A shortened version of this paper appears under the same title
in the proceedings of *Asiacrypt 2006* (Xuejia Lai, ed.), *LNCS*, Springer-Verlag.

On the (In)security of Stream Ciphers Based on Arrays and Modular Addition*

Souradyuti Paul and Bart Preneel

Katholieke Universiteit Leuven, Dept. ESAT/COSIC,
Kasteelpark Arenberg 10,
B–3001 Leuven-Heverlee, Belgium
{Souradyuti.Paul,Bart.Preneel}@esat.kuleuven.be

Abstract. Stream ciphers play an important role in symmetric cryptology because of their suitability in high speed applications where block ciphers fall short. A large number of fast stream ciphers or pseudorandom bit generators (PRBGs) can be found in the literature that are based on arrays and simple operations such as modular additions, rotations and memory accesses (e.g. RC4, RC4A, Py, Py6, ISAAC etc.). This paper investigates the security of array-based stream ciphers (or PRBGs) against certain types of distinguishing attacks in a unified way. We argue, counter-intuitively, that the most useful characteristic of an array, namely, the association of array-elements with unique indices, may turn out to be the origins of distinguishing attacks if adequate caution is not maintained. In short, an adversary may attack a cipher simply exploiting the dependence of array-elements on the corresponding indices. Most importantly, the weaknesses are not eliminated even if the indices and the array-elements are made to follow uniform distributions separately. Exploiting these weaknesses we build distinguishing attacks with reasonable advantage on five recent stream ciphers (or PRBGs), namely, Py6 (2005, Biham *et al.*), IA, ISAAC (1996, Jenkins Jr.), NGG, GGHN (2005, Gong *et al.*) with data complexities $2^{68.61}$, $2^{32.89}$, $2^{16.89}$, $2^{32.89}$ and $2^{32.89}$ respectively. In all the cases we worked under the assumption that the key-setup algorithms of the ciphers produced uniformly distributed internal states. We only investigated the mixing of bits in the keystream generation algorithms. In hindsight, we also observe that the previous attacks on the other array-based stream ciphers (e.g. Py, etc.), can also be explained in the general framework developed in this paper. We hope that our analyses will be useful in the evaluation of the security of stream ciphers based on arrays and modular addition.

Update (29th March 2008). The attack described in this paper for Py6 also applies as-is to its strengthened variant TPy6 [5].

* This work was supported in part by the Concerted Research Action (GOA) Ambiorix 2005/11 of the Flemish Government and in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.

1 Introduction

Stream ciphers are of paramount importance in fast cryptographic applications such as encryption of streaming data where information is generated at a high speed. Unfortunately, the state-of-the art of this type of ciphers, to euphemize, is not very promising as reflected in the failure of the NESSIE project to select a single cipher for its profile [14] and also the attacks on a number of submissions for the ongoing ECRYPT project [7]. Because of plenty of common features as well as dissimilarities, it is almost impossible to classify the entire gamut of stream ciphers into small, well-defined, disjoint groups, so that one group of ciphers can be analyzed in isolation of the others. However, in view of the identical data structures and similar operations in a number of stream ciphers and the fact that they are vulnerable against certain kinds of attacks originating from some basic flaws inherent in the design, it makes sense to scrutinize the class of ciphers in a unified way. As the title suggests, the paper takes a closer look at stream ciphers connected by a common feature that each of them uses (i) one or more arrays[1] as the *main* part of the internal state and (ii) the operation *modular addition* in the *pseudorandom bit generation algorithm*. Apart from *addition* over different *groups* (e.g, GF(2^n) and GF(2)), the stream ciphers under consideration only admit of simple operations such as memory access (direct and indirect) and cyclic rotation of bits, which are typical of any fast stream cipher. In the present discussion we omit the relatively rare class of stream ciphers which may nominally use *array* and *addition*, but their security depends significantly on special functions such as those based on algebraic hard problems, Rijndael S-box etc.

To the best of our knowledge, the RC4 stream cipher, designed by Ron Rivest in 1987, is the first stream cipher which exploits the features of an array in generating pseudorandom bits, using a few simple operations. Since then a large number of array-based ciphers or PRBGs – namely, RC4A [15], VMPC stream cipher [21], IA, IBAA, ISAAC [11], Py [2], Py6 [4], Pypy [3], HC-256 [19], NGG [13], GGHN [9] – have been proposed that are inspired by the RC4 design principles. The Scream family of ciphers [10] also uses arrays and modular additions in their round functions, however, the security of them hinges on a tailor-made function derived from Rijndael S-box rather than mixing of *additions* over different *groups* (e.g., GF(2^n) and GF(2)) and cyclic rotation of bits; therefore, this family of ciphers is excluded from the class of ciphers to be discussed in the paper.

First, in Table 1, we briefly review the pros and cons of the RC4 stream cipher which is the predecessor of all the ciphers to be analyzed later. Unfortunately, the RC4 cipher is compatible with the old fashioned 8-bit processors only. Except RC4A and the VMPC cipher (which are designed to work on 8-bit processors), all the other ciphers described before are suitable for modern 16/32-bit architectures. Moreover, those 16/32-bit ciphers have been designed with an ambition of

[1] An array is a data structure containing a set of elements associated with unique indices.

Table 1. Pros and cons of the RC4 cipher

Advantages of RC4	Disadvantages of RC4
Arrays allow for huge *secret* internal state	Not suitable for 16/32-bit architecture
Fast because of fewer operations per round	Several distinguishing attacks
Simple Design	Weak Key-setup algorithm
No key recovery attacks better than brute force	

incorporating all the positive aspects of RC4, while ruling out it's negative properties as listed in Table 1. However, the paper observes that a certain amount of caution is necessary to adapt RC4-like ciphers to 16/32-bit architecture. Here, we mount distinguishing attacks on the ciphers (or PRBGs) Py6, IA, ISAAC, NGG, GGHN – all of them are designed to suit 16/32-bit processors – with data $2^{68.61}$, $2^{32.89}$, $2^{16.89}$, $2^{32.89}$ and $2^{32.89}$ respectively, exploiting similar weaknesses in their designs (note that another 32-bit array-based cipher Py has already been attacked in a similar fashion [6, 16]). Summarily the attacks on the class of ciphers described in this paper originate from the following basic although not independent facts. However, note that our attacks are based on the assumptions that the key-setup algorithms of the ciphers are 'perfect', that is, after the execution of the algorithms they produce uniformly distributed internal states (more on that in Sect. 1.2).

– Array-elements are large (usually of size 16/32 bits), but the array-indices are short (generally of size 8 bits).
– Only a few elements of the arrays undergo changes in consecutive rounds.
– Usage of both pseudorandom index-pointers and pseudorandom array-elements in a round, which apparently seems to provide stronger security than the ciphers with fixed pointers, may leave room for attacks arising from the *correlation* between the index-pointers and the corresponding array-elements (see discussion in Sect. 2.2).
– Usage of simple operations like *addition* over GF(2^n) and GF(2) in output generation.

Essentially our attacks based on the above facts have it origins in the *fortuitous states* attack on RC4 by Fluhrer and McGrew [8].

A general framework to attack array-based stream ciphers with the above characteristics is discussed in Sect. 2. Subsequently in Sect. 3.1, 3.2 and 3.3, as concrete proofs of our argument, we show distinguishing attacks on five stream ciphers (or PRBGs). The purpose of the paper is, by no means, to claim that the array-based ciphers are intrinsically insecure, and therefore, should be rejected without analyzing its merits; rather, we stress that when such a PRBG turns out to be extremely fast – such as Py, Py6, IA, ISAAC, NGG, GGHN – an alert message should better be issued for the designers to recheck that they are free from the weaknesses described here. In Sect. 3.5, we comment on the security of three other array-based ciphers (or PRBGs) IBAA, Pypy and HC-256 which, for the moment, do not come under attacks, however they are slower than the ones attacked in this paper.

1.1 Notation and Convention

- The symbols $\oplus$, $+$, $-$, $\lll$, $\ggg$, $\gg$, $\ll$ are used as per convention.
- The ith bit of the variable X is denoted $X_{(i)}$ (the *lsb* is the 0th bit).
- The segment of $m - n + 1$ bits between the mth and the nth bits of the variable X is denoted by $X_{(m,n)}$.
- The abbreviation for Pseudorandom Bit Generator is PRBG.
- $P[A]$ denotes the probability of occurrence of the event A.
- E^c denotes the complement of the event E.
- At any round t, some part of the internal state is updated before the output generation and the rest is updated after that. Example: in Algorithm 3, the variables a and m are updated before the output generation in line 5. The variables i and b are updated after or at the same time with the output generation. Our convention is: a variable S is denoted by S_t at the time of output generation of round t. As each of the variables is modified in a single line of the corresponding algorithm, after the modification its subscript is incremented.

1.2 Assumption

In this paper we concentrate solely on the *mixing of bits* by the keystream generation algorithms (i.e., PRGB) of several array-based stream ciphers and assume that the corresponding key-setup algorithms are *perfect*. A *perfect* key-setup algorithm produces internal state that leaks no statistical information to the attacker. In other words, because of the *difficulty* of deducing any relations between the inputs and outputs of the key-setup algorithm, the internal state produced by the key-setup algorithm is assumed to follow the uniform distribution.

2 Stream Ciphers Based on Arrays and Modular Addition

2.1 Basic Working Principles

The basic working principle of the PRBG of a stream cipher, based on one or multiple arrays, is shown in Fig. 1. For simplicity, we take snapshots of the internal state, composed of *only* two arrays, at two close rounds denoted by round t and round $t' = t + \delta$. However, our analysis is still valid with more arrays and rounds than just two. Now we delineate the rudiments of the PRBG of such ciphers.

- **Components:** the internal state of the cipher comprises all or part of the following components.
  1. One or more arrays of n-bit elements (X_1 and X_2 in Fig. 1).
  2. One or more variables for indexing into the arrays, i.e., the index-pointers (down arrows in Fig. 1).

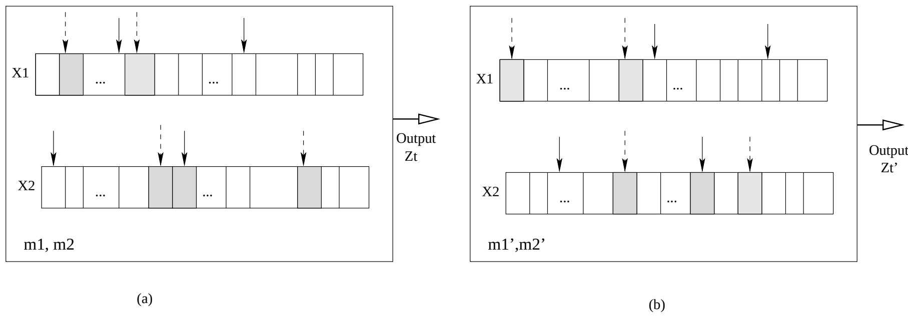

Fig. 1. Internal State at (a) round t and (b) round $t' = t + \delta$

3. One or more random variables usually of n-bit length (m_1, m_2, m'_1, m'_2 in Fig. 1).

– **Modification to the Internal State at a round.**
  1. *Index Pointers:* the most notable feature of such ciphers is that it has two sets of index pointers. (i) Some of them are fixed or updated in a *known way*, i.e., independent of the secret part of the state (solid arrows in Fig. 1) and (ii) the other set of pointers are updated *pseudorandomly*, i.e., based on one or more secret components of the internal state (dotted arrows in Fig. 1).
  2. *Arrays*: a few elements of the arrays are updated pseudorandomly based on one or more components of the internal state (the shaded cells of the arrays in Fig. 1). Note that, in two successive rounds, only a small number of array-elements (e.g. one or two in each array) are updated. Therefore, most of the array-elements remain identical in consecutive rounds.
  3. *Other variables if any*: they are updated using several components of the internal state.

– **Output generation:** the output generation function at a round is a non-linear combination of different components described above.

2.2 Weaknesses and General Attack Scenario

Before assessing the security of array-based ciphers in general, for easy understanding, we first deal with a simple toy-cipher with certain properties which induce distinguishing attack on it. Output at round t is denoted by Z_t.

Remark 1. The basis for the attacks described throughout the paper including the one in the following example is searching for internal states for which the outputs can be predicted with bias. This strategy is inspired by the *fortuitous states* attacks by Fluhrer and McGrew on the RC4 stream cipher [8].

Example 1. Let the size of the *internal state* of a stream cipher with the following properties be k bits.

Property 1. The outputs Z_{t_1}, Z_{t_2} are as follows.

$$Z_{t_1} = X \oplus Y + (A \lll B)\,, \tag{1}$$
$$Z_{t_2} = M + N \oplus (C \lll D) \tag{2}$$

where X, Y, A, B, M, N, C, D are uniformly distributed and independent.

Property 2. [Bias-inducing State] If certain k' bits $(0 < k' \leq k)$ of the *internal state* are set to all 0's (denote the occurrence of such state by event E) at round t_1, then the following equations hold good.

$$X = M,\ Y = N,\ B = D = 0,\ A = C.$$

Therefore, (1) and (2) become

$$Z_{t_1} = X \oplus Y + A\,,\ \ Z_{t_2} = X + Y \oplus A\,.$$

Now, it follows directly form the above equations that, for a fraction of $2^{-k'}$ of all *internal states*,

$$P[Z_{(0)} = (Z_{t_1} \oplus Z_{t_2})_{(0)} = 0|E] = 1. \tag{3}$$

Property 3. If the *internal state* is chosen randomly from the rest of the states, then

$$P[Z_{(0)} = 0|E^c] = \frac{1}{2}. \tag{4}$$

Combining (3) and (4) we get the overall bias for $Z_{(0)}$,

$$\begin{aligned} P[Z_{(0)} = 0] &= \frac{1}{2^{k'}} \cdot 1 + (1 - \frac{1}{2^{k'}}) \cdot \frac{1}{2} \\ &= \frac{1}{2}(1 + \frac{1}{2^{k'}}) \end{aligned} \tag{5}$$

Note that, if the cipher were a secure PRBG then $P[Z_{(0)} = 0] = \frac{1}{2}$. □

Discussion. Now we argue that an array-based cipher has all the three properties of the above example; therefore, the style of attack presented in the example can possibly be applied to an array-based cipher too. First, we discuss the operations involved in the output generation of the PRBG. Let the internal state consist of N arrays and M other variables. At round t, the arrays are denoted by $S_{1,t}[\cdot]$, $S_{2,t}[\cdot], \cdots$, $S_{N,t}[\cdot]$ and the variables by $m_{1,t}$, $m_{2,t}, \cdots$, $m_{M,t}$. We observe that the output Z_t is of the following form,

$$\begin{aligned} Z_t = \mathsf{ROT}[\cdots \mathsf{ROT}[\mathsf{ROT}[\mathsf{ROT}[V_{1,t}] \circledast \mathsf{ROT}[V_{2,t}]] \\ \circledast \mathsf{ROT}[V_{3,t}]] \circledast \cdots \circledast \mathsf{ROT}[V_{k,t}]] \end{aligned} \tag{6}$$

where $V_{i,t} = m_{g,t}$ or $S_{j,t}[I_l]$; $\mathsf{ROT}[\cdot]$ is the cyclic rotation function either constant or variable depending on the secret state; the function $\circledast[\cdot,\cdot]$ is either *bit-wise* XOR or *addition* modulo 2^n.

Now we describe a general technique to establish a distinguishing attack on an array-based cipher from the above information. We recall that, at the first round (round t_1 in the present context), the internal state is assumed to be uniformly distributed (see Sect. 1.2).

Step 1. [Analogy with *Property 1* of *Example 1*] Observe the elements of the internal state which are involved in the outputs Z_{t_1}, $Z_{t_2}, \cdots$ (i.e., the $V_{i,t}$'s in (6)) when the rounds in question are close ($t_1 < t_2 < \cdots$).

Step 2. [Bias-inducing state, Analogy with *Property 2* of *Example 1*] Fix a few bits of some array elements (or fix a relation among them) at the initial round t_1 such that *indices* of array-elements in later rounds can be predicted with probability 1 or close to it. More specifically, we search for a *partially specified internal state* such that one or both of the following cases occur due to predictable *index-pointers*.

1. The $V_{i,t}$'s involved in Z_{t_1}, $Z_{t_2}, \cdots$ are those array-elements whose bits are already fixed.
2. Each $V_{i,t}$ is dependent on one or more other variables in Z_{t_1}, $Z_{t_2}, \cdots$.

Now, for this case, we compute the bias in the output bits. Below we identify the reasons why an array-based cipher can potentially fall into the above scenarios.

Reason 1. Usually, an array-based cipher uses a number of pseudorandom index-pointers which are updated by the elements of the array. This fact turns out to be a weakness, as fixed values (or a relation) can be assigned to the array-elements such that the index-pointers fetch values from known locations. In other words, the weakness results from the correlation between index-pointers and array-elements which are, although, uniformly distributed individually but not independent of each other.

Reason 2. Barring a few, most of the array-elements do not change in rounds which are close to each other. Therefore, by fixing bits, it is sometimes easy to force the pseudorandom index-pointers fetch certain elements from the arrays in successive rounds.

Reason 3. The size of an index-pointer is small, usually 8 bits irrespective of the size of an array-element which is either 16 bits or 32 bits or 64 bits. Therefore, fixing a small number of bits of the array-elements, it is possible to assign appropriate values to the index-pointers. The less the number of fixed bits, the greater is the bias (note the parameter k' in (5)).

Reason 4. If the rotation operations in the output function are determined by pseudorandom array elements (see (6)) then fixing a few bits of internal state can simplify the function by freeing it from rotation operations. In many cases rotation operations are not present in the function. In any case the output function takes the following form.

$$Z_t = V_{1,t} \circledast V_{2,t} \circledast V_{3,t} \circledast \cdots \circledast V_{k,t}.$$

Irrespective of whether '$\circledast$' denotes '$\oplus$' or '$+$', the following equation holds for the *lsb* of Z_t.

$$Z_{t(0)} = V_{1,t_{(0)}} \oplus V_{2,t_{(0)}} \oplus V_{3,t_{(0)}} \oplus \cdots \oplus V_{k,t_{(0)}}.$$

Now by adjusting the index-pointers through fixing bits, if certain equalities among the $V_{i,t}$'s are ensured then $\bigoplus\limits^{t} Z_{t(0)} = 0$ occurs with probability 1 rather than probability 1/2.

Step 3. [Analogy with *Property 3* of *Example 1*] Prove or provide strong evidence that, for the rest of the states other than the bias-inducing state, the bias generated in the previous step is not counterbalanced.

Reason. The internal state of such cipher is huge and uniformly distributed at the initial round. The correlation, detected among the indices and array-elements in Step 2, is fortuitous although not entirely surprising because the variables are not independent. Therefore, the possibility that a bias, produced by an accidental *state*, is *totally* counterbalanced by another accidental state is negligible. In other words, if the bias-inducing state, as explained in Step 2, does not occur, it is likely that at least one of the $V_{i,t}$'s in (6) is uniformly distributed and independent; this fact ensures that the outputs are also uniformly distributed and independent.

Step 4. [Analogy with (5) of *Example 1*] Estimate the overall bias from the results in Step 2 and 3. □

In the next section, we attack several array-based ciphers following the methods described in this section.

3 Distinguishing Attacks on Array-based Ciphers or PRBGs

This section describes distinguishing attacks on the ciphers (or PRBGs) Py6, IA, ISAAC, NGG and GGHN – each of which is based on *arrays* and *modular addition*. Due to space constraints, full description of the ciphers is omitted; the reader is kindly referred to the corresponding design papers for details. For each of the ciphers, our task is essentially two-forked as summed up below.

1. **Identification of a Bias-inducing State.** This state is denoted by the event E which adjusts the *index-pointers* in such a way that the *lsb*'s of the outputs are biased. The *lsb*'s of the outputs are potentially vulnerable as

they are generated without any carry bits which are nonlinear combinations of input bits (see Step 2 of the general technique described in Sect. 2.2).
2. **Computation of the Probability of Overall Bias.** The probability is calculated considering both E and E^c. As suggested in Step 3 of Sect. 2.2, for each cipher, the *lsb*'s of the outputs are uniformly distributed if the event E does not occur under the assumption mentioned in Sect. 1.2.

Note. For each of the five ciphers attacked in the subsequent sections, it can be shown that, if E (i.e., the bias-inducing state) does not occur then the variable under investigation is uniformly distributed under the assumption of uniformly distributed *internal state* after the key-setup algorithm. We omit those proofs due to space constraints.

3.1 Bias in the outputs of Py6

The stream cipher Py6, designed especially for fast software applications by Biham and Seberry in 2005, is one of the modern ciphers that are based on arrays [2, 4].[2] Although the cipher Py, a variant of Py6, was successfully attacked [16, 6], Py6 has so far remained alive. The PRBG of Py6 is described in Algorithm 1 (see [2, 4] for a detailed discussion).

Algorithm 1 Single Round of Py6

Input: $Y[-3, ..., 64]$, $P[0, ..., 63]$, a 32-bit variable s
Output: 64-bit random output
/*Update and rotate P*/
1: swap ($P[0]$, $P[Y[43]\&63]$);
2: rotate (P);
/* Update s*/
3: $s+=Y[P[18]]-Y[P[57]]$;
4: $s=ROTL32(s,((P[26]+18)\&31))$;
/* Output 8 bytes (least significant byte first)*/
5: output $((ROTL32(s,25)\oplus Y[64])+Y[P[8]])$;
6: output $((\qquad s \qquad \oplus Y[-1])+Y[P[21]])$;
/* Update and rotate Y*/
7: $Y[-3]=(ROTL32(s,14)\oplus Y[-3])+Y[P[48]]$;
8: rotate(Y);

Bias-producing State of Py6. Below we identify six conditions among the elements of the S-box P, for which the distribution of $Z_{1,1}\oplus Z_{2,3}$ is biased ($Z_{1,t}$ and $Z_{2,t}$ denote the lower and upper 32 bits of output respectively, at round t).

C1. $P_2[26]\equiv -18 (\bmod 32)$; **C2.** $P_3[26]\equiv 7 (\bmod 32)$; **C3.** $P_2[18]=P_3[57]+1$; **C4.** $P_2[57]=P_3[18]+1$; **C5.** $P_1[8]=1$; **C6.** $P_3[21]=62$.

[2] The cipher has been submitted to the ECRYPT Project [7].

Let the event E denote the simultaneous occurrence of the above conditions ($P[E] \approx 2^{-33.86}$). It can be shown that, if E occurs then $Z_{(0)} = 0$ where Z denotes $Z_{1,1} \oplus Z_{2,3}$ (see Appendix A). Now we calculate the probability of occurrence of $Z_{(0)}$.

$$\begin{aligned} P[Z_{(0)} = 0] &= P[Z_{(0)} = 0|E] \cdot P[E] + P[Z_{(0)} = 0|E^c] \cdot P[E^c] \\ &= 1 \cdot 2^{-33.86} + \frac{1}{2} \cdot (1 - 2^{-33.86}) \\ &= \frac{1}{2} \cdot (1 + 2^{-33.86}) \,. \end{aligned} \tag{7}$$

Note that, if Py6 had been an ideal PRBG then the above probability would have been exactly $\frac{1}{2}$.

Remark 2. The above bias can be generalized for rounds t and $t+2$ ($t > 0$) rather than only rounds 1 and 3.

Remark 3. The main difference between Py and Py6 is that the locations of S-box elements used by one cipher is different from those by the other. The significance of the above results is that it shows that changing the locations of array-elements is futile if the cipher retains some intrinsic weaknesses as explained in Sect. 2.2. Note that Py was attacked with $2^{84.7}$ data while Py6 is with $2^{68.61}$ (explained in Sect. 3.4)

3.2 Biased outputs in IA and ISAAC

At FSE 1996, R. Jenkins Jr. proposed two fast PRBGs, namely IA and ISAAC, along the lines of the RC4 stream cipher [11]. The round functions of IA and ISAAC are shown in Algorithm 2 and Algorithm 3. Each of them uses an array of 256 elements. The size of an array-element is 16 bits for IA and 32 bits for ISAAC. However, IA and ISAAC can be adapted to work with array-elements of larger size too. For IA, this is the first time that an attack is proposed. For ISAAC, the earlier attack was by Pudovkina who claimed to have deduced its internal state with time $4 \cdot 67 \cdot 10^{1240}$ which was way more than the exhaustive search through the keys of usual size of 256-bit or 128-bit [18]. On the other hand, we shall see later in Sect. 3.4 that our distinguishing attacks can be built with much lower time complexities. The Z_t denotes the output at round t.

Bias-inducing State of IA. Let $m_t[i_t + 1 \bmod 256] = a$. If the following condition

$$ind((a + Z_t) \gg 8) = ind(a) = i_{t+1} \tag{8}$$

is satisfied then

$$Z_{(0)}(= Z_{t\,(0)} \oplus Z_{t+1\,(0)}) = 0$$

Algorithm 2 PRBG of IA

Input: $m[0, 1, ...255]$, 16-bit random variable b
Output: 16-bit random output
1: $i = 0$;
2: $x = m[i]$;
3: $m[i] = y = m[ind(x)] + b \bmod 2^{16}$; /* $ind(x) = x_{(7,0)}$ */
4: Output= $b = m[ind(y \gg 8)] + x \bmod 2^{16}$;
5: $i = i + 1 \bmod 256$;
6: Go to step 2;

A pictorial description of the state is provided in Fig. 3 of Appendix B. Let event E occur when (8) holds good. Note that $P[E] = 2^{-16}$ assuming a and Z_t are independent and uniformly distributed. Therefore,

$$\begin{aligned} P[Z_{(0)} = 0] &= P[Z_{(0)} = 0|E] \cdot P[E] + P[Z_{(0)} = 0|E^c] \cdot P[E^c] \\ &= 1 \cdot 2^{-16} + \frac{1}{2} \cdot (1 - 2^{-16}) \\ &= \frac{1}{2} \cdot (1 + 2^{-16}) \,. \end{aligned} \tag{9}$$

Algorithm 3 PRBG of ISAAC

Input: $m[0, 1, ...255]$, two 32-bit random variables a and b
Output: 32-bit random output
1: $i = 0$;
2: $x = m[i]$;
3: $a = a \oplus (a \ll R) + m[i + K \bmod 256] \bmod 2^{32}$;
4: $m[i + 1] = y = m[ind(x)] + a + b \bmod 2^{32}$; /* $ind(x) = x_{(9,2)}$ */
5: Output= $b = m[ind(y \gg 8)] + x \bmod 2^{32}$;
6: $i = i + 1 \bmod 256$;
7: Go to Step 2.

Bias-inducing State of ISAAC. For easy understanding, we rewrite the PRBG of the ISAAC in a simplified manner in Algorithm 3. The variables R and K, described in step 3 of Algorithm 3, depend on the parameter i (see [11] for details); however, we show that our attack can be built independent of those variables.

Let $m_{t-1}[i_t] = x$. Let event E occur when the following equation is satisfied.

$$ind((m_{t-1}[ind(x)] + a_t + b_{t-1}) \gg 8) = i_t. \tag{10}$$

If E occurs then $Z_t = x + x \bmod 2^{32}$, i.e., $Z_{t(0)} = 0$ (see Appendix C for a proof). As a_t, b_{t-1} and x are independent and each of them is uniformly distributed over

$\mathbb{Z}_{2^{32}}$, the following equation captures the bias in the output.

$$\begin{aligned}
P[Z_{t(0)} = 0] &= P[Z_{t(0)} = 0|E] \cdot P[E] + P[Z_{t(0)} = 0|E^c] \cdot P[E^c] \\
&= 1 \cdot 2^{-8} + \frac{1}{2} \cdot (1 - 2^{-8}) \\
&= \frac{1}{2} \cdot (1 + 2^{-8})\,. \qquad (11)
\end{aligned}$$

3.3 Biases in the outputs of NGG and GGHN

Gong *et al.* very recently have proposed two array-based ciphers NGG and GGHN with 32/64-bit word-length [13, 9] for very fast software applications. The PRBGs of the ciphers are described in Algorithm 4 and Algorithm 5. Both the ciphers are claimed to be more than three times as fast as RC4. Due to the introduction of an extra 32-bit random variable k, the GGHN is evidently a stronger version of NGG. We propose attacks on both the ciphers based on the general technique described in Sect. 2.2. Note that the NGG cipher was already experimentally attacked by Wu without theoretical quantification of the attack parameters such as bias, required outputs [20]. For NGG, our attack is new, theoretically justifiable and most importantly, conforms to the basic weaknesses of an array-based cipher, as explained in Sect. 2.2. For GGHN, our attack is the first attack on the cipher. In the following discussion, the Z_t denotes the output at round t.

Algorithm 4 Pseudorandom Bit Generation of NGG

Input: $S[0, 1, ...255]$
Output: 32-bit random output

1: $i = 0$, $j = 0$;
2: $i = i + 1 \bmod 256$;
3: $j = j + S[i] \bmod 256$;
4: Swap ($S[i]$, $S[j]$);
5: Output= $S[S[i] + S[j] \bmod 256]$;
6: $S[S[i] + S[j] \bmod 256] = S[i] + S[j] \bmod 2^{32}$
7: Go to step 2;

Bias-inducing State of NGG. Let the event E occur, if $i_t = j_t$ and $S_{t+1}[i_{t+1}] + S_{t+1}[j_{t+1}] = 2 \cdot S_t[i_t] \bmod 256$. We observe that, if E occurs then $Z_{t+1(0)} = 0$ (see Appendix D). Now we compute $P[Z_{t+1(0)} = 0]$ where $P[E] = 2^{-16}$.

$$\begin{aligned}
P[Z_{t+1(0)} = 0] &= P[Z_{t+1(0)} = 0|E] \cdot P[E] + P[Z_{t+1(0)} = 0|E^c] \cdot P[E^c] \\
&= 1 \cdot 2^{-16} + \frac{1}{2} \cdot (1 - 2^{-16}) \\
&= \frac{1}{2} \cdot (1 + 2^{-16})\,. \qquad (12)
\end{aligned}$$

Algorithm 5 Pseudorandom Bit Generation of GGHN

Input: $S[0, 1, ...255]$, k
Output: 16-bit random output
1: $i = 0$, $j = 0$;
2: $i = i + 1 \bmod 256$;
3: $j = j + S[i] \bmod 256$;
4: $k = k + S[j] \bmod 2^{32}$;
5: Output$= S[S[i] + S[j] \bmod 256] + k \bmod 2^{32}$;
6: $S[S[i] + S[j] \bmod 256] = k + S[i] \bmod 2^{32}$;
7: Go to step 2;

Bias-producing State of GGHN. If $S_t[i_t] = S_{t+1}[j_{t+1}]$ and $S_t[j_t] = S_{t+1}[i_{t+1}]$ (denote it by event E) then $Z_{t+1(0)} = 0$ (see Appendix E). Now we compute $P[Z_{t+1(0)} = 0]$ where $P[E] = 2^{-16}$.

$$\begin{aligned} P[Z_{t+1(0)} = 0] &= P[Z_{t+1(0)} = 0|E] \cdot P[E] + P[Z_{t+1(0)} = 0|E^c] \cdot P[E^c] \\ &= 1 \cdot 2^{-16} + \frac{1}{2} \cdot (1 - 2^{-16}) \\ &= \frac{1}{2} \cdot (1 + 2^{-16}) \,. \end{aligned} \tag{13}$$

3.4 Data and Time of the Distinguishing Attacks

In the section we compute the data and time complexities of the distinguishers derived from the biases computed in the previous sections. A *distinguisher* is an algorithm which distinguishes a stream of bits from a perfectly random stream of bits, that is, a stream of bits that has been chosen according to the uniform distribution. The *advantage* of a distinguisher is the measure of its success rate (see [1] for a detailed discussion).

Let there be n binary random variables $z_1, z_2, \cdots, z_n$ which are independent of each other and each of them follows the distribution $\mathsf{D_{BIAS}}$. Let the uniform distribution on alphabet $\mathbb{Z}_2$ be denoted by $\mathsf{D_{UNI}}$. Method to construct an *optimal distinguisher* with a fixed number of samples is given in [1].[3] While the detailed description of an *optimal distinguisher* is omitted, the following theorem determines the number of samples required by an *optimal distinguisher* to attain an advantage of 0.5 which is considered a reasonable goal.

Theorem 1. *Let the input to an* optimal distinguisher *be a realization of the binary random variables $z_1, z_2, z_3, \cdots, z_n$ where each z_i follows $\mathsf{D_{BIAS}}$. To attain an advantage of more than 0.5, the least number of samples required by the*

[3] Given a fixed number of samples, an *optimal distinguisher* attains the maximum advantage.

Table 2. Data and time of the distinguishers with advantage exceeding 0.5

PRBG	M	Bytes of a single stream $= 0.4624 \cdot M^2$	Time
Py6	$2^{34.86}$	$2^{68.61}$	$O(2^{68.61})$
IA	2^{17}	$2^{32.89}$	$O(2^{32.89})$
ISAAC	2^{9}	$2^{16.89}$	$O(2^{16.89})$
NGG	2^{17}	$2^{32.89}$	$O(2^{32.89})$
GGHN	2^{17}	$2^{32.89}$	$O(2^{32.89})$

optimal distinguisher *is given by the following formula*

$$n = 0.4624 \cdot M^2 \quad \textit{where}$$

$$P_{\mathsf{D}_{BIAS}}[z_i = 0] - P_{\mathsf{D}_{UNI}}[z_i = 0] = \frac{1}{M} \,.$$

Proof. See Sect. 5 of [16] for the proof.

Now $\mathsf{D}_{\mathrm{UNI}}$ is known and $\mathsf{D}_{\mathrm{BIAS}}$ can be determined from (7) for Py6, (9) for IA, (11) for ISAAC, (12) for NGG, (13) for GGHN. In Table 2, we list the data and time complexities of the distinguishers. Our experiments agree well with the theoretical results. The constant in $O(m)$ is determined by the time taken by single round of the corresponding cipher.

3.5 A Note on IBAA, Pypy and HC-256

IBAA, Pypy and HC-256 are the array-oriented ciphers/PRBGs which are still free from any attacks. The IBAA works in a similar way as the ISAAC works, except for the variable a which plays an important role in the output generation of IBAA [11]. It seems that a relation has to be discovered among the values of the parameter a at different rounds to successfully attack IBAA. Pypy is a slower variant of Py and Py6 [3]. Pypy produces 32 bits per round when each of Py and Py6 produces 64 bits. To attack Pypy a relation need to be found among the elements which are separated by at least three rounds. To attack HC-256 [19], some correlations need to be known among the elements which are cyclically rotated by constant number of bits.

4 Conclusion

In this paper, we have studied array-based stream ciphers or PRBGs in a general framework to assess their resistance against certain distinguishing attacks originating from the correlation between index-pointers and array-elements. We show that the weakness becomes more profound because of the usage of simple modular additions in the output generation function. In the unified framework we have attacked five modern array-based stream ciphers (or PRBGs) Py6, IA, ISAAC, NGG, GGHN with data complexities $2^{68.61}$, $2^{32.89}$, $2^{16.89}$, $2^{32.89}$ and $2^{32.89}$ respectively. We also note that some other array-based stream ciphers (or PRBGs) IBAA, Pypy, HC-256 still do not come under any threats, however, the algorithms need to be analyzed more carefully in order to be considered secure. We believe that our investigation will throw light on the security of array-based stream ciphers in general and can possibly be extended to analyze other types of ciphers too.

5 Acknowledgments

We thank Jongsung Kim, Hongjun Wu, Gautham Sekar for useful discussions. We also acknowledge the constructive comments of the anonymous reviewers of Asiacrypt 2006.

References

1. T. Baignères, P. Junod and S. Vaudenay, "How Far Can We Go Beyond Linear Cryptanalysis?," *Asiacrypt 2004* (P. Lee, ed.), vol. 3329 of *LNCS*, pp. 432–450, Springer-Verlag, 2004.
2. E. Biham, J. Seberry, "Py (Roo): A Fast and Secure Stream Cipher using Rolling Arrays," eSTREAM, ECRYPT Stream Cipher Project, Report 2005/023, 2005.
3. Eli Biham and Jennifer Seberry, "Pypy: Another Version of Py," eSTREAM, ECRYPT Stream Cipher Project, Report 2006/038, 2006.
4. E. Biham, J. Seberry, "C Code of Py6," as available from `http://www.ecrypt.eu.org/stream/py.html`, eSTREAM, ECRYPT Stream Cipher Project, 2005.
5. E. Biham, J. Seberry, "Tweaking the IV Setup of the Py Family of Ciphers – The Ciphers Tpy, TPypy, and TPy6," Published on the author's webpage at `http://www.cs.technion.ac.il/ biham/`, January 25, 2007.
6. P. Crowley, "Improved Cryptanalysis of Py," *Workshop Record of SASC 2006 – Stream Ciphers Revisited*, ECRYPT Network of Excellence in Cryptology, February 2006, Leuven (Belgium), pp. 52–60.
7. Ecrypt, `http://www.ecrypt.eu.org`.
8. S. Fluhrer, D. McGrew, "Statistical Analysis of the Alleged RC4 Keystream Generator," *Fast Software Encryption 2000* (B. Schneier, ed.), vol. 1978 of *LNCS*, pp. 19-30, Springer-Verlag, 2000.
9. G. Gong, K. C. Gupta, M. Hell, Y. Nawaz, "Towards a General RC4-Like Keystream Generator," *First SKLOIS Conference, CISC 2005* (D. Feng, D. Lin, M. Yung, eds.), vol. 3822 of *LNCS*, pp. 162–174, Springer-Verlag, 2005.
10. S. Halevi, D. Coppersmith, C. S. Jutla, "Scream: A Software-Efficient Stream Cipher," *Fast Software Encryption 2002* (J. Daemen and V. Rijmen, eds.), vol. 2365 of *LNCS*, pp. 195–209, Springer-Verlag, 2002.
11. Robert J. Jenkins Jr., "ISAAC," *Fast Software Encryption 1996* (D. Gollmann, ed.), vol. 1039 of *LNCS*, pp. 41–49, Springer-Verlag, 1996.

12. I. Mantin, A. Shamir, "A Practical Attack on Broadcast RC4," *Fast Software Encryption 2001* (M. Matsui, ed.), vol. 2355 of *LNCS*, pp. 152–164, Springer-Verlag, 2001.
13. Y. Nawaz, K. C. Gupta, and G. Gong, "A 32-bit RC4-like Keystream Generator," Cryptology ePrint Archive, 2005/175.
14. NESSIE: New European Schemes for Signature, Integrity and Encryption, `http://www.cryptonessie.org`.
15. Souradyuti Paul, Bart Preneel, "A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher," *Fast Software Encryption 2004* (B. Roy, ed.), vol. 3017 of *LNCS*, pp. 245–259, Springer-Verlag, 2004.
16. Souradyuti Paul, Bart Preneel, Gautham Sekar, "Distinguishing Attacks on the Stream Cipher Py," *Fast Software Encryption 2006* (M. Robshaw, ed.), vol. 4047 of *LNCS*, Springer-Verlag, pp. 405-421, 2006 (to appear).
17. Souradyuti Paul, Bart Preneel, "On the (In)security of Stream Ciphers Based on Arrays and Modular Addition (Full Version)," Cryptology ePrint Archive: Report 2005/448, IACR, 2005, Available online at `http://eprint.iacr.org/2005/448`.
18. Marina Pudovkina, "A known plaintext attack on the ISAAC keystream generator," *Cryptology ePrint Archive: Report 2001/049,* IACR, 2001.
19. H. Wu, "A New Stream Cipher HC-256," *Fast Software Encryption 2004* (B. Roy, ed.), vol. 3017 of *LNCS*, pp. 226–244, Springer-Verlag, 2004.
20. H. Wu, "Cryptanalysis of a 32-bit RC4-like Stream Cipher," Cryptology ePrint Archive, 2005/219.
21. Bartosz Zoltak, "VMPC One-Way Function and Stream Cipher," *Fast Software Encryption 2004* (B. Roy, ed.), vol. 3017 of *LNCS*, pp. 210–225, Springer-Verlag, 2004.

A Bias-inducing State of Py6

Claim. $Z_{1,1(0)} = Z_{2,3(0)}$ if the following six conditions on the elements of the S-box P are simultaneously satisfied.

1. $P_2[26] \equiv -18 (\bmod 32)$,
2. $P_3[26] \equiv 7 (\bmod 32)$,
3. $P_2[18] = P_3[57] + 1$,
4. $P_2[57] = P_3[18] + 1$,
5. $P_1[8] = 1$,
6. $P_3[21] = 62$.

Proof. The formulas for the $Z_{1,1}$, $Z_{2,3}$ and s_2 are given below (see Algorithm 1).

$$Z_{1,1} = (ROTL32(s_1, 25) \oplus Y_1[64]) + Y_1[P_1[8]]\,, \tag{14}$$

$$Z_{2,3} = (s_3 \oplus Y_3[-1]) + Y_3[P_3[21]]\,, \tag{15}$$

$$s_2 = ROTL32(s_1 + Y_2[P_2[18]] - Y_2[P_2[57]], ((P_2[26] + 18) \bmod 32))\,. \tag{16}$$

• Condition 1 (i.e., $P_2[26] \equiv -18 (\bmod 32)$) reduces (16) to

$$s_2 = s_1 + Y_2[P_2[18]] - Y_2[P_2[57]]\,.$$

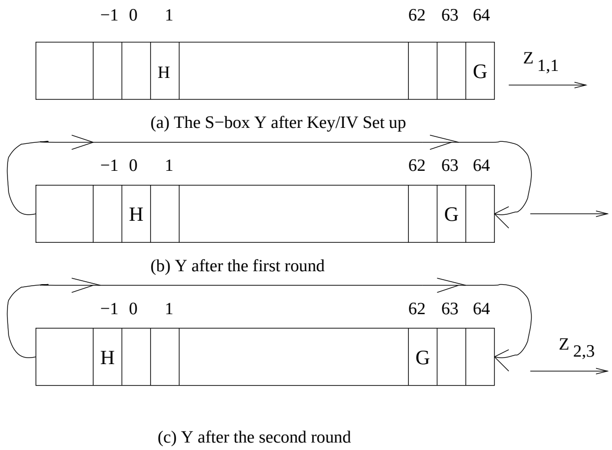

Fig. 2. Py6: (a) $P_1[8] = 1$ (condition 5): G and H are used in $Z_{1,1}$, (b) Y_2 (i.e., Y after the 1^{st} round), (c) $P_3[21] = 62$ (condition 6): G and H are used in $Z_{2,3}$

- Condition 2 (i.e., $P_3[26] \equiv 7(\bmod 32)$) together with Condition 1 implies

$$s_3 = ROTL32((s_1 + Y_2[P_2[18]] - Y_2[P_2[57]] + Y_3[P_3[18]] - Y_3[P_3[57]]), 25).$$

- Condition 3 and Condition 4 (that is, $P_2[18] = P_3[57]+1$ and $P_2[57] = P_3[18]+1$) reduce the previous equation to

$$s_3 = ROTL32(s_1, 25). \tag{17}$$

From (14), (15), (17) we get:

$$Z_{1,1} = (ROTL32(s_1, 25) \oplus Y_1[64]) + Y_1[P_1[8]], \tag{18}$$
$$Z_{2,3} = (ROTL32(s_1, 25) \oplus Y_3[-1]) + Y_3[P_3[21]]. \tag{19}$$

In Fig. 2, conditions 5 and 6 are described. According to the figure,

$$H = Y_1[P_1[8]] = Y_3[-1], \tag{20}$$
$$G = Y_1[64] = Y_3[P_3[21]]. \tag{21}$$

Applying (20) and (21) in (18) and (19) we get,

$$Z_{1,1(0)} \oplus Z_{2,3(0)} = Y_1[64]_{(0)} \oplus Y_1[P_1[8]]_{(0)} \oplus Y_3[-1]_{(0)} \oplus Y_3[P_3[21]]_{(0)} = 0.$$

This completes the proof. □

B Bias-inducing State of IA

Claim. Let $m_t[i_t + 1 \bmod 256] = a$. If the following condition

$$ind((a + Z_t) \gg 8) = ind(a) = i_{t+1}$$

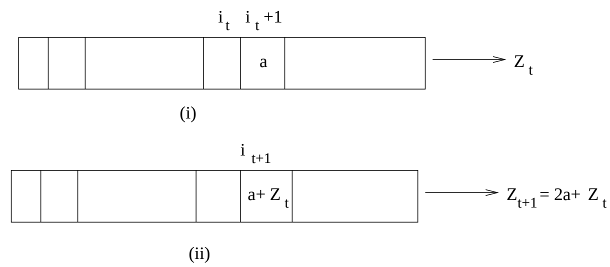

Fig. 3. IA: (i) at round t when $m_t[i_t + 1] = a$, (ii) at round $t + 1$

is satisfied then

$$Z_{t+1} = 2 \cdot a + Z_t \bmod 2^{16} \Rightarrow Z_{t(0)} \oplus Z_{t+1(0)} = 0\,.$$

Proof. Proof is easy from Algorithm 2. In Fig. 3, the configuration of the array m and the output are shown for two consecutive rounds when the condition is satisfied. □

C Bias-inducing State of ISAAC

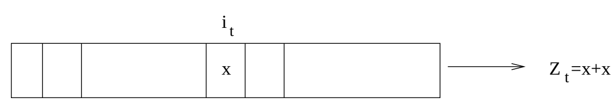

Fig. 4. ISAAC: at round t

Claim. Let $m_{t-1}[i_t] = x$. If the following condition

$$ind((m_{t-1}[ind(x)] + a_t + b_{t-1}) \gg 8) = i_t \tag{22}$$

is satisfied then

$$Z_t = x + x \bmod 2^{32} \Rightarrow Z_{t(0)} = 0\,. \tag{23}$$

Proof. The claim can be easily verified from Algorithm 3. In Fig. 4, the configuration of the *internal state* at round t is shown. □

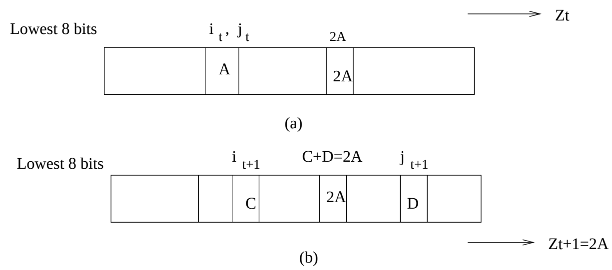

Fig. 5. NGG: (a) the array S at the end of round t, (b) the array S just before output generation at round $t+1$

D Bias-inducing State of NGG

Claim. If (i) $i_t = j_t$ and (ii) $S_{t+1}[i_{t+1}] + S_{t+1}[j_{t+1}] = 2 \cdot S_t[i_t] \bmod 256$, then

$$Z_{t+1} = 2 \cdot S_t[i_t] \bmod 2^{16} \Rightarrow Z_{t+1(0)} = 0$$

assuming that $i_{t+1} \neq 2 \cdot S_t[i_t] \bmod 256$ and $j_{t+1} \neq 2 \cdot S_t[i_t] \bmod 256$.[4]

Proof. The proof can be easily followed from Fig. 5. □

E Bias-inducing State of GGHN

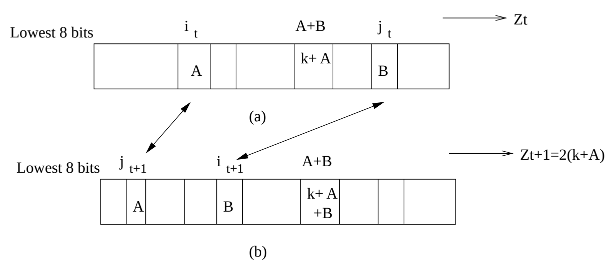

Fig. 6. GGHN: (a) the array S at the end of round t (b) the array S at the end of round $t+1$

Claim. If $S_t[i_t] = S_{t+1}[j_{t+1}]$ and $S_t[j_t] = S_{t+1}[i_{t+1}]$ then

$$Z_{t+1} = 2 \cdot (k + S_t[i_t]) \bmod 2^{32} \Rightarrow Z_{t+1(0)} = 0\,.$$

Proof. From Algorithm 5 and Fig. 6 the proof can be ascertained. □

[4] The assumption has negligible effect on the probability computed in Sect. 3.3.