On the pseudo-random generator ISAAC

Jean-Philippe Aumasson

FHNW, 5210 Windisch, Switzerland

Abstract. This paper presents some properties of he deterministic random bit generator ISAAC (FSE'96), contradicting several statements of its introducing article. In particular, it characterizes huge subsets of internal states which induce a strongly non-uniform distribution in the 8 192 first bits produced. A previous attack on ISAAC presented at Asiacrypt'06 by Paul and Preneel is demonstrated to be non relevant, since relies on an erroneous algorithm. Finally, a modification of the algorithm is proposed to fix the weaknesses discovered.

ISAAC [2] is a deterministic random bits generator presented at FSE'96 by Jenkins, who claims that it has "*no bad initial states, not even the state of all zeros*". We contradict this affirmation, presenting more than $2^{8\,167}$ weak states, in Section 2, after a short description of ISAAC and the observation of some minor weaknesses, in Section 1. Recall that, as a source of non-uniform randomness, weak states might distort simulations, and harm cryptographic applications, and so generators with many such states should not be used. Sections 3 and 4 respectively propose a modification of ISAAC's algorithm to avoid the design flaws presented, and point out an error in a previous analysis of ISAAC.

1 Preliminaries

1.1 Presentation of ISAAC

ISAAC is an array-based pseudo-random generator, derived from the generators IA and IBAA, presented in the same paper [2]. Although it is *"designed to be cryptographically secure"* [2], no security proof is given, and only statistical tests argue for its security. Nevertheless, only two publications tackled it until now: one [6] of 2001 by Pudovkina, presenting a state recovery attack running in time $2^{4\,121}$, and a recent one [5] by Paul and Preneel which presents a distinguisher running in time 2^{17}. However, as we show in Section 4, the authors of the latter attack considered an algorithm slightly distinct from the real one, that makes their attack unrelevant.

We follow the description of the algorithm provided in Figure 4 of [2]; the internal state is an array of 256 32-bit words, and at each round, the algorithm computes another array of 256 32-bit words. In the following, α denotes the initial state, and α_i its i-th element, while ω denotes the first output, and ω_i its i-th element, for $i \in \{0, \dots, 255\}$. The generation algorithm takes as parameters the initial values of the three variables a, b and c; a (32-bit) is used as an entropy accumulator, b (32-bit) contains the previous pseudo-random word, and c (8-bit) is a simple counter, incremented at each round of the algorithm. Their initial values are public, and are not part of the secret initial state.

We give the keystream procedure in Algorithm 1.1, for an arbitrary round, where the variable internal state is s, the output array is k, and the inputs a, b, and c are those computed in the previous round. The symbol $\oplus$ denotes the bitwise XOR, $+$ stands for the integer addition (modulo 2^k when needs to fit a k bit value), and $\ll$ and $\gg$ are the usual shift operators. The value $f(a,i)$ in Algorithm 1.1 is a 32-bit word, defined for all a and $i \in \{0,\dots,255\}$ as:

$$f(a,i) = \begin{cases} a \ll 13 \text{ if } i \equiv 0 \bmod 4 \\ a \gg 6 \ \ \text{ if } i \equiv 1 \bmod 4 \\ a \ll 2 \ \ \text{ if } i \equiv 2 \bmod 4 \\ a \gg 16 \text{ if } i \equiv 3 \bmod 4 \end{cases} .$$

Input: a, b, c, and the internal state s, an array of 256 32-bit words
Output: an array r of 256 32-bit words

$c \leftarrow c+1$
$b \leftarrow b+c$
for $i = 0,\dots,255$ **do**
  $x \leftarrow s_i$
  $a \leftarrow f(a,i) + s_{i+128 \bmod 256}$
  $s_i \leftarrow a + b + s_{x \gg 2 \bmod 256}$
  $r_i \leftarrow x + s_{s_i \gg 10 \bmod 256}$
  $b \leftarrow r_i$
end for
return r

Algorithm 1.1. ISAAC algorithm for an arbitrary round.

For a better understanding of the following developments, we give the redundant Algorithm 1.2, which shows more clearly how the initial state α is used to produce the first array ω.

Input: a, b, c, and the initial state α, an array of 256 32-bit words
Output: an array ω of 256 32-bit words

$b \leftarrow b+c+1$
for $i = 0,\dots,255$ **do**
  $s_i \leftarrow \alpha_i$
end for
for $i = 0,\dots,255$ **do**
  $a \leftarrow f(a,i) + s_{i+128 \bmod 256}$
  $s_i \leftarrow a + b + s_{\alpha_i \gg 2 \bmod 256}$
  $\omega_i \leftarrow \alpha_i + s_{s_i \gg 10 \bmod 256}$
  $b \leftarrow \omega_i$
end for
return ω

Algorithm 1.2. ISAAC algorithm computing the first ouput ω from the initial state α.

1.2 Observations

We report here some undesirable properties of ISAAC at the origin of the weak states presented in the next section, verified experimentally with the source code provided by ISAAC's author [1]. From now, $\equiv$ symbolizes the equivalence modulo 2^{32}.

Fact 1. *For a random initial state α, and fixed a, b, and c, the following statements are verified.*

$$\Pr\left[\exists i \in \{1,\ldots,255\}, \omega_0 \equiv \alpha_0 + \alpha_i\right] \geq \frac{255}{256}. \tag{1}$$

$$\Pr\left[\exists i \in \{1,\ldots,255\}, \omega_0 - \omega_1 \equiv \alpha_0 - \alpha_i\right] \geq \frac{254}{256^2}. \tag{2}$$

Proof. (1): let $\mu = f(a,0) + \alpha_{128} + b + c + 1 + \alpha_{(\alpha_0 \gg 2) \bmod 256}$, the value obtained at line 7 of Algorithm 1.2 at the first iteration ($i = 0$). At line 8 , when $i = 0$, we get $\omega_0 = \alpha_0 + \lambda$, where $\lambda = \mu$ if $(\mu \gg 10) \bmod 256 \neq 0$, and $\lambda = \alpha_{(\mu \gg 10) \bmod 256}$ otherwise. Since α_0 is random, $(\alpha_0 \gg 2) \bmod 256$ is a random value in $\{0,\ldots,255\}$. Since α_{128} is random, then μ is a random value in $\{0,\ldots,2^{32}-1\}$. Hence $\mu \gg 10 \bmod 256 \neq 0$ with probability 255/256, which proves the result.
(2): the result is straightforward, one simply needs to apply the previous reasoning to the two following situations.

- $\omega_0 \equiv \alpha_0 + \alpha_j$ and $\omega_1 \equiv \alpha_1 + \alpha_j$, for some $j \in \{2,\ldots,255\}$.
- $\omega_0 \equiv \alpha_0 + \alpha_1$ and $\omega_1 \equiv \alpha_1 + \alpha_j$, for some $j \in \{2, 255\}$.

□

Fact 2. *When there exists $i \in \{2,\ldots,255\}$ such that $\omega_0 = \alpha_0 + \alpha_i$, α_0 and i are correctly guessed with probability respectively 2^{-32} and $1/255$. Thus for a random α, one recovers α_0 and α_i for a certain i, with probability $2^{-32} \cdot 1/255 \cdot 255/256 = 2^{-40}$, whereas ideally this probability should be 2^{-64}.*

Fact 3. *Let $N \in \{0,\ldots,127\}$, and set $\alpha_i = X$ for all $i > N$, and $\alpha_i = Y$ for all $i \leq N$, with fixed positive integers $X < 2^9$ and $Y < 2^{10}$. If $a = b = c = 0$, then*

$$\omega_0 = \begin{cases} X + 2Y + 1 \text{ if } Y \in \{0,\ldots,M\} \\ 2X + Y + 1 \text{ if } Y \in \{M+1,\ldots,2^{10}-1\} \end{cases}, \text{with } M = \max_{0<m<2^9}\{m, (m \gg 2) < N\}.$$

The above result directly follows from Algorithm 1.2; the limitation of X to a 9-bit value comes from the fact that above this bound, $\alpha_i \gg 10 \neq 0$ (*cf.* line 8 of Algorithm 1.2). We also need $Y < 2^{10}$ so that, at line 7, we do not pick an index less than N, that is, for which $\alpha_i = Y$. For the general case, the bound M comes from the fact that, at the line 7, we shall pick the value Y as soon as $Y \gg 2$ is less than $N - 1$, and X otherwise. Finally, we need $N < 128$ in order to get $i + 128 > N \bmod 256$ for all $i \in \{0,\ldots,N-1\}$ (line 6), and so $a = X$. We obtain exactly $2^9 \cdot 2^{10} \cdot 2^7 = 2^{26}$ such states.

2 The weak states

Basically, the weak states considered have a fraction of random elements, and the remaining elements are fixed to the same value. We divide them into four non-disjoint sets: W_1, W_2, W_3 and W_4. This section defines each set, then presents the bias induced by its elements, and provides a few comments. We keep the notation α for the initial state, and ω for the first array that the algorithm outputs.

2.1 Set W_1

Definition. $\alpha \in W_1 \iff \alpha_0 = \alpha_1$.

Bias. For a random $\alpha \in W_1$,

$$\Pr[\omega_0 = \omega_1] \geq 254/256^2.$$

Indeed, for states of W_1, $\omega_0 = \omega_1$ holds as soon as a same element of index greater than 2 is picked at the first and second rounds (first has index ≥ 2 with probability 254/256, then is picked again with conditional probability 1/256).

Comments. There are $2^{32\cdot 254} \cdot 2^{32} = 2^{8\,160}$ states in W_1.

2.2 Set W_2

Definition. $\alpha \in W_2 \iff \exists N \in \{2, \ldots, 256\}, \exists X \in \{0, \ldots, 2^{32} - 1\}, \alpha_0 = X, \#\{0 < i < 256, \alpha_i = X\} = N - 1$.

Bias. For a random $\alpha \in W_2$,

$$\Pr[\omega_0 = 2X] \geq \frac{N-1}{256}.$$

Indeed, at the first round of the algorithm, a random value v of the state is picked, which is X with probability $(N-1)/256$, then $\omega_0 = \alpha_0 + v$ is returned.

Comments. A high statistical bias appears in the distribution of the first 32 bits. For example, if N is set to 6, $\Pr[\omega_0 \equiv 2X] \approx 0.02$, and there are $2^{8\,033}$ states of W_2 with $N = 5$.

There are more than $255 \cdot 2^{32\cdot 254} \cdot 2^{32} \geq 2^{8\,167.99}$ states in W_2.

2.3 Set W_3

Definition. $\alpha \in W_3 \iff \exists N \in \{2, \ldots, 256\}, \exists X \in \{0, \ldots, 2^{32}-1\}, \forall i \in \{0, \ldots, N-1\}, \alpha_i = X$.

Bias. For a random $\alpha \in W_3$,

$$\Pr[\omega_i \equiv 2X] \geq \frac{N-1-i}{256}, i = 0, \ldots, N-1.$$

Indeed, at line 8 of Algorithm 1.2, $x = X$ holds, and so $\alpha_{\alpha_i \gg 10 \bmod 256}$ is equal to X if $\alpha_i \gg 10 \bmod 256$ is greater than i and strictly less than N, which occurs with probability greater than $(N-1-i)/256$, *cf.* Fact 1.

Comments. Clearly, $W_3 \subset W_2$. Again, the value $2X$ shall appear with high probability, compared to a random bitstream, but not only in ω_0. For example, if $N = 64$ and $X = 0$: the last 192 elements of α are random, and the 64 first ones set to 0, then $\Pr[\omega_0 = \omega_1 = 0] \approx 0.06 \approx 2^{-4}$. If N is as small as 2, $\Pr[\omega_0 \equiv 2X] \approx 2^{-8}$, much higher than the 2^{-32} of an ideal generator. If N is greater than, say, 216, then $2X$ appears in average more than 90 times, thus X is recovered with high probability, and the random elements remaining can be computed by exhaustive search in 2^{48}.

There are more than $2^{32\cdot 254} \cdot 2^{32} = 2^{8\,160}$ states in W_2.

2.4 Set W_4

Definition. $\alpha \in W_4 \iff \exists X \in \{0, \ldots, 2^{32} - 1\}, \forall i \in \{0, \ldots, 255\}, \alpha_i = X$.

Bias. For a random $\alpha \in W_4$,

$$\Pr[\omega_i \equiv 2X] \geq= 1 - \frac{i+1}{256}.$$

This result comes as a particular case of W_1 states. Moreover, the expected number of i such that $\omega_i \equiv 2X$ is greater than

$$\sum_{i=0}^{255}(1 - \frac{i+1}{256}) = 127.5,$$

that is, more than half of the elements produced at the first round are $\equiv 2X$ in average, when $\alpha_i = X$ for $i = 0, \ldots, 255$.

Comments. It is straightforward to distinguish between a real random bitstream and a one produced by ISAAC initialised with a state with constant value, since the latter shall have about half of the ω_i equal to $2X$. The full state can even be trivially recovered in a few seconds with a paper and a pen.

There are exactly 2^{32} states in W_4.

2.5 Remarks

We characterized four subsets of states presenting a given bias in the distribution of ω, however they cannot be used to build a distinguisher between pseudo-random bits and true random bits, since the bias is canceled over the whole state space.

3 ISAAC+

To fix the weaknesses presented, we modify ISAAC's algorithm, and get Algorithm 1.3. We call the corresponding pseudo-random generator ISAAC+. The modifications: we add $\oplus a$ (line 7 of Algorithm 1.3) to avoid the biases observed, perform rotations (symbols $\lll$, $\ggg$) instead of shifts, so as to get more diffusion from the state bits, and replace an addition by a XOR (line 6) to reduce the linearity over $\mathbb{Z}_{2^{32}}$.

ISAAC+has the following properties.

- The properties stated in Section 1.2 do not hold: we get $\omega_0 = \alpha_0 + \alpha_i \oplus (a \ll 13 + \alpha_{128})$, for a random state, α_{128} is random in $\{0, \ldots, 2^{32} - 1\}$, thus so is $a \ll 13 + \alpha_{128}$. This contradicts the first proposition, and thereby the followings.
- The states presented in Section 2 lose their undesirable biases, for analog reasons.
- ISAAC+ runs with roughly the same algorithmic complexity.
- Like ISAAC, ISAAC+ successfully passes all the Diehard [3] and NIST [4] statistical tests (this guarantees a minimal statistical quality of the pseudo-random bitstream).

This new generator does not offer much more security guarantees than its brother, and so should not be considered as a proposal for a new pseudo-random generator.

Input: a, b, c, and the internal state s, an array of 256 32-bit words
Output: an array r of 256 32-bit words
$c \leftarrow c + 1$
$b \leftarrow b + c$
for $i = 0, \dots, 255$ **do**
  $x \leftarrow s_i$
  $a \leftarrow f'(a, i) + s_{i+128 \bmod 256}$
  $s_i \leftarrow a \oplus b + s_{x \ggg 2 \bmod 256}$
  $r_i \leftarrow x + a \oplus s_{s_i \ggg 10 \bmod 256}$
  $b \leftarrow r_i$
end for
return r

Algorithm 1.3. ISAAC+'s algorithm for an arbitrary round.

4 Comment on a previous attack

At Asiacrypt'06, Paul and Preneel presented [5] distinguishers for several stream ciphers and pseudo-random generators with RC4-like construction, including ISAAC. However their analysis is based on a incorrect version of the algorithm, probably due to the hardly understandable code given in [2]: in their paper, at line 4 of Algorithm 3, the internal state updated is not the current one, but the next; they wrote "$4 : m[i+1] = \dots$" instead of "$4 : m[i] = \dots$". In ISAAC's code, the statement `*(m++) = (...)` indeed affects the current value pointed by m at the expression given, *then* increments the pointer.

Based on this incorrect algorithm, the authors observe that the output at iteration i comes equal to $2s_i$ with probability $\frac{1}{2}(1 + 2^{-8})$. From the bias over the parity they construct a distinguisher running in time $\approx 2^{17}$. However this does not apply to the real algorithm of ISAAC, where the value s_i (denoted $m[i]$ in [5]) is updated *before* picking the output (*cf.* line 7 of Algorithm 1.2), and so the previous value of s_i is not picked with the probability they considered.

References

1. Robert J. Jenkins. `http://www.burtleburtle.net/bob/rand/isaacafa.html`.
2. Robert J. Jenkins. ISAAC. In D. Gollmann, editor, *FSE'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 41–49. Springer, 1996.
3. Georges Marsaglia. *The Diehard Battery of Tests of Randomness*, 1995. Available at `http://stat.fsu.edu/pub/diehard/`.
4. National Institue of Standards and Technology. *Statistical Test Suite 1.8*, 2005. Available at `http://http://csrc.nist.gov/rng/`.
5. Souradyuti Paul and Bart Preneel. On the (in)security of stream ciphers based on arrays and modular addition. In Xuejia Lai, editor, *ASIACRYPT'06*, Lecture Notes in Computer Science, page ? Springer, 2006.
6. Marina Pudovkina. A known plaintext attack on the ISAAC keystream generator. IACR ePrint Archive, Report 2001/049, 2001. Available at `http://eprint.iacr.org/2001/049`.