NV_i = \text{Delta}(NW_i)$. So, if

we write $NV_{i,0} = v_{31} \dots v_0$ then,

$$\begin{aligned} v_{31} &= y_{10} \oplus y_{17} \oplus y_{24} \\ v_{27} &= y_6 \oplus y_{13} \oplus y_{20} \\ v_{23} &= y_2 \oplus y_9 \oplus y_{16} \\ v_{19} &= y_{30} \oplus y_5 \oplus y_{12} \\ v_{15} &= y_{26} \oplus y_1 \oplus y_8 \\ v_{11} &= y_{22} \oplus y_{29} \oplus y_4 \\ v_7 &= y_{18} \oplus y_{25} \oplus y_0 \\ v_3 &= y_{14} \oplus y_{21} \oplus y_{28} \end{aligned}$$

Thus, given $NZ_{i,0}$, we know 8 specified bits of $NV_{i,0}$. In particular, we know 2 bits of every byte of $NV_{i,0}$. Finally note that, $N_{i-1,0} = \text{NLSub}^{-1}(NV_{i,0})$ and hence, given $NZ_{i,0}$ we know 8 bits of image (NLSub) of $N_{i-1,0}$. To make this formal, let us define two functions $g_1(x),\ g_2(x)\ :\ \{0,1\}^{32} \to \{0,1\}^8$ as follows:

Function $g_1(x)$
1. Compute $y = \text{NLSub}(x) = y_{31} \dots y_0$
2. Compute $a = y_{31}y_{27}y_{23}y_{19}y_{15}y_{11}y_7y_3$
3. Return a

Function $g_2(x)$
1. Compute $y = \text{NLSub}^{-1}(x) = y_{31} \dots y_0$
2. **for** $j = 0, \dots, 7$ **do**
3. $\quad a_j = y_{14+4j} \oplus y_{21+4j} \oplus y_{28+4j}$ (subscripts are computed mod 32)
4. **end-do**
5. Compute $a = a_7a_6a_5a_4a_3a_2a_1a_0$
6. Return a

Also, for $a \in \{0,1\}^8$, define the following sets:

$$\mathcal{N}^*(a) = \left\{x \in \{0,1\}^{32}\ :\ g_1(x) = a\right\} \tag{1}$$

Then, using functions g_1, g_2 and sets $\mathcal{N}^*(a)$, we have the following proposition:

Proposition 1. *Given $NZ_{i,0}$, there are exactly 2^{24} choices of $N_{i-1,0}$ given by:*

$$\mathcal{N}_{i-1,0} = \mathcal{N}^*(g_2(NZ_{i,0}))$$

In particular, every byte of $N_{i-1,0}$ can have 2^6 choices.

3.2 Compute $\mathcal{N}^V_{t+1,0}$ from $\mathcal{N}^V_{t,0}$

For k-bit strings $x = x_{k-1} \dots x_0$ and $r = r_{k-1} \dots r_0$, with $k > 2$, define the following:

1. $m(x)$ will denote the string obtained from x by deleting its MSB and LSB.
2. $f(x,r) = (x_{k-2} \dots x_0 \| 0) \oplus (0 \| x_{k-1} \dots x_1) \oplus (x \wedge r)$.

Using this notation, $y\prime \in \mathcal{L}^V_{t+1,3}|\mathcal{L}_{0,3}$ if and only if $y\prime \in \mathcal{L}_{t+1,3}$ and, for some choice $x\prime \in \mathcal{L}^V_{t,3}|\mathcal{L}_{0,3}$, we get

$$m(f(x\prime, r)) = m(y\prime) \tag{2}$$

with r chosen suitably from definition of EvolveCA function given in HBB [1].

Lemma 1. *Fix $y = y_{31} \dots y_0$. Then $y \in \mathcal{N}^V_{t+1,0}$ if and only if $y \in \mathcal{N}_{t+1,0}$ and for some $x \in \mathcal{N}^V_{t,0}$ and some $\epsilon_{31}, \epsilon_0 \in \{0,1\}$,*

$$\epsilon_{31} \| m(f(x \oplus NZ_{t,0}, r)) \oplus m(NZ_{t+1,0}) \| \epsilon_0 = y \tag{3}$$

Proof. The proof follows using equation 2.

To check if $y \in \mathcal{N}_{t+1,0}$, we use the fact that,

$$\mathcal{N}_{t+1,0} = \mathcal{N}^*(g_2(NZ_{t+2,0})) = \{y \ : \ g_1(y) = g_2(NZ_{t+2,0})\}$$

So, we can compute $\mathcal{N}^V_{t+1,0}$ as follows: Initialize sets $D1[j],\ 0 \le j < 256$ as empty sets. For each $x \in \mathcal{N}^V_{t,0}$ and for every $\epsilon_{31}, \epsilon_0 \in \{0,1\}$, we compute y as in equation 3. Then insert y to the set $D1[g1(y)]$. Once we have exhausted $\mathcal{N}^V_{t,0}$, we set $\mathcal{N}^V_{t+1,0}$ as $D1[g2(NZ_{t+2,0})]$. The pseudocode is given below.

Algorithm 4. Computation of $\mathcal{N}^V_{t+1,0}$ from $\mathcal{N}^V_{t,0}$

```
0. for v1 = 0 to 255 do
1.     D1(v1) ← φ /* set initialized by φ */
2. end-do
3. for every x ∈ N^V_{t,0} do
4.     z ← m(f(x ⊕ NZ_{t,0}, r)) ⊕ m(NZ_{t+1,0})
5.     for (ε31, ε0) ∈ {0,1}^2 do
6.         Set y ← ε31 ‖ z ‖ ε0
7.         Add y to the set D1[g1(y)]
8.     end-do
9. end-do
10. N^V_{t+1,0} ← D1[g2(NZ_{t+2,0})]
```

This computation of $\mathcal{N}^V_{t+1,0}$ has two major advantages: First, we do not need to maintain the set $\mathcal{N}_{t+1,0}$ (having 2^{24} elements) explicitly and hence no time is required to handle such sets. Second advantage is that we can compute the sets $D1[v]$ for $0 \le v < 256$ without the knowledge of $NZ_{t+2,0}$. This is going to be useful while finding the value of LX_0.

3.3 Algorithm to Determine LX_0

All valid choices of LX_0 will be found and will constitute a list FX. Writing $LX_0 = L_{0,0} \| b_0 b_1 \dots b_{223}$, we can find FX as union of sets $FX(\ell)$ that represents all valid choices of LX_0 with $L_{0,0} = \ell$. For a given value ℓ of $L_{0,0}$, we will now construct the set $FX(\ell)$. Suppose, we have a list $F0(t)$ of tuples $(y, n_0, n_1, D[y])$ having the following interpretation:

1. n_0 represents $NZ_{t,0}$ when $\ell \| b_0 \ldots, b_t = y$,
2. n_1 represents $NZ_{t+1,0}$ when $\ell \| b_0 \ldots, b_t = y$, and
3. $D[y]$ represents $\mathcal{N}^V_{t,0}$ when $\ell \| b_0 \ldots b_t = y$.

Then, y is an invalid choice of $\ell \| b_0 \ldots, b_t$ if $D[y]$ is empty. So, we also put the restriction on $F0(t)$ that it will contain only those tuples $(y, n_0, n_1, D[y])$ for which $D[y]$ is non-empty. With this interpretation, clearly $FX(\ell) = F0(223)$. If for some $t < 223$ the set $F0(t)$ is empty, then the set $F0(223)$ is also empty. Hence, the following steps will compute $FX(\ell)$ for $L_{0,0} = \ell$.

Step 1 Build list $F0(0)$

Using $K = K_{0,0}$, compute $n_0 = NZ_{0,0} = L_{0,0} \oplus K$ and, initialize the list $F0$ by ϕ. Next take, $K = K_{1,0}$ and for each $b = b_0 \in \{0, 1\}$, set $y \leftarrow \ell \| b$, compute $L_{1,0}$ from $\ell \| b$, and then compute

$$n_1 = NZ_{1,0} = L_{1,0} \oplus K$$

Define $D[y] = \mathcal{N}^*(g_2(n1))$ and add the following tuple to the list $F0(0)$:

$$(y,\ n0,\ n1,\ D[y])$$

provided the set $D[y]$ is non-empty. Thus, $F0(0)$ now looks like the following:

$$F0(0) = \{(\ell \| 0, n_0, n_1, D[\ell \| 0]),\quad (\ell \| 1, n_0, n_1, D[\ell \| 1])\}$$

The set $D[\ell \| 0]$ represents $\mathcal{N}^V_{0,0}$ for $L_{0,0} = \ell$ and $b_0 = 0$. The other sets in the list $F0(0)$ has similar interpretations. This completes our computation of $F0(0)$. In the remaining steps, we will build list $F0(t+1)$ from $F0(t)$.

Step 2 Set $t \leftarrow 0$.

Step 3 Compute $F0(t+1)$ from $F0(t)$

For each tuple in $F0(t)$, we first generate sets $D1(v1)$ for $0 \leq v1 < 256$ using Algorithm 3. Then for every value of $b \in \{0, 1\}$, we compute the corresponding $\mathcal{N}^V_{t+1,0}$ as follows:

1. Compute $L_{t+2,0}$ from $y \| b$ as in Algorithm A2, Appendix A.
2. Compute $NZ_{t+2,0} = L_{t+2,0} \oplus K_{t+2,0}$,
3. Set $D[y \| b] = D1(g_2(NZ_{t+2,0}))$,
4. Add a tuple $(y \| b, n_1, NZ_{t+2,0}, D[y \| b])$ to $F0(t+1)$ only if $D[y \| b]$ is non-empty.

Note that, $y \| b$ represents a valid choice of $\ell \| b_0 \ldots b_t b_{t+1}$ if and only if the set $D[y \| b]$ is non-empty. For such valid choices of $\ell \| b_0 \ldots b_t b_{t+1}$, the corresponding tuples are put in a list $F0(t+1)$. Once, we have exhausted all the elements in the list $F0(t)$, we have generated $F0(t+1)$. Again, elements in $F0(t+1)$ will have interpretations (similar to that) given in the beginning of this step. Finally, this computation of $F0(t+1)$ from $F0(t)$ is presented in the following algorithm.

Algorithm 5. Compute $F0(t+1)$ from $F0(t)$

```
0. for each (y, n_0, n_1, D[y]) ∈ F0(t) do
1.     Compute D1(0), ..., D1(255) from D[y] using Algorithm 3.
2.     for b ∈ {0,1} do
3.         Compute n_2 = NZ_{t+2,0} from y, b and K_{t+2,0}.
4.         Set D[y||b] = D1[g_2(n_2)].
5.         if     D[y||b] is non-empty
6.         then Add tuple (y||b, n_1, n_2, D[y||b]) to F0(t+1).
7.         end-if
8.     end-do
9. end-do
```

Step 4 $t \leftarrow t+1$

Step 5 Check for loop

If now $t < 223$ and the list $F0(t)$ is non-empty, go to Step 3.

Step 6 Compute $FX(\ell)$

Set $FX(\ell) \leftarrow F0(t)$.

We now argue that this process does not lead to handling of infinite sets. We have seen *empirically* that for every choice of $NZ_{0,0}$, $NZ_{1,0}$ and $NZ_{2,0}$, the set $D[\ell \| b_0 b_1]$ has less than 2^{21} elements for each possible choice of $\ell \| b_0 b_1$. In other words, the size of $D[\ell \| b_0 b_1]$ is at most 1/8-th of the size of $D[\ell \| b_0]$. But for every b_0, there are two choices of b_1. Hence, if η_t denotes the total number of 32-bit strings contained in the $D[\,]$ sets of list $F0(t)$, then

$$\eta_1 \leq \frac{1}{4}\eta_0 = 2^{-2}\eta_0$$

Proceeding this way, we will have $\eta_t \leq 2^{-t}\eta_0$. Note that, $F0(0)$ contains two tuples, each of which contains a set $D[\,]$ of 2^{24} elements. Thus, $\eta_0 = 2^{24} + 2^{24} = 2^{25}$. So, the total number of 32-bit strings contained in all $F0(t)$ for $0 \leq t \leq 223$ is

$$\sum_{t=0}^{223} \eta_t < \sum_{t \geq 0} \eta_t = \eta_0 \sum_{t \geq 0} 2^{-2t} < 2^{26}$$

Now, from Algorithms 4 and 5, it is clear that, during the computation of $F0(t+1)$ from $F0(t)$, each string from each set $D[\,]$ in $F0(t)$ will be processed only once. And for each such processing, we will consider four possible choices of ϵ_{31} and ϵ_0. Hence the total number of computation of strings in the entire process of finding $FX(\ell)$ is given by

$$4 \times \sum_{t=0}^{223} \eta_t < 4 \times 2^{26} = 2^{28} \tag{4}$$

Now note that there are 2^{32} choices of ℓ and so, we have proved the following:

Proposition 2. *For every fixed value of ℓ, the set $FX(\ell)$ can be computed in less than 2^{28} time. And so, time complexity of finding the set FX is $2^{32} \times 2^{28} = 2^{60}$. In other words, time complexity of finding LX_0 is 2^{60}.*

Since each element in $F0(t)$ gives rise to at most two tuples in $F0(t+1)$, and since $F0(0)$ has two tuples, number of tuples in $F0(t)$ will be at most 2^{t+1}. The $D[\,]$ sets in all these tuples will together contain $\eta_t = 2^{25-2t}$ strings. Hence for $t > 8$, some sets $D[\,]$ are bound to be empty. For example, for $k = 10$, the list $F0$ is supposed to contain at most 2^{11} tuples, and the $D[\,]$ sets will have at most 2^5 strings. So, $F0(10)$ can not contain more than 2^5 valid tuples (with corresponding non-empty set $D[\,]$). Thus, the list $F0(\,)$ will go on shrinking for $t > 8$. Hence, we are going to get a singleton set FX.

Complexity of First Attack Against HBB: By Proposition 2, LX_0 can be found in 2^{60} time. Time complexity for finding LY_0 will be the same and so, time complexity of our first attack against B–mode of HBB has time complexity:

$$2^{60} + 2^{60} = 2^{61}.$$

4 A Faster Attack

This attack is almost same as the first attack, except for computation of the set $\mathcal{N}^V_{t,0}$. We first note the following: Fix $t \geq 0$. Let $N_{t,0} = H\|L$ where H (L) is a 16-bit string and "$\|$" represents concatenation of strings. Then, by Proposition 1, H (L) will have 2^{12} choices. We will denote the collection of all such choices of H (L) by $\mathcal{HN}_{t,0}$ $(\mathcal{LN}_{t,0})$ and write $\mathcal{N}_{t,0} = \mathcal{HN}_{t,0}\|\mathcal{LN}_{t,0}$. We will now mimic the computation of $\mathcal{N}^V_{t,0}$ from the sets $\mathcal{N}_{i,0}$ $0 \leq i \leq t+1$. In the same way, we can compute the set $\mathcal{HN}^V_{t,0}$ from the sets $\mathcal{HN}_{i,0}$ $0 \leq i \leq t+1$. The pseudocode is given in Algorithm A3, Appendix A. Similarly, we can compute $\mathcal{LN}^V_{t,0}$. (For pseudocode, see Algorithm A4, Appendix A.) Clearly, $\mathcal{N}^V_{t,0}$ will be a subset of $\mathcal{HN}^V_{t,0}\|\mathcal{LN}^V_{t,0}$. If for any t, one of the sets $\mathcal{HN}^V_{t,0}$ or $\mathcal{LN}^V_{t,0}$ is empty, so will be $\mathcal{N}^V_{t,0}$. So, we will only compute $\mathcal{HN}^V_{t,0}$ for $t \geq 0$. The computation follows similar steps as in Section 3.3. Thus, our faster attack can be described by the following algorithm:

Algorithm 6. Sketch of faster attack

Guess $\ell\|b_0 \ldots b_{t-1}$						
Compute	$L_{0,0}$	$L_{1,0}$	$\ldots$	$L_{t-1,0}$	$L_{t,0}$	(unique choice)
Compute	$NZ_{0,0}$	$NZ_{1,0}$	$\ldots$	$NZ_{t-1,0}$	$NZ_{t,0}$	(unique choice)
Compute	$\mathcal{HN}_{0,0}$	$\mathcal{HN}_{1,0}$	$\ldots$	$\mathcal{HN}_{t-1,0}$	$\mathcal{HN}_{t,0}$	(2^{12} choices)
Compute	$\mathcal{HN}^V_{0,0}$	$\mathcal{HN}^V_{1,0}$	$\ldots$	$\mathcal{HN}^V_{t-1,0}$		(shrinking sets)

For computations of $\mathcal{HN}^V_{t,0}$, we introduce the following functions:

Function $g_3(x)$: $\{0,1\}^{16} \to \{0,1\}^4$
1. Compute $y = \mathrm{NLSub}(x) = y_{15} \ldots y_0$
2. Compute $a = y_{15}y_{11}y_7y_3$
3. Return a

Function $g_4(x)$: $\{0,1\}^{32} \rightarrow \{0,1\}^4$
1. Compute $y = \text{NLSub}^{-1}(x) = y_{31} \dots y_0$
2. **for** $j = 0, \dots, 7$ **do**
3. $\quad a_j = y_{14+4j} \oplus y_{21+4j} \oplus y_{28+4j}$ (subscripts are computed mod 32)
4. **end-do**
5. Compute $a = a_7 a_6 a_5 a_4$
6. Return a

Function $g_5(x)$: $\{0,1\}^{32} \rightarrow \{0,1\}^4$
1. Compute $y = \text{NLSub}^{-1}(x) = y_{31} \dots y_0$
2. **for** $j = 0, \dots, 7$ **do**
3. $\quad a_j = y_{14+4j} \oplus y_{21+4j} \oplus y_{28+4j}$ (subscripts are computed mod 32)
4. **end-do**
5. Compute $a = a_3 a_2 a_1 a_0$
6. Return a

Now for $a \in \{0,1\}^4$, defining the sets $\mathcal{J}^*(a) = \left\{x \in \{0,1\}^{16} \ : \ g_3(x) = a\right\}$, we get the following from Proposition 1:

$$\mathcal{HN}_{i-1,0} = \mathcal{J}^*(g_4(NZ_{i,0})) \text{ and, } \mathcal{LN}_{i-1,0} = \mathcal{J}^*(g_5(NZ_{i,0}))$$

where each set $\mathcal{J}^*(\ )$ contains 2^{12} elements. The pseudocodes are given in Algorithm A3 and A4 of Appendix A. We have seen *empirically* that, for $t \geq 0$,

$$\frac{\text{size of } \mathcal{HN}^V_{t+1,0}}{\text{size of } \mathcal{HN}^V_{t,0}} < \frac{1}{2\sqrt{2}} \quad \text{and,} \quad \frac{\text{size of } \mathcal{LN}^V_{t+1,0}}{\text{size of } \mathcal{LN}^V_{t,0}} < \frac{1}{2\sqrt{2}}$$

So, in this revised faster attack, each initial tuple contains one set, of cardinality 2^{12} (as opposed to 2^{24} elements in the first attack). Define π_t to be the sum of cardinalities of all surviving $\mathcal{HN}^V_{t,0}$ sets, for all values of $\ell \| b_0 \dots b_t$ with fixed ℓ. Then, $\pi_t \leq \frac{1}{\sqrt{2}} \pi_{t-1}$. So, the complexity of finding $FX(\ell)$ for a given ℓ is

$$4 \times \sum_{t=0}^{223} \pi_t < 4 \times 2^{13} \times \sum_{t=0}^{223} \left(\frac{1}{\sqrt{2}}\right)^t \leq 2^{17}$$

as opposed to 2^{28} (equation 4) for the first attack. So, this revised attack is faster than the first attack by a margin of $2^{11} (= 2^{28}/2^{17})$. In other words, the time required to find L_0 is given by: $2^{61}/2^{11} = 2^{50}$.

5 Conclusion

We have presented an attack against the B mode of HBB. The time complexity of the attack is 2^{50} requiring 225 blocks of plaintext to be known. Thus, HBB using even 128-bit secret key is also not secured. We think there are certain design weaknesses in HBB shown by our attack: (1) Improper use of CA generator.

knowing any p bits of the CA at any point of time ensures that one knows $p-2$ bits of the CA in the next time point. This is crucial and previously unobserved property of the CA. Compared to an LFSR, it is this property that makes CA much more susceptible to guess-then-determine attacks. This is a lesson on the secure CA usage. (2) The key stream is produced by XORing a portion of the linear and the nonlinear part. Further the nonlinear part is updated by mixing a separate portion of the linear part into it. While this mixing is necessary, the manner in which it is done is not correct. The linear part is simple XORed into the nonlinear part creating a weakness that can again be exploited in a guess-then-determine attack. (This property allows the recent algebraic attack on HBB.) On the other hand, SNOW also updates the nonlinear part by mixing with the linear part. But this mixing is effected by an addition modulo 2^{32}. In fact, as has been recently observed that if this addition is replaced by a XOR, SNOW also becomes weak and susceptible to algebraic attacks [6]. (3) Too much of the state is revealed by HBB. In order to achieve efficiency, the entire nonlinear part is mixed with a portion of the linear part to produce the 128-bit keystream block. Again this is an undesirable thing to do and makes the verification stage of the guess-then-determine attack easier. Thus, to develop a cipher using CA, a designer should avoid the above pitfalls.

If an LFSR is used instead of a CA, then the described attack will not hold. Whether the attack can be modified to also hold for LFSR is still an open problem. Implementation of the attack can be obtained from the author.

References

1. Palash Sarkar. Hiji-Bij-Bij: A New Stream Cipher with a Self-Synchronizing Mode of Operation. In T. Johansson and S. Maitra, editors, *Progress in Cryptology - INDOCRYPT'03, volume 2904 of Lecture Notes in Computer Science, pages 36-51, Springer Verlag.*
2. Antoine Joux and Frederic Muller: Two Attacks against the HBB Stream Cipher. In *FSE'05, pages 341-353.*
3. Vlastimil Klima: Cryptanalysis of Hiji-bij-bij (HBB). In *Cryptology ePrint Archive: Report 2005/003.*
4. P. Ekdahl and T. Johansson: A new version of the stream cipher SNOW. In K. Nyberg and H Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, volume 2595 of Lecture Notes in Computer Science, pages 47–61. Springer-Verlag, Berlin, 2003.*
5. G. Rose and P. Hawkes: Turing: a fast software stream cipher. In *Rump session of Crypto 2002, http://people.qualcomm.com/ggr/QC/Turing.tgz*
6. Olivier Billet, Henri Gilbert: Resistance of SNOW 2.0 Against Algebraic Attacks. In *CTRSA 2005, pages 19-28.*
7. J. Golic: Cryptanalysis of three mutually clock-controlled stop/go shift registers. In *IEEE Trans. Information Theory, vol. 46, pages 1081-1090, May 2000.*
8. J. Golic, A. Clark and E. Dawson: Generalized inversion attack on nonlinear filter generators. In *IEEE Trans. Computers, vol. 49, pages 1100-1109, Oct. 2000.*
9. J. Golic: Multibit cascades may be vulnerable to inversion attack. In *Electronics Letters, vol. 36(18), pages 1536-1538, Aug. 2000.*

Appendix A

Algorithm A1. Compute $L_{i,0}$ $0 \leq i \leq t$ from $\ell \| b_0 \dots b_{t-1}$

1. $\mathcal{R}_{0,t} \leftarrow$ most significant $32 + t$ bits of $\mathcal{R}_0$
2. $x \leftarrow \ell \| b_0 \dots b_{t-1}$; $L_{0,0} = \ell$;
3. **for** $i = 0$ to $t - 1$ **do**
4. $\quad x \leftarrow (x \ll 1) \oplus (x \gg 1) \oplus (x \wedge \mathcal{R}_{0,t})$;
5. $\quad L_{i+1,0} =$ most significant 32 bits of x ;
6. **end-do**

Algorithm A2. Compute $L_{t+2,0}$ from $\ell \| b_0 \dots b_{t+1}$

1. $x \leftarrow \ell \| b_0 \dots b_t b_{t+1}$;
2. **for** $i = 0$ to $t + 1$ **do**
3. $\quad x \leftarrow (x \ll 1) \oplus (x \gg 1) \oplus (x \wedge \mathcal{R}_{0,t+1})$;
4. **end-do**
5. $L_{t+2,0} =$ most significant 32 bits of x ;

Algorithm A3. Computation of $\mathcal{HN}^V_{t+1,0}$ from $\mathcal{HN}^V_{t,0}$

1. **for** $v1 = 0$ to 15 **do**
2. $\quad HD1(v1) \leftarrow \phi$ /* set initialized by ϕ */
3. **end-do**
4. $NZ_{t+1,0} = nz_{t+1,1} \| nz_{t+1,0}$ /* each sub-string has 16 bits */
5. $NZ_{t,0} = nz_{t,1} \| nz_{t,0}$ /* each sub-string has 16 bits */
6. **for** every $x \in \mathcal{HN}^V_{t,0}$ **do**
7. $\quad z \leftarrow m(f(x \oplus nz_{t,1}, r1)) \oplus m(nz_{t+1,1})$
8. $\quad$ **for** $(\delta_{31}, \delta_{16}) \in \{0,1\}^2$ **do**
9. $\quad\quad$ Set $z^* \leftarrow \delta_{31} \| z \| \delta_{16}$
10. $\quad\quad$ Add z^* to the set $HD1(g_3(z^*))$
11. $\quad$ **end-do**
12. **end-do**
13. $\mathcal{HN}^V_{t+1,0} \leftarrow HD1(g_4(NZ_{t+2,0}))$

Algorithm A4. Computation of $\mathcal{LN}^V_{t+1,0}$ from $\mathcal{LN}^V_{t,0}$

1. **for** $v1 = 0$ to 15 **do**
2. $\quad LD1(v1) \leftarrow \phi$ /* set initialized by ϕ */
3. **end-do**
4. $NZ_{t+1,0} = nz_{t+1,1} \| nz_{t+1,0}$ /* each sub-string has 16 bits */
5. $NZ_{t,0} = nz_{t,1} \| nz_{t,0}$ /* each sub-string has 16 bits */
6. **for** every $x \in \mathcal{LN}^V_{t,0}$ **do**
7. $\quad z \leftarrow m(f(x \oplus nz_{t,0}, r0)) \oplus m(nz_{t+1,0})$
8. $\quad$ **for** $(\delta_{15}, \delta_0) \in \{0,1\}^2$ **do**
9. $\quad\quad$ Set $z^* \leftarrow \delta_{15} \| z \| \delta_0$
10. $\quad\quad$ Add z^* to the set $LD1(g_3(z^*))$
11. $\quad$ **end-do**
12. **end-do**
13. $\mathcal{LN}^V_{t+1,0} \leftarrow LD1(g_5(NZ_{t+2,0}))$

Here, $r1$ and $r0$ are chosen suitably from $\mathcal{R}_0$ (Section 2).