Sapparot-2
Fast Pseudo-Random Number Generator.

Ilya O. Levin, eli@literatecode.com

Abstract. The purpose of this paper is to introduce an enhanced version of Sapparot. It is a high-speed pseudo-random number generator efficient in both software and hardware yet very simple.

1 An Overview of Sapparot-2

Sapparot-2 is an enhanced version of Sapparot and is a very simple and compact pseudo-random number generator with high performance and good statistical characteristics. It produces t-bit random value per round, where $t \in \{32,64\}$.

The generator consist of three t-bit rotors and output is a result of rotors exclusive-OR (XOR) confusion. The seed of the generator is an initial state of the rotors and is a $3t$-bit value. The figure below illustrates a single round of Sapparot-2.

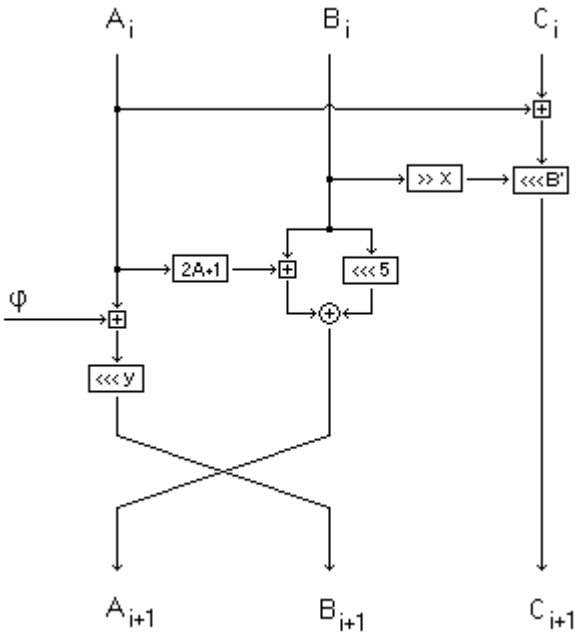

Fig. 1. A single round of Sapparot-2

A, B and C denotes rotors and and φ ("phi") is a golden ratio constant. Symbol ≫ denotes bit shift; ⋘ – cyclic rotation; ⊞ – addition modulo 2^t and ⊕ – exclusive-OR (XOR). The rotors are synchronous and updates as

$$C=(C+A) \lll (B \gg x)$$
$$B= (B + 2A +1) \oplus (B \lll 5)$$
$$A= (A+\varphi) \lll y$$

At the end of round rotors A and B swaps around. For a 32-bit version of Sapparot-2 (t=32) the parameters are: $\varphi = 9e3779b9_{hex}$, x=27 and y=7; for a 64-bit one *(t=64):* $\varphi = 9e3779b97f4a7c55_{hex}$, x=58 and y=13.

The value x was chosen to get the amount of most significant bits from B sufficient to represent any number from 0 to t-1. The impact of various choices of rotation and shift counts was examined during the design process and this combination demonstrated an optimal balance across the different statistical tests. The DIEHARD battery of tests of randomness [1] and the "tough" tests [2] were used for analysis.

2 Security

There are no weak seed values known for the generator at this moment. This mean that any seed values will produce adequate pseudo-random output stream. Even the seed values with all bits as zero would do just fine.

The internal state should be recovered to compromise the generator. The most obvious way to attack it is an exhaustive search over 2^{3t}, which is impractical for $t \geq 32$. Collision attack is impractical too because of ambiguity of $C=x \oplus y \oplus z$ when all x, y and z are unknown.

There was a naïve attack on original Sapparot with workload 2^t: guess A, use output D to deduce B, calculate next A and B, check the guess with the next sequential output. It is not applicable to Sapparot-2, the best attack of such kind will require no less than $O(2^{2t})$.

As it was mentioned in, for example, [3] any PRNG with an m-bit state can be attacked using its precomputed outputs. Sapparot-2 can be attacked the same way. Precompute the table of outputs for random $2^{\frac{3t}{2}}$ states and with given actual outputs try to locate them in this table. The single output of Sapparot-2 cannot uniquely identify the state of the generator because of ambiguity mentioned above thus the table should keep a state and a sequential output for every single precomputed output and it requires $5t(2^{\frac{3t}{2}})$ bits of memory. Having two sequential actual outputs try to locate the first one in the table; clock the generator using stored state for each found entry and match result with the second output. If the amount of actual outputs approximately the same as the amount of precomputed outputs then there is a significant probability of success.

The effective security of Sapparot-2 is 48 bits for t=32 and 96 bits for t=64.

3 Implementation

Sapparot-2 is a very simple generator and its implementation is quite straightforward. Here is the C code:

```
/* #define USE64 */
#ifndef USE64
#define uint_t      unsigned long
#define R(x,y)      (((x)<<(y))|((x)>>(32-(y))))
#define PHI         0x9e3779b9
#define C_RTR       7
#define C_SH        27
```

```
#else
#define uint_t      unsigned __int64 /* check this usage with your compiler */
#define R(x,y)      (((x)<<(y))|((x)>>(64-(y))))
#define PHI         0x9E3779B97F4A7C55UL
#define C_RTR       13
#define C_SH        58
#endif

static uint_t a=0, b=0, c=0;

uint_t Sapparot2(void)
{
  register uint_t m;

  c+=a; c=R(c,b>>C_SH);
  b=(b+((a<<1)+1))^R(b,5);
  a+=PHI; a=R(a,C_RTR);
  m=a; a=b; b=m;

  return (c^b^a);
} /* Sapparot2 */
```

References

1. The Marsaglia Random Number CDROM with The Diehard Battery of Tests of Randomness, http://stat.fsu.edu/~geo/, http://www.csis.hku.hk/~diehard/cdrom
2. G. Marsaglia and W. W. Tsang. Some difficult-to-pass tests of randomness, Journal of Statistical Software,Vol 7, http://jstatsoft.org/v07/i03/tuftests.pdf
3. A. Klimov and A. Shamir. Cryptographic Applications of T-functions, Selected Areas in Cryptography -2003, http://www.wisdom.weizmann.ac.il/~ask/t1.ps.gz

A. Acknowledgments

Thanks to Dave Wagner for input on original Sapparot.

B. About the Name

“Sapparot” is a transliteration of "**สับปะรด**" that means “pineapple” in Thai. It does not have any significant meaning to the subject but it may feet somewhere in between Rambutan and Yarrow.