Abstract

This document describes the TinyMT32 Pseudorandom Number Generator (PRNG), which produces 32-bit pseudorandom unsigned integers and aims at having a simple-to-use and deterministic solution. This PRNG is a small-sized variant of the Mersenne Twister (MT) PRNG. The main advantage of TinyMT32 over MT is the use of a small internal state, compatible with most target platforms that include embedded devices, while keeping reasonably good randomness that represents a significant improvement compared to the Park-Miller Linear Congruential PRNG. However, neither the TinyMT nor MT PRNG is meant to be used for cryptographic applications.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8682.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions
with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
1. Introduction

This document specifies the TinyMT32 PRNG as a specialization of the reference implementation version 1.1 (2015/04/24) by Mutsuo Saito and Makoto Matsumoto from Hiroshima University, which can be found at [TinyMT-web] (the TinyMT website) and [TinyMT-dev] (the GitHub site). This specialization aims at having a simple-to-use and deterministic PRNG, as explained below. However, the TinyMT32 PRNG is not meant to be used for cryptographic applications.

TinyMT is a new, small-sized variant of the Mersenne Twister (MT) PRNG introduced in 2011 [MT98]. This document focuses on the TinyMT32 variant (rather than TinyMT64) of the TinyMT PRNG, which outputs 32-bit unsigned integers.

The purpose of TinyMT is not to replace the Mersenne Twister: TinyMT has a far shorter period \(2^{127} - 1\) than MT. The merit of TinyMT is in the small size of the 127-bit internal state, far smaller than the 19937 bits of MT. The outputs of TinyMT satisfy several statistical tests for non-cryptographic randomness, including BigCrush in TestU01 [TestU01] and AdaptiveCrush [AdaptiveCrush], leaving it well placed for non-cryptographic usage, especially given the small size of its internal state (see [TinyMT-web]). From this point of view, TinyMT32 represents a
major improvement with respect to the Park-Miller Linear Congruential PRNG (e.g., as specified in [RFC5170]), which suffers from several known limitations (see, for instance, [PTVF92], Section 7.1, p. 279 and [RFC8681], Appendix B).

The TinyMT32 PRNG initialization depends, among other things, on a parameter set, namely (mat1, mat2, tmat). In order to facilitate the use of this PRNG and to make the sequence of pseudorandom numbers depend only on the seed value, this specification requires the use of a specific parameter set (see Section 2.1). This is a major difference with respect to the implementation version 1.1 (2015/04/24), which leaves this parameter set unspecified.

Finally, the determinism of this PRNG for a given seed has been carefully checked (see Section 2.3). This means that the same sequence of pseudorandom numbers should be generated, no matter the target execution platform and compiler, for a given initial seed value. This determinism can be a key requirement, as is the case with [RFC8681], which normatively depends on this specification.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] when, and only when, they appear in all capitals, as shown here.

2. TinyMT32 PRNG Specification

2.1. TinyMT32 Source Code

The TinyMT32 PRNG must be initialized with a parameter set that needs to be well chosen. In this specification, for the sake of simplicity, the following parameter set MUST be used:

- mat1 = 0x8f7011ee = 2406486510
- mat2 = 0xfc78ff1f = 4235788063
- tmat = 0x3793fdff = 932445695

This parameter set is the first entry of the precalculated parameter sets in tinymt32dc/tinymt32dc.0.1048576.txt by Kenji Rikitake, available at [TinyMT-params]. This is also the parameter set used in [KR12].

The TinyMT32 PRNG reference implementation is reproduced in Figure 1. This is a C language implementation written for C99 [C99]. This reference implementation differs from the original source code as follows:

- The original authors, who are coauthors of this document, have granted IETF the rights to publish this version with a license and copyright that are in accordance with BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info).
- The source code initially spread over the tinymt32.h and tinymt32.c files has been merged.
• The unused parts of the original source code have been removed. This is the case of the tinymt32_init_by_array() alternative initialization function. This is also the case of the period_certification() function after having checked it is not required with the chosen parameter set.
• The unused constants TINYMT32_MEXP and TINYMT32_MUL have been removed.
• The appropriate parameter set has been added to the initialization function.
• The function order has been changed.
• Certain internal variables have been renamed for compactness purposes.
• The const qualifier has been added to the constant definitions.
• The code that was dependent on the representation of negative integers by 2's complements has been replaced by a more portable version.
/* Tiny Mersenne Twister: only 127-bit internal state. 
 * Derived from the reference implementation version 1.1 (2015/04/24) 
 * by Mutsuo Saito (Hiroshima University) and Makoto Matsumoto 
 * (Hiroshima University). */
#include <stdint.h>

/* tinymt32 internal state vector and parameters */
typedef struct {
  uint32_t status[4];
  uint32_t mat1;
  uint32_t mat2;
  uint32_t tmat;
} tinymt32_t;

static void tinymt32_next_state (tinymt32_t* s);
static uint32_t tinymt32_temper (tinymt32_t* s);

/* Parameter set to use for this IETF specification. Don't change. 
 * This parameter set is the first entry of the precalculated 
 * parameter sets in tinymt32dc/tinymt32dc.0.1048576.txt by 
 * Kenji Rikitake, available at: 
 *   https://github.com/jj1bdx/tinymt3dc-longbatch/. 
 * It is also the parameter set used in: 
 * Rikitake, K., "TinyMT pseudo random number generator for 
 * Erlang", Proceedings of the 11th ACM SIGPLAN Erlang Workshop, 
 * September 2012. */
const uint32_t  TINYMT32_MAT1_PARAM = UINT32_C(0x8f7011ee);
const uint32_t  TINYMT32_MAT2_PARAM = UINT32_C(0xfc78ff1f);
const uint32_t  TINYMT32_TMAT_PARAM = UINT32_C(0x3793fdff);

/* This function initializes the internal state array with a 
 * 32-bit unsigned integer seed. 
 * @param s pointer to tinymt internal state. 
 * @param seed a 32-bit unsigned integer used as a seed. */
void tinymt32_init (tinymt32_t* s, uint32_t seed) {
  const uint32_t    MIN_LOOP = 8;
  const uint32_t    PRE_LOOP = 8;
  s->status[0] = seed;
  s->status[1] = s->mat1 = TINYMT32_MAT1_PARAM;
  s->status[2] = s->mat2 = TINYMT32_MAT2_PARAM;
  for (int i = 1; i < MIN_LOOP; i++) {
    s->status[i & 3] ^= i + UINT32_C(1812433253) 
    * (s->status[(i - 1) & 3] 
    ^ (s->status[(i - 1) & 3] >> 30));
  }
}
NB: The parameter set of this specification warrants that none of the possible $2^{32}$ seeds leads to an all-zero 127-bit internal state. Therefore, the period_certification() function of the original TinyMT32 source code has been safely removed. If another parameter set is used, this function will have to be reintroduced here.

```c
for (int i = 0; i < PRE_LOOP; i++) {
    tinymt32_next_state(s);
}
```

This function outputs a 32-bit unsigned integer from the internal state.

```c
uint32_t tinymt32_generate_uint32 (tinymt32_t* s)
{
    tinymt32_next_state(s);
    return tinymt32_temper(s);
}
```

Internal tinymt32 constants and functions. Users should not call these functions directly.

```c
const uint32_t  TINYMT32_SH0 = 1;
const uint32_t  TINYMT32_SH1 = 10;
const uint32_t  TINYMT32_SH8 = 8;
const uint32_t  TINYMT32_MASK = UINT32_C(0x7fffffff);
```

This function changes the internal state of tinymt32.

```c
static void tinymt32_next_state (tinymt32_t* s)
{
    uint32_t x;
    uint32_t y;

    y = s->status[3];
    x = (s->status[0] & TINYMT32_MASK)
        ^ s->status[1]
        ^ s->status[2];
    x ^= (x << TINYMT32_SH0);
    y ^= (y >> TINYMT32_SH0) ^ x;
    s->status[0] = s->status[1];
    s->status[1] = s->status[2];
    s->status[2] = x ^ (y << TINYMT32_SH1);
    s->status[3] = y;
    /* The if (y & 1) {...} block below replaces:
     * s->status[1] ^= -((int32_t)(y & 1)) & s->mat1;
     * s->status[2] ^= -((int32_t)(y & 1)) & s->mat2;
     * The adopted code is equivalent to the original code
* but does not depend on the representation of negative integers by 2's complements. It is therefore more portable but includes an if branch, which may slow down the generation speed.

```c
if (y & 1) {
    s->status[1] ^= s->mat1;
    s->status[2] ^= s->mat2;
}
```

/*
 * This function outputs a 32-bit unsigned integer from the internal state.
 * @param s pointer to tinymt internal state.
 * @return 32-bit unsigned pseudorandom number.
 */
static uint32_t tinymt32_temper (tinymt32_t* s)
{
    uint32_t t0, t1;
    t0 = s->status[3];
    t1 = s->status[0] + (s->status[2] >> TINYMT32_SH8);
    t0 ^= t1;
    /*
     * The if (t1 & 1) {...} block below replaces:
     *     t0 ^= -((int32_t)(t1 & 1)) & s->tmat;
     * The adopted code is equivalent to the original code but does not depend on the representation of negative integers by 2's complements. It is therefore more portable but includes an if branch, which may slow down the generation speed.
     */
    if (t1 & 1) {
        t0 ^= s->tmat;
    }
    return t0;
}

<CODE ENDS>

Figure 1: TinyMT32 Reference Implementation

### 2.2. TinyMT32 Usage

This PRNG **MUST** first be initialized with the following function:

```c
void tinymt32_init (tinymt32_t* s, uint32_t seed);
```

It takes as input a 32-bit unsigned integer used as a seed (note that value 0 is permitted by TinyMT32). This function also takes as input a pointer to an instance of a tinymt32_t structure that needs to be allocated by the caller but is left uninitialized. This structure will then be updated by the various TinyMT32 functions in order to keep the internal state of the PRNG. The use of this structure admits several instances of this PRNG to be used in parallel, each of them having its own instance of the structure.
Then, each time a new 32-bit pseudorandom unsigned integer between 0 and $2^{32} - 1$ inclusive is needed, the following function is used:

```c
uint32_t tinymt32_generate_uint32 (tinymt32_t * s);
```

Of course, the tinymt32_t structure must be left unchanged by the caller between successive calls to this function.

### 2.3. Specific Implementation Validation and Deterministic Behavior

For a given seed, PRNG determinism can be a requirement (e.g., with [RFC8681]). Consequently, any implementation of the TinyMT32 PRNG in line with this specification **MUST** have the same output as that provided by the reference implementation of Figure 1. In order to increase the compliance confidence, this document proposes the following criteria. Using a seed value of 1, the first 50 values returned by `tinymt32_generate_uint32(s)` as 32-bit unsigned integers are equal to the values provided in Figure 2, which are to be read line by line. Note that these values come from the tinymt/check32.out.txt file provided by the PRNG authors to validate implementations of TinyMT32 as part of the MersenneTwister-Lab/TinyMT GitHub repository.

```plaintext
2545341989 981918433 3715302833 2387538352 3591001365
3829442162 2114406566 2196103051 2783359912 764534599
643179475 1822416315 881558334 4207623636 398273640
3240535687 2921447122 3984931427 4692394160 44209675
2188315343 2908663843 1834519336 3774670961 3019990787
4065554902 1293765502 4035716197 3412127188 552822483
161364450 353727785 140685994 149132608 254778827
4064642525 4678297538 2057335587 622384752 2041665899
2193913817 1080849512 33160901 662956935 642999063
3384709977 1723175122 3866752252 521822317 2292524454
```

Figure 2: First 50 decimal values (to be read per line) returned by `tinymt32_generate_uint32(s)` as 32-bit unsigned integers, with a seed value of 1

In particular, the deterministic behavior of the Figure 1 source code has been checked across several platforms: high-end laptops running 64-bit Mac OS X and Linux/Ubuntu; a board featuring a 32-bit ARM Cortex-A15 and running 32-bit Linux/Ubuntu; several embedded cards featuring either an ARM Cortex-M0+, a Cortex-M3, or a Cortex-M4 32-bit microcontroller, all of them running RIOT [Baccelli18]; two low-end embedded cards featuring either a 16-bit microcontroller (TI MSP430) or an 8-bit microcontroller (Arduino ATMEGA2560), both of them running RIOT.

This specification only outputs 32-bit unsigned pseudorandom numbers and does not try to map this output to a smaller integer range (e.g., between 10 and 49 inclusive). If a specific use case needs such a mapping, it will have to provide its own function. In that case, if PRNG determinism is also required, the use of a floating point (single or double precision) to perform this mapping should probably be avoided, as these calculations may lead to different rounding errors across different target platforms. Great care should also be taken to not introduce biases in the
randomness of the mapped output (which may be the case with some mapping algorithms) incompatible with the use-case requirements. The details of how to perform such a mapping are out of scope of this document.

3. Security Considerations

The authors do not believe the present specification generates specific security risks per se. However, the TinyMT and MT PRNG must not be used for cryptographic applications.

4. IANA Considerations

This document has no IANA actions.

5. References

5.1. Normative References


5.2. Informative References


[MT98]


Acknowledgments

The authors would like to thank Belkacem Teibi, with whom we explored TinyMT32 specificities when looking to an alternative to the Park-Miller Linear Congruential PRNG. The authors would also like to thank Carl Wallace; Stewart Bryant; Greg Skinner; Mike Heard; the three TSVWG chairs, Wesley Eddy (our shepherd), David Black, and Gorry Fairhurst; as well as Spencer Dawkins and Mirja Kuehlewind. Last but not least, the authors are really grateful to the IESG members, in particular Benjamin Kaduk, Eric Rescorla, Adam Roach, Roman Danyliw, Barry Leiba, Martin Vigoureux, and Eric Vyncke for their highly valuable feedback that greatly contributed to improving this specification.
Authors' Addresses

Mutsuo Saito
Hiroshima University
Japan
Email: saito@math.sci.hiroshima-u.ac.jp

Makoto Matsumoto
Hiroshima University
Japan
Email: m-mat@math.sci.hiroshima-u.ac.jp

Vincent Roca (EDITOR)
INRIA
Univ. Grenoble Alpes
France
Email: vincent.roca@inria.fr

Emmanuel Baccelli
INRIA
France
Email: emmanuel.baccelli@inria.fr