Tantra : A fast PRNG algorithm and its implementation

Mahadevan Gomathisankaran and Ruby Lee
Electrical Engineering, Princeton University, Princeton, NJ, USA
{mgomathi,rblee}@princeton.edu

Abstract—Tantra is a novel Pseudorandom number generator (PRNG) design that provides a long sequence high quality pseudorandom numbers at very high rate both in software and hardware implementations.

1. Introduction

Pseudorandomness is fundamental to cryptography and is essential to achieve any cryptographic function such as encryption, authentication and identification. A pseudorandom number generator (PRNG) is an deterministic algorithm that on input of a short random seed outputs a (typically much) longer sequence that is computationally indistinguishable from a uniformly chosen random sequence.

Blum, Micali and Yao [1], [2] introduced the concept of pseudorandom number generators. Håstad et. al [3] show that existence of one-way functions is a necessary and sufficient condition for the existence of pseudorandom number generators. The central property of pseudorandom generators is effective similarity (introduced by Goldwasser, Micali and Yao [4], [5]). The underlying thesis is that it is not important whether or not objects are equal, but whether or not a difference between them can be observed by a feasible computation is of importance. This is defined by the notion computational indistinguishability. Thus, the fundamental properties of any PRNG are expansion, irreversibility and computational indistinguishability.

The quality of a PRNG should be based on theoretical fundamentals but should also be tested empirically. Various statistical tests [6], [7], [8], [9], [10], [11] are available in the literature that test the given sequence for some level of computational indistinguishability. TestU01 [11] provides a general and extensive library for statistical testing of PRNGs. It implements a larger variety of tests than any other test-suites like DIEHARD [8], NIST [10], etc. In fact it encompasses most of the other test-suites. Thus, the proposed PRNG is tested using TestU01 [11] for its statistical pseudorandomness.

The primary design goals of our proposed algorithm, Tantra, are:
- sound theoretical basis
- passing the empirical tests of TestU01 [11]
- efficient design for both hardware and software implementations

The rest of our paper is organized as follows: Section 2 discusses related work. Section 3 describes the design and analysis of our algorithm. Section 4 provides implementation details and performance measurements of our algorithm. Finally Section 5 concludes the paper.

2. Related Work

PRNGs have been well researched for a long time in various application domains. One of the direct applications of PRNGs is in synchronous stream ciphers. This class of stream ciphers encrypt the plaintext by XORing it with a pseudorandom key stream. The security of the stream cipher is dependent on the quality of the pseudorandom key stream. Therefore, the design goal of a synchronous stream cipher is to efficiently generate pseudo-random bits which are practically indistinguishable from truly random bits. In other words a PRNG that meets the requirements of expansion, irreversibility and computational indistinguishability is a good basis for synchronous stream cipher.

Recently, the European Network of Excellence for Cryptology (ECRYPT) conducted a Stream cipher project (eSTREAM) [12] from 2004 to 2008. The goal of this project is to come up with efficient and secure stream cipher algorithms. Out of several submitted algorithms, eSTREAM has chosen a few algorithms suitable for software implementation and few for hardware implementation. eSTREAM emphasizes that most of the stream
ciphers in the finalists are very new and while it believes them to be promising, it is left to others to decide when analysis is sufficiently mature for an algorithm to be considered in standards or used in a deployment. The finalists in software are: HC-128, Rabbit, SOSEMANUK, Salsa20/12 and the finalists in hardware are: Grain v1, Mickey v2 and Trivium.

eSTREAM ciphers are subjected to statistical tests [13] but did not include TestU01 [11]. We tested one of the finalists, HC-128, with the TestU01 statistical tests and it showed some weakness in the statistical properties of the sequence (detailed explanation is given in Section 3.4). While a failure in statistical tests does not mean that the stream cipher is not secure, it means that the key stream is not close to random hence could reveal some statistical properties of the plain text. Thus, one of the motivations of our design is to fare well against the eSTREAM finalists both in terms of performance and quality.

3. Design

3.1 Notations

\[
\begin{align*}
\mathbb{U}_{m \times N} & \rightarrow \text{Universal set of } m \times N \text{ functions.} \\
\mathbb{I}_m & \rightarrow \text{Set of all } m \text{ bit vectors.} \\
\in_R & \rightarrow \text{Choosing an element uniformly at random from its set.} \\
i\|x\|_n & \rightarrow i^{\text{th}} n\text{-bit nibble of the bit vector } x \\
+ & \rightarrow \text{modulo } n \text{ addition.} \\
- & \rightarrow \text{modulo } n \text{ subtraction.}
\end{align*}
\]

3.2 Rationale

Any function in the universe \( \mathbb{U}_{m \times N} \) can be represented as a Look-Up-Table (LUT) of \( 2^m \) rows of \( N \)-bit values. If a function \( f \) is chosen at random from the universe \( \mathbb{U}_{m \times N} \) then, a random sequence can be generated by outputting \( f(i) \) for all the possible inputs \( i \). Such a generator will not achieve expansion as the amount of bits it requires to choose the function \( f \) is the same as the amount of bits it can generate.

Achieving Expansion:

Consider a PRNG \( \Omega \), shown in Figure 1, that reduces the input length without reducing the output length.

Construction 1 (PRNG: \( \Omega \)): Let \( f_0 \) and \( f_1 \) be a \( \frac{m}{2} \times N \) functions chosen at random from the universal set \( \mathbb{U}_{\frac{m}{2} \times N} \). The PRNG generates a sequence of \( 2^m \) \( N \)-bit numbers,

\[
y_i = f_0( 0 \| i \| \frac{m}{2} ) + N f_1( 1 \| i \| \frac{m}{2} ), \forall 0 \leq i < 2^m.
\]

\( \Omega \) requires \( 2 \cdot N \cdot 2^\frac{m}{2} \) random bits (each of \( f_0 \) and \( f_1 \) require \( 2^\frac{m}{2} \cdot N \) bits) as input and produces \( N \cdot 2^m \) as output, an expansion factor of \( 2^\frac{m}{2} - 1 \). Even though \( \Omega \) achieves expansion, the sequence it generates is not computationally indistinguishable.

The deterministic way in which the tables \( f_0 \) and \( f_1 \) are accessed becomes the weakness of \( \Omega \) i.e., the position of \( m \)-bit number in the random sequence determines the indices of the tables \( f_0 \) and \( f_1 \).

Achieving Indistinguishability:

PRNG \( \mathcal{R} \) (shown in Figure 2) removes the correlation between the indices of tables \( f_0 \) and \( f_1 \) and output sequence.

Construction 2 (PRNG: \( \mathcal{R} \)): Let \( f_k \in_R \mathbb{U}_{\frac{m}{2} \times N} \forall 0 \leq k \leq 1 \) and \( iv \in_R \mathbb{I}_m \). The PRNG generates a sequence of \( 2^m \) \( N \)-bit numbers,

\[
y_i = f_0( 0 \| x_i \| \frac{m}{2} ) + N f_1( 1 \| x_i \| \frac{m}{2} ) , \forall 0 \leq i < L,
\]

where \( x_i = iv + m i + m y_{i-1}, y_{-1} = 0 \) and \( L \leq 2^m \).

\( \mathcal{R} \) takes \( 2 \cdot N \cdot 2^\frac{m}{2} + m \) random bits as input and produces \( N \cdot L \) bits as output. \( \mathcal{R} \) uses a random initial vector \( iv \) and feeds the output back to remove the correlation between the output sequence index and input index. Thus \( \mathcal{R} \) achieves both expansion and computational indistinguishability. The advantage the adversary enjoys in distinguishing the output of \( \mathcal{R} \) from truly random sequence determines the degree of computational indistinguishability.

Let \( \mathcal{R} \) be the truly random sequence of \( L \cdot N \) bits. Then, \( P[y_i = y_j | \mathcal{R}] = \frac{1}{2^m} \). In the case of \( \mathcal{R} \), the collision probability depends on two factors namely, the collision of inputs \( x_i \) and \( x_j \) and the collision of outputs themselves.

\[
P[y_i = y_j | \mathcal{R}] = P[y_i = y_j | x_i = x_j] P[x_i = x_j] + P[y_i = y_j | x_i \neq x_j] P[x_i \neq x_j]
< \frac{1}{2^m} + \frac{1}{2N}
\]
Thus in a sequence of length $L \cdot N$ bits, the advantage of the adversary is bounded by (birthday paradox),

$$\text{Advantage}(R) < \frac{L^2}{2^{m+1}}$$  \hspace{1cm} (1)

The expansion factor is given by,

$$\text{Exp}(R) = \frac{L}{k \cdot 2^m + m}$$  \hspace{1cm} (2)

Notice that both the expansion factor and adversary advantage are directly proportional to $L$.

### Further Expansion:

To increase the expansion factor without increasing the adversary’s advantage we can regenerate one of the component functions after every $L$ generations, i.e., $f_0(x) + f_1(y)$ is independent of $f_0'(x) + f_1(y)$ provided $f_0$ and $f_0'$ are independent. To regenerate the component functions we need an efficient method to generate pseudo-random bits and an efficient method to feed that back into the tables.

The bits that need to be fed back should be independent of the bits the PRNG generates for output. The carry bit that gets generated in the addition operation of $R$ satisfies this requirement. An efficient feed back method is to use the existing bits in the component tables and use some nonlinear function to generate new bits. The simplest of nonlinear operations is shifting or rotation. Thus the carry bit can be fed back to each row in the LUT with each row shifting one bit out. Both the expansion factor and adversary advantage is dependent on the bias (quality) of the bit that is fed back into the table.

### 3.3 Specification

Algorithm 1 provides the specification in pseudocode for Tantra($N = 64, m = 32, k = 4$) PRNG. It has 4 LUTs ($T_0, T_1, T_2$ and $T_3$) of 256 rows of 64 bits, thus requiring $4 \times 256 \times 64 = 8\text{KB}$ of LUT space. These four tables are organized in such a way that they can be accessed as a unified single table $T$. It has two 64-bit registers ($IV$ and $CV$) and a one-bit register ($CS$). It has one 8-bit counter ($C_1$) and one 10-bit counter ($C_2$). For every iteration it generates a 64-bit pseudorandom number. All the additions and subtractions in the algorithm are modulo $2^N$.

This algorithm uses the principles laid down in Section 3.2. The choice of registers and elements in tables to be of size 64 bits is due to the data width of existing 64-bit processor architectures. This simplifies the addition and subtraction operations mapping them directly to processor instructions.

Tantra($N = 64, m = 32, k = 4$) requires 65,664 bits (8KB for the tables and 128-bits for $CV$ and $IV$) of seed. The quality of seed bits affects the quality of the PRNG output. Hence we use AES-128 [14] in counter mode to generate the seed bits. AES-128

requires 256-bits (128 bits for the key and 128 bits for the initial value of the counter) of seed. Any good quality PRNG can be used to seed Tantra, AES is just one of the many choices.

### 3.4 Analysis

**Tantra** \((N = 64, m = 32, k = 4)\) uses two techniques in the feedback mechanism to increase the expansion factor without increasing the adversary advantage. The first technique is to use XOR operation to reduce the bias of the carry bit used to feedback into the table. Let \(b_0\) and \(b_1\) are two bits with bias \(\epsilon\), i.e. \(P[b_0 = 0] = P[b_1 = 0] = \frac{1}{2} + \epsilon\). Then, \(P[b_0 \text{xor } b_1 = 0] = \frac{1}{2} + 2\epsilon^2\). Since \(\epsilon < \frac{1}{2}\), \(2\epsilon^2 < \epsilon\). In general after \(k\) XOR operations the bias will be \(2^{k-1}\epsilon^k\). The second technique is to use shift operation (which is nonlinear) to generate independent numbers. Let \(X\) and \(Y\) be two random number of same bit length and let \(Z\) be \(X\) left shifted by 1 bit and a new random bit added to the least significant bit (LSB), then the number \(X + Y\) is independent of the number \(Z + Y\).

**Tantra** \((N = 64, m = 32, k = 4)\) updates all the rows in a table in \(2^{16}\) iterations. This feedback rate \((FBR = C1 \times 2^7)\) is chosen so that any polynomial adversary in \(m\) cannot distinguish the sequence from a random one with significant advantage. The resources (memory or computation steps) available to a polynomial adversary in \(m\) is bounded by polynomial \(P(m)\). The adversary requires \(O(L^2)\) computation steps to mount a birthday attack on a sequence of \(L\) numbers. The feedback rate is chosen such that \(L < \frac{2^{32}}{FBR}\).

The constant 0x77777777 (a relative prime to \(2^{64}\)) is chosen so that in every iteration, all the table accesses gets affected. The operations \(CV - IV\) and \(CV + IV\) is to mask both the input and output of the tables which is required to make the tables irreversible. The PRNG cycles when all the tables resemble the initial state or the table is filled with entries such that they generate a constant number. Both these conditions depend on the quality (bias) of the feedback bits (assuming the initial seed is perfectly random). The bias of feedback bits are bounded by \(2^{55\epsilon - 256}\). Thus even with a starting bias of \(\frac{1}{4}\) the feedback bias is less than \(\frac{1}{128}\). With such a low bias the period is closer to \(2^{644356}\).

We used TestU01 [11] to test the statistical properties of the sequence generated by **Tantra** \((N = 64, m = 32, k = 4)\). TestU01 is chosen as it is the most comprehensive test suite available and it encompasses most of the other public domain tests [6], [8], [10]. We tested for p-values within the boundary \([10^{-3}, 1 - 10^{-3}]\). We considered any p-value lying outside this boundary as a failure. This is the standard range the test-suite suggests. We also performed the same tests for HC-128, which is one of the software stream ciphers in the eSTREAM [12] profile. Table 1 lists the battery of tests performed and the results. *Standard* parameters in the Table refers to built-in parameters of the battery of tests. In all, 376 statistical tests were performed and Tantra passed all of them.

### 4. Implementation

We implemented **Tantra** \((N = 64, m = 32, k = 4)\) PRNG in C and assembly on a 64-bit x86

---

Table 1: TestU01 Statistical Test Results

<table>
<thead>
<tr>
<th>Battery</th>
<th>Parameters</th>
<th># Statistics</th>
<th># Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tantra</td>
<td>HC-128</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>Alphabit</td>
<td>32 × 10⁹ bits</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>Rabbit</td>
<td>32 × 10⁹ bits</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>Crush</td>
<td>Standard</td>
<td>144</td>
<td>0</td>
</tr>
<tr>
<td>Big Crush</td>
<td>Standard</td>
<td>160</td>
<td>0</td>
</tr>
<tr>
<td>FIPS</td>
<td>Standard</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: Performance Results using eSTREAM framework

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Key bits</th>
<th>IV bits</th>
<th>Stream cyc/byte</th>
<th>Key Setup cycles</th>
<th>IV Setup cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tantra</td>
<td>128</td>
<td>128</td>
<td>3.30</td>
<td>375.50</td>
<td>269624.00</td>
</tr>
<tr>
<td>HC-128</td>
<td>128</td>
<td>128</td>
<td>5.25</td>
<td>63.75</td>
<td>80835.56</td>
</tr>
<tr>
<td>SOSEMANUK</td>
<td>128</td>
<td>64</td>
<td>5.28</td>
<td>1005.10</td>
<td>753.23</td>
</tr>
<tr>
<td>Salsa20/12</td>
<td>128</td>
<td>64</td>
<td>7.20</td>
<td>45.45</td>
<td>29.17</td>
</tr>
<tr>
<td>Rabbit</td>
<td>128</td>
<td>64</td>
<td>10.24</td>
<td>681.15</td>
<td>643.43</td>
</tr>
</tbody>
</table>

architecture. The implementation is pretty close to the pseudocode presented in Algorithm 1. We used the instruction ADC (add with carry) to get the carry bit from the last addition operation. We used O3 level of optimization with gcc compiler and measured the performance on a Intel Pentium D (dual core) machine with 2MB L2 cache. Tantra operates at a steady rate of 2.72 cycles/byte. We also implemented Tantra as a synchronous stream cipher conforming to the eSTREAM API. We used the eSTREAM standard framework to measure the performance. Table 2 list these results for Tantra and few other eSTREAM ciphers. The only drawback of Tantra is the IV setup time. Since the period of Tantra is virtually infinite the IV setup does not have to be done as often in other stream ciphers.

The Tantra design is well-suited for hardware implementation. All the lookups can be parallelized and four additions can be done in two steps. Thus requiring only 3 cycles per loop or a rate of 0.375 cyc/byte.

5. Conclusion

We proposed a novel Pseudorandom number generator (PRNG) design Tantra that provides a long sequence of high quality pseudo-random numbers at very high rate both in software (2.7 cycles/byte in Pentium-4) and hardware implementations. We provided the theoretical analysis of our design and empirically validated them using statistical tests. We also compared various aspects of our design against other algorithms and show how our design outperforms them.

References


