TinyMT32 Pseudo Random Number Generator (PRNG)
draft-roca-tsvwg-tinymt32-01

Abstract

   This document describes the TinyMT32 Pseudo Random Number Generator
   (PRNG) that produces 32-bit pseudo-random unsigned integers and aims
   at having a simple-to-use and deterministic solution. This PRNG is a
   small-sized variant of Mersenne Twister (MT) PRNG, also designed by
   M. Saito and M. Matsumoto. The main advantage of TinyMT32 over MT
   is the use of a small internal state, compatible with most target
   platforms including embedded devices, while keeping a reasonably good
   randomness.

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF). Note that other groups may also distribute
   working documents as Internet-Drafts. The list of current Internet-
   Drafts is at https://datatracker.ietf.org/drafts/current/.

   Internet-Drafts are draft documents valid for a maximum of six months
   and may be updated, replaced, or obsoleted by other documents at any
   time. It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   This Internet-Draft will expire on September 6, 2019.

Copyright Notice

Table of Contents

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:

o Official web site: <http://www.math.sci.hiroshima-u.ac.jp/~m-
   mat/MT/TINYMT/>
o Official github site and reference implementation:
   <https://github.com/MersenneTwister-Lab/TinyMT>

This specialisation aims at having a simple-to-use and deterministic
PRNG, as explained below.

TinyMT is a new small-sized variant of Mersenne Twister (MT)
introduced by Mutsuo Saito and Makoto Matsumoto in 2011. This
document focusses on the TinyMT32 variant (rather than TinyMT64) of
the PRNG, which outputs 32-bit unsigned integers.

The purpose of TinyMT is not to replace Mersenne Twister: TinyMT has
a far shorter period than MT. The merit of TinyMT is in its small
size of the internal state of 127 bits, far smaller than 19937 bits
of MT. According to statistical tests (BigCrush in TestU01
<http://simul.iro.umontreal.ca/testu01/tu01.html> and AdaptiveCrush

<http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ADAPTIVE/>) the
quality of the outputs of TinyMT seems pretty good, taking the small
size of the internal state into consideration. From this point of
view, TinyMT32 represents a major improvement with respect to the
Park-Miler Linear Congruential PRNG (e.g., as specified in
[RFC5170]).

The TinyMT32 PRNG initialization depends, among other things, on a
parameter set -- namely (mat1, mat2, tmat) -- that needs to be well
chosen (pre-calculated values are available in the official web
site). In order to facilitate the use of this PRNG, and unlike the
implementation version 1.1 (2015/04/24) by Mutsuo Saito and Makoto
Matsumoto, this specification requires the use of a specific
parameter set (see Section 3.1). The implementation version 1.1
(2015/04/24) also proposes two initialisation functions that differ
on the approach to seed the PRNG. A second difference is the removal
of the tinymt32_init_by_array() function to keep only the simple
initialisation through a single seed value (see Section 3.2).

Finally, the determinism of this PRNG, for a given seed, has been
carefully checked (see Section 3.3). Indeed, this determinism can be
a key requirement as it the case with [RLC-ID] that normatively
depends on this specification.

2. Definitions

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] [RFC8174] when, and only when, they appear in all
capitals, as shown here.

3. TinyMT32 PRNG Specification

3.1. TinyMT32 Source Code

The TinyMT32 PRNG requires to 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:

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

This parameter set is the first entry of the precalculated parameter
sets in file tinymt32dc.0.1048576.txt, by Kenji Rikitake, and
available at <https://github.com/jj1bdx/tinymtdc-

longbatch/blob/master/tinymt32dc/tinymt32dc.0.1048576.txt>. This is
also the parameter set used in [KR12].

The TinyMT32 PRNG reference implementation is reproduced in Figure 1,
with the following differences with respect to the original source
code:

o the original copyright and licence have been removed, in
   accordance with BCP 78 and the IETF Trust's Legal Provisions
   Relating to IETF Documents (http://trustee.ietf.org/license-info);
o the source code initially spread over the tinymt32.h and
   tinymt32.c files has been merged;
o the unused parts of the original source code have been removed.
   This is the case of the tinymt32_init_by_array() alternative
   initialisation function;
o the unused constants TINYMT32_MEXP and TINYMT32_MUL have been
   removed;
o the appropriate parameter set has been added to the initialization
   function;
o the function order has been changed;
o certain internal variables have been renamed for compactness
   purposes;
o the const qualifier has been added to the constant definitions.

```
<CODE BEGINS>
/**
 * 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 file tinymt32dc.0.1048576.txt, by Kenji
    * Rikitake, available at:
    *     https://github.com/jj1bdx/tinymtdc-longbatch/blob/master/
    *     tinymt32dc/tinymt32dc.0.1048576.txt
    * It is also the parameter set used:
    *     Rikitake, K., "TinyMT Pseudo Random Number Generator for
    *     Erlang", ACM 11th SIGPLAN Erlang Workshop (Erlang'12),
    *     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;
       s->status[3] = s->tmat = TINYMT32_TMAT_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 re-introduced here.
        */
       for (int i = 0; i < PRE_LOOP; i++) {
           tinymt32_next_state(s);
       }
   }

   /**
```

```
 * This function outputs a 32-bit unsigned integer from
 * the internal state.
 * @param s     pointer to tinymt internal state.
 * @return      32-bit unsigned integer r (0 <= r < 2^32).
 */
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.
 */
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.
 * @param s     pointer to tinymt internal state.
 */
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.
```

```
     */
    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 pseudo-random 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;
    t0 ^= -((int32_t)(t1 & 1)) & s->tmat;
    return t0;
}
<CODE ENDS>
```

             Figure 1: TinyMT32 Reference Implementation

3.2. TinyMT32 Usage

   This PRNG MUST first be initialized with the following function:

      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 authorized 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 left uninitialized. This structure
   will then updated by the various TinyMT32 functions in order to keep
   the internal state of the PRNG. The use of this structure authorizes
   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 pseudo-random unsigned integer between 0
   and 2^32 - 1 inclusive is needed, the following function is used:

      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.

3.3. Specific Implementation Validation and Deterministic Behavior

   PRNG determinism, for a given seed, can be a requirement (e.g., with
   [RLC-ID]). Consequently, any implementation of the TinyMT32 PRNG in
   line with this specification MUST comply with the following criteria.
   Using a seed value of 1, the first 50 values returned by
   tinymt32_generate_uint32(s) as 32-bit unsigned integers MUST be equal
   to values provided in Figure 2. 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.

   2545341989   981918433 3715302833 2387538352 3591001365
   3820442102 2114400566 2196103051 2783359912   764534509
    643179475 1822416315   881558334 4207026366 3690273640
   3240535687 2921447122 3984931427 4092394160   44209675
   2188315343 2908663843 1834519336 3774670961 3019990707
   4065554902 1239765502 4035716197 3412127188   552822483
    161364450   353727785   140085994   149132008 2547770827
   4064042525 4078297538 2057335507   622384752 2041665899
   2193913817 1080849512   33160901   662956935   642999063
   3384709977 1723175122 3866752252   521822317 2292524454

                 Figure 2: First 50 decimal values 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-bits Mac OSX and Linux/Ubuntu; a board featuring a 32-bits 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 a 8-bit microcontroller (Arduino ATMEGA2560), both of them running
   RIOT.

   This specification only outputs 32-bit unsigned pseudo-random 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 floating point (single
   or double precision) to perform this mapping should probably be
   avoided, these calculations leading potentially to different rounding
   errors across different target platforms. Great care should also be
   put on not introducing biases in the randomness of the mapped output
   (it 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.

4. Security Considerations

The authors do not believe the present specification generates
specific security risks per se.

5. IANA Considerations

This document does not require any IANA action.

6. Acknowledgments

The authors would like to thank Belkacem Teibi with whom we explored
TinyMT32 specificities when looking to an alternative to the Park-
Miler Linear Congruential PRNG. The authors would like to thank the
three TSVWG chairs, Wesley Eddy, our shepherd, David Black and Gorry
Fairhurst, as well as Spencer Dawkins and Mirja Kuhlewind. Last but
not least, the authors are really grateful to the IESG members, in
particular Benjamin Kaduk, Eric Rescorla, and Adam Roach for their
highly valuable feedbacks that greatly contributed to improve this
specification.

7. References

7.1. Normative References

[RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
            Requirement Levels", BCP 14, RFC 2119,
            DOI 10.17487/RFC2119, March 1997,
            <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174]   Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
            2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,
            May 2017, <https://www.rfc-editor.org/info/rfc8174>.

7.2. Informative References

[Baccelli18]
            Baccelli, E., Gundogan, C., Hahm, O., Kietzmann, P.,
            Lenders, M., Petersen, H., Schleiser, K., Schmidt, T., and
            M. Wahlisch, "RIOT: An Open Source Operating System for
            Low-End Embedded Devices in the IoT", IEEE Internet of
            Things Journal (Volume 5, Issue 6), DOI:
            10.1109/JIOT.2018.2815038, December 2018.

   [KR12]   Rikitake, K., "TinyMT Pseudo Random Number Generator for
              Erlang", ACM 11th SIGPLAN Erlang Workshop (Erlang'12),
              September 14, 2012, Copenhagen, Denmark, DOI:
              http://dx.doi.org/10.1145/2364489.2364504, September 2012.

   [RFC5170] Roca, V., Neumann, C., and D. Furodet, "Low Density Parity
              Check (LDPC) Staircase and Triangle Forward Error
              Correction (FEC) Schemes", RFC 5170, DOI 10.17487/RFC5170,
              June 2008, <https://www.rfc-editor.org/info/rfc5170>.

   [RLC-ID]   Roca, V. and B. Teibi, "Sliding Window Random Linear Code
              (RLC) Forward Erasure Correction (FEC) Scheme for
              FECFRAME", Work in Progress, Transport Area Working Group
              (TSVWG) draft-ietf-tsvwg-rlc-fec-scheme (Work in
              Progress), February 2019, <https://tools.ietf.org/html/
              draft-ietf-tsvwg-rlc-fec-scheme>.

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
   INRIA
   Univ. Grenoble Alpes
   France

   EMail: vincent.roca@inria.fr

   Emmanuel Baccelli
   INRIA
   France

   EMail: emmanuel.baccelli@inria.fr