**KT\_RNG Technical Document**

Explanatory technical document for ITALY compliance

# Introduction

KT\_RNG is a software random number generator implemented for compliance to the Italian gambling regulation.

The randomization algorithm is Mersenne Twister MT19937-64 which generates 64bit numbers (https://en.wikipedia.org/wiki/Mersenne\_Twister), with a period of 2^19937 − 1. KT\_RNG uses various techniques and implements security measures in order to ensure the best randomness and unpredictability of number generation

# Compliance checklist

This document will go through all the requirements as described by the current compliance guidelines.

**1.2.4.1 b) The RNG source code presented must be accompanied by the following information:**

* **name of the file(s)/module/function:**

***Files***

|  |  |
| --- | --- |
| basetypes.h | C header file containing basic types definitions and includes |
| KT\_RNG.h | C header file containing the definition of the RNG class and the exported functions |
| KT\_RNG.cpp | C++ source file containing the implementation of the RNG class and exported functions |

* ***Modules***

The software is a single module shared library component with no dependencies except for the following operative system libraries: libc (to run the software), libpthread (to handle threads), libm (to do mathematical operations).

## - Brief description of the functions

EXPORTED FUNCTIONS

|  |  |
| --- | --- |
| void startRNG(unsigned long long seed) | If not already initialized then will initialize the RNG with a |
|  | given seed. The seed parameter won’t be used as-is but entropy will be added with other seeds |
| unsigned int getRandom(unsigned int limit) | Returns a random number from 0 to ‘limit’ -1 |

RANDOMIZATION FUNCTIONS

|  |  |
| --- | --- |
| void init\_genrand64(unsigned long s) | Initialize the MT algorithm |
| Void init\_by\_array64(unsigned long init\_key[], int key\_length) | Initialize the MT algorithm from an array of seeds |
| unsigned long genrand64\_int64(void) | Return a random number from 0 to 0xFFFFFFFF] |

KT\_RNG FUNCTIONS

|  |  |
| --- | --- |
| void KT\_RNG\_discardNumbers(unsigned int howMany) | Discards a given amount of random numbers  advancing the generator state |
| void KT\_RNG\_init(unsigned int seed) | Generate entropized seeds and calls the MT initialization funcion |
| void KT\_RNG\_controlCycle() | Background worker that controls the re-seed period and in background discards one number from MT algorithm every CYCLE\_MILLISECONDS milliseconds |
| unsigned int KT\_RNG\_getRandom(unsigned int limit) | Returns a random number in the range [0-(limit-1)] randomly discarding from 1 to 3 numbers each time |

## - Edit history of any modifications;

Revisions

|  |  |
| --- | --- |
| v001 | First version |

**- Explanatory technical documentation**

Represented by this document (RNG\_Document.pdf or RNG\_Document.odt)

## 1.2.4.2 Documents

**a) A list of all games connected to the RNG**

Refer to KT\_RNG\_games.txt document (subject to constant revisions)

### b) n/a c) For software-based RNGs

**I) Type of mathematical algorithm used;**

Mersenne Twister MT19937 is used.

#### ii) Full details, in technical terms, of random number generation process and mathematical algorithm theory;

Mersenne Twister is a strong pseudo-random number generator. In non-rigorous terms, a strong PRNG has a long period (how many values it generates before repeating itself) and a statistically uniform distribution of values (bits 0 and 1 are equally likely to appear regardless of previous values). A version of the Mersenne Twister available in many programming languages, MT19937, has an impressive period of 2^19937-1. Sequences with too short a period can be observed, recorded, and reused by an attacker. Sequences with long periods force the adversary to select alternate attack methods.

Mersenne Twister is well known and studied, refer to public academic researches and papers for the mathematical theory explanation.

The public exported function that returns random numbers is

unsigned int getRandom(unsigned int limit)

getRandom function then calls unsigned int KT\_RNG\_getRandom(unsigned int limit) which will firstly discards 1 plus from zero to 3 random numbers using *memory entropy* as described in the point v) of this article, the value extracted by unsigned long genrand64\_int64(void) is returned giving a number from MT algorithm.

1. **Details of the mathematical algorithm period;**

The algorithm period is [(2^19937)−1]

1. **Details of the mathematical algorithm range;**

The algorithm range is from 0 to (2^32)-1

#### v) Details of the methods for seeding;

KT\_RNG library is initialized by calling the function void startRNG(unsigned int seed) which calls void KT\_RNG\_init(unsigned int seed) KT\_RNG uses 4-way seeding tecnique in order to guarantee an unpredictable and good seeding:

1. One seed is provided by the software that uses KT\_RNG library, this seed is arbitrary and provides aseparation layer from the library’s developer and the user.
2. One seed is got from the address of a dynamic virtual memory allocation

unsigned int\* giveMeRandomAddress = (unsigned int\*)malloc(sizeof(unsigned int)), the newly allocated memory address is casted into a 32bit number. Given a non-compromised operative system, it is impossible for both the library user and developer to know which memory address will be given at that time due to the unpredictable nature of the OS memory handler in conjunction to the state of the machine (total free memory, other processes, etc..). Even in the case the OS gives the same address, (fPIC) this still remains an unpredictable complementary element.

1. One seed if obtained from the machine steady clock, which returns a machine monotonic time that is NOTrelated to the wall clock system time, the epoch is related to unknown factors such as the last machine reboot or other events depending on the OS implementation. Given a non-compromised system it’s practically impossible to foresee the value of such monotonic timer because its fine milliseconds resolution.
2. Using std::random\_device we also the machine's random device own implementation, which is not required to be nondeterministic but still adds unknown random elements

The seeds are then combined together into an array of 8 elements (5 of which from std::random\_device) and given to the Mersenne Twister initialization function that provides to build the matrix accordingly

#### vi) Details of the methods for background cycling / activity

After the library is initialized a thread is started void KT\_RNG\_controlCycle()

Such function ticks every CYCLE\_MILLISECONDS milliseconds and is in charge of discarding 1 number at every ticks, furthermore, the function will also re-initialize and re-seed the RNG after a given time (CYCLE\_RESET)

The void controlCycle function will check if the number of calls have exceeded the period hence reinitialize the RNG.

Even if there is no active games, the RNG will always keep cycling.

**vii) Details of all RNG / game implementation, including methods of scaling and mapping.**

Unbiased scaling is performed by rejection sampling.

Refer to the game’s documentation for the game’s implementation details.

# 2.5 RNG REQUIREMENTS

## 2.5.1 General Introduction

### 1. a,b,c,d

Mersenne Twister algorithm is acknowledged by the scientific community to be statistically independent.

KT\_RNG implements a series of mechanisms that prevents predictability from both the developers and system administrators.

**2, 3, 4**

Refer to the game’s documentation

## 2.5.2 Scaling

The down-scaling is solely performed by rejection sampling.

## 2.5.3 Mapping

KT\_RNG does not map any game’s outcome.

Refer to the game’s documentation for the transformation of RNG numbers into outcomes.

**2.5.4 Selection of RNG**

A purely software-based RNG is selected

**2.5.5 Hardware RNG**

N/A

## 2.5.6 Software RNG

### 2.5.6.1 Period

Mersenne Twister algorithm period is 2^19937 − 1, refer to the game’s documentation for the coverage of all possible outcomes.

**2.5.6.2 Range**

The provided random numbers are in 32bit format, which is large enough for any game on the market.

### 2.5.6.1 Seeding/re-seeding

1. The 4-way seeding method implemented by KT\_RNG ensure that the seed value is secure.
2. Re-seeding is done roughly every 45 days of running.
3. The 4-way seeding method implemented by KT\_RNG ensure that the final seed value is always unpredictable.

## 2.5.7 Background cycling / activity

### 2.5.7.1

The exported function that returns a random number also always discards 1 + an unpredictable number of random numbers from the algorithm, making impossible to foresee any pattern. On top of that the controlCycle thread keeps consuming numbers every CYCLE\_MILLISECONDS ms.

**2.5.7.2**

A background activity is implemented.

### 2.5.7.3

The controlCycle ticks every CYCLE\_MILLISECONDS milliseconds, to be sure to keep the state of the RNG moving even when no game is using the RNG.