Journal of Information Security and Applications 73 (2023) 103430

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

A tradeoff paradigm shift in cryptographically-secure pseudorandom number generation based on discrete logarithm

Takeshi Koshiba [a], Behrouz Zolfaghari [a], Khodakhast Bibak [b,*]

[a] Department of Mathematics, Faculty of Education and Integrated Arts and Sciences, Waseda University, Japan
[b] Department of Computer Science and Software Engineering, Miami University, Oxford, OH, 45056, USA

ARTICLE INFO

MSC:
65C10
94A60
11Y16

Keywords:
Cryptographically-secure pseudorandom number generator
Discrete logarithm problem
Mersenne primes
Security-performance tradeoff

ABSTRACT

Discrete logarithmic pseudorandom number generators are a prevailing class of cryptographically-secure pseudorandom number generators (CSPRNGs). In generators of this type, the security parameter affects both security and performance. This adds to the design complexity via creating a critical tradeoff between security and performance. This research is an attempt at shifting the security-performance tradeoff paradigm in this realm. To this end, we propose a modification to Gennaro's pseudorandom number generator via replacing word-wise arithmetic operations with bit-wise logical operations in trapdoor and hard-core functions. The security of our generator (like that of Gennaro's) is based on the hardness of a special variant of the discrete logarithm problem. We establish an equivalence between the specific variant of the discrete logarithm problem with the standard problem. Moreover, we demonstrate that in the modified generator, performance will be almost independent of the security parameter as logical operations can be performed in register level without the interference of the Arithmetic-Logic Unit (ALU). This relaxes the security-performance tradeoff and allows designers to maneuver more flexibly in the tradeoff space. We implement and evaluate our proposed generator and prove its security. Our CSPRNG is deemed random by all randomness tests in NIST SP 800-22 suite.

1. Introduction and basic concepts

Nowadays, numerous computing environments ranging from image processing [1] and vehicular technology [2] to cloud computing [3] depend on cryptography for their security. On the other hand, cryptosystems depend on various cryptographic primitives such as hashing [4] and key management [5] as well as random generation of different numeric [6] or non-numeric [7] objects. Particularly, random number generation plays a critical role in most existing cryptosystems. With recent advancements, random numbers can be used even for generating random non-numeric objects [8].

Random numbers can be divided into two categories, namely true-random [9] and pseudorandom [10] numbers. True-random numbers are extracted from unpredictable natural and physical phenomena such as waves, noises and irregularities in fabrication technologies [11]. Contrastingly, pseudorandom numbers are deterministically generated by Pseudorandom Number Generators (PRNGs) using mathematical computation and computer algorithms [12].

A cryptographically-secure pseudorandom sequence of bits or numbers is roughly defined as a sequence indistinguishable from a true-random one in a polynomial time using any possible statistical test. The notion of indistinguishability can be formally defined as follows.

Let X_n and Y_n be arbitrary probability ensembles over $\{0,1\}^n$. We denote by $x \leftarrow X_n$ the selection of an element x in $\{0,1\}^n$ according to the distribution X_n.

We say that X_n and Y_n are *computationally indistinguishable* if no polynomial-time Turing machines can distinguish $x \leftarrow X_n$ from $y \leftarrow Y_n$. We define this term more formally as follows.

Definition 1.1. Let X_n and Y_n be probability ensembles over $\{0,1\}^n$. Given a probabilistic polynomial-time Turing machine D consider the following quantity:

$$\delta_D(X_n, Y_n) = \left| \Pr[x \leftarrow X_n; D(x) = 1] - \Pr[x \leftarrow Y_n; D(x) = 1] \right|.$$

We say that the *computational distance* w.r.t. D between X_n and Y_n is $\delta_D(X_n, Y_n)$. We say that X_n and Y_n are *computationally indistinguishable* if for every probabilistic polynomial-time Turing machine D, for every polynomial $p(\cdot)$, and for sufficiently large n, the computational distance w.r.t. D is bounded by $1/p(n)$, namely,

$$\delta_D(X_n, Y_n) \leq \frac{1}{p(n)}.$$

The concept of Cryptographically-Secure Pseudorandom Number Generator (CSPRNG) was first formalized by Yao [13] and further

* Corresponding author.
E-mail addresses: tkoshiba@waseda.jp (T. Koshiba), zolfaghari@aoni.waseda.jp (B. Zolfaghari), bibakk@miamioh.edu (K. Bibak).

https://doi.org/10.1016/j.jisa.2023.103430

Available online 17 January 2023

developed by Blum and Micali [14]. They proved the unpredictability of the next number to be a sufficient condition for cryptographic security of a PRNG. Using the concept of indistinguishable probabilistic ensembles, a Cryptographically-Secure Pseudorandom Bit Generator (CSPRBG), which is usually used as a CSPRNG [15], can be formalized as follows.

Consider a family of functions

$$\mathcal{G}_n : \{0,1\}^{k_n} \to \{0,1\}^n,$$

where $k_n < n$. The function $\mathcal{G}_n$ induces a probability ensemble (which we denote by G_n) over $\{0,1\}^n$ as follows

$$\Pr[y \leftarrow G_n] = \Pr[y = \mathcal{G}_n(s); s \leftarrow R_{k_n}],$$

where R_i is the uniform distribution over $\{0,1\}^i$. The input s to the function $\mathcal{G}_n$ is usually called the *seed*.

Definition 1.2. We say that $\mathcal{G}_n$ is a *CSPRBG* if the function $\mathcal{G}_n$ can be computed in polynomial time and two families of probability distributions R_n and G_n are computationally indistinguishable.

Although there are other equivalent definitions for CSPRNG, the above one best suits our purpose. A typical CSPRNG $\mathcal{R}$ is described using a recursive state transition equation like

$$S^{(i)} = f_1\left(S^{(i-1)}\right). \tag{1.1}$$

In (1.1), the ith recurrence changes the inner state of the CSPRNG from $S^{(i-1)}$ into $S^{(i)}$. In this equation, $S^{(0)}$ is the predefined initial state usually referred to as the seed. Moreover, f_1 is a trapdoor function. A trapdoor function is an easily-computed function, whose inverse cannot be easily computed in the absence of some special information referred to as the trapdoor [16]. This property guarantees the cryptographic security of the PRNG [17]. In trapdoor functions, the calculation of the inverse usually depends on hard mathematical problems. Among most common hard problems traditionally used in cryptographic applications, one may refer to Discrete Logarithm Problem (DLP) [18,19] and integer factorization [20,21].

In the PRNG $\mathcal{R}$ defined by Eq. (1.1), the ith output is calculated as shown by Eq. (1.2), where f_2 is a hard-core function with respect to f_1,

$$\mathcal{O}^{(i)} = f_2\left(S^{(i)}\right). \tag{1.2}$$

A polynomial-time function $h(x)$ is called a hard-core function with respect to the function $f(x)$ if there exist no polynomial-time algorithm capable of distinguishing $(f(x), h(x))$ from $(f(x), r)$, where r is a random bit string of length $|h(x)|$ ($|h(x)|$, being the length of $h(x)$ in bits). A hard-core function can be more formally defined as follows.

Definition 1.3. Let $h : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function such that $|h(x_1)| = |h(x_2)|$ for all $|x_1| = |x_2|$. Let $l(n) = |h(1^n)|$. The function h is called a *hard-core function* of function f if the following probability ensembles X_n and Y_n are computationally indistinguishable:

$$\Pr[(a,b) \leftarrow X_n] = \Pr[(a,b) = (f(r), h(r));\ r \leftarrow R_n],$$

$$\Pr[(a,b) \leftarrow Y_n] = \Pr[(a,b) = (f(r), r');\ r \leftarrow R_n, r' \leftarrow R_{l(n)}].$$

Gennaro's CSPRNG [22] is a well-known discrete logarithmic CSPRNG (DL-CSPRNG). Its security depends on the hardness of a variant of DLP called Discrete Logarithm with Short Exponents (DLSE). In this PRNG (like many related ones), the trapdoor function as well as the hard-core function are calculated using arithmetic operations. In this paper, we propose a modified variant of Gennaro's PRNG that uses logical operations instead of arithmetic operations. We implement the proposed CSPRNG, prove its security and evaluate its performance. More importantly, we demonstrate how replacing arithmetic operations by logical operations can shift the security-performance tradeoff paradigm in the design and implementation of DL-CSPRNGs. Moreover, we demonstrate that DLSE and the standard DLP can be reduced to each other.

1.1. Goals and objectives

In DL-CSPRNGs, the trapdoor function and the hard-core function are both calculated using arithmetic operations [22,23]. The use of these operations creates a critical security-performance tradeoff in the design and implementation of these PRNGs. Designers and researchers usually maneuver in this tradeoff space via controlling the security parameter, which is associated with the key length or any other parameter affecting the harness of the underlying problem. Lengthening the security parameter will improve the security, while it lowers the performance. In this study, we attempt to enlarge the design space of DL-CSPRNGs via providing clear answers to the following questions.

- Is DLSE as hard as the standard DLP?
- Is it possible to break the tradition of using arithmetic operations in the definition of trapdoor and hard-core functions? How can this lead to the design of yet another CSPRNG?
- What is main the source of the critical security-performance tradeoff in DL-CSPRNG? Is it possible and feasible to shift the existing tradeoff paradigm in order to allow designers move more flexibly in the design space?

1.2. Contributions

Our contributions in this paper can be listed as follows.

- We demonstrate that DL-CSPRNGs can operate without commonly-used, but inefficient modular arithmetic operations via delicate use of the Mersenne primes. A Mersenne prime p is a prime number of the form $p = 2^q - 1$ for some prime q. The Lenstra–Pomerance–Wagstaff conjecture states that there are infinitely many Mersenne primes. As of today, 51 Mersenne primes are known. Also, the largest known prime number, $2^{82589933} - 1$, is a Mersenne prime. The latter number is large enough for our purposes in this paper. A Mersenne prime can be represented by a binary string, wherein all bits are equal to '1'. Via designing a high-performance variant of Gennaro's PRNG, we show how DL-CSPRNGs can make use of different operations supported by state-of-the-art CPUs, including logical shifts and other logical operations. We formally prove our PRNG to be cryptographically-secure through bridging between the finite field $\mathbb{F}_{2^n}$ and the n-dimensional vector space $(\mathbb{Z}_2)^n$. We implement and evaluate our proposed PRNG.
- Replacing word-wise arithmetic operations with bit-wise logical operations leads to two more achievements. First, logical operations can be accomplished by CPU register without the interference of the ALU, which considerably improves the performance. Second, and more importantly, this replacement gives raise to a shift in the security-performance trade-off paradigm in the realm of DL-CSPRNGs. Since the accomplishment time of logical operations does not depend on the security parameter, increasing the security parameter size will improve security without compromising performance. This allows other design objectives such as area and power consumption to be more effectively managed, which helps designers maneuver more flexibly in the tradeoff space.
- Our PRNG (like Gennaro's) is based on DLSE, which is a variant of DLP [23]. We establish an equivalence between the standard DLP and DLSE through proving them to be reducible to each other.

1.3. Organization

The rest of this paper is organized as follows.

Section 2 discusses related works in its first five subsections; Section 2.1 through Section 2.5. The sixth subsection of this section (Section 2.6) compares our work in this paper with the most relevant

work. This section clarifies the research gap we intend to cover and highlights our motivations in its last subsection (Section 2.7). Section 3 designs, discusses and implements the proposed DL-CSPRNG. Section 3.1 presents some preliminary discussions. Section 3.2 analyses the computational complexity of the DLP variant that underpins the proposed CSPRNG. Section 3.3 designs the CSPRNG, and Section 3.5 proceeds to implement it. Section 4 discusses the tradeoff space of the newly-introduced PRNG. Section 4.1 makes a minor modification to the proposed generator, and Section 4.2 analyses the impact of the modification. Section 5 evaluates the proposed PRNG in two different ways. Section 5.1 presents the results obtained from performance evaluations, and Section 5.2 reports the results of randomness tests. Lastly, Section 6 concludes the paper and suggests future works.

2. Related works

In this section, we present and overview on relevant research works in order to clarify the gap we are going to cover in our study. This paper proposes a DL-CSPRNG, which can be considered a modified variant of Gennaro's PRNG. In this section, we review relevant research works. Starting from Section 2.1, we narrow down the scope of the literature review in consecutive subsections, and make it closer to our work in this paper. We study Gennaro's PRNG, and compare it with our proposed PRNGr in Section 2.5. In Section 2.6, we review research works focusing on the security-performance tradeoff. Lastly, in Section 2.7, we highlight a niche in related work, which motivates our work in this paper.

2.1. True-Random Number Generators (TRNGs)

TRNGs have been of interest to the research community in recent years [24]. They depend on physical and natural sources of randomness such as noises [25] or uncertainties in fabrication technologies [26]. They are usually implemented in hardware using different implementation technologies including Complementary Metal–Oxide–Semiconductor (CMOS) [27] and Field-Programmable Gate Array (FPGA) [28]. Moreover, different enablers such as chaos theory [29] and information theory [30] are used in the design of TRNGs. Especially, the use of photonics in the design of TRNGs, is a recent research trend [31,32]. This technology makes it possible to design high-performance TRNGs [33]. Both security [34] and performance [35] of these random number generators have been of concern to researchers.

2.2. Non-cryptographic PRNGs

The research literature comes with several PRNGs without security proof. Different approaches have been proposed for the design of such PRNGs. To mention a few, one may refer to approaches based on modular arithmetics [36] and congruences [37], approaches based on artificial intelligence [38] and evolutionary algorithms [39], and approaches based on chaos theory [40], cellular automata [41] or recursive equations [42]. PRNGs of this type can be implemented in software [43] or hardware [44]. Although the cryptographic security of these PRNGs is not a research concern, their performance have been a focus for some researchers [45].

2.3. Cryptographically-Secure Pseudorandom Number Generators (CSPRNGs)

Many CSPRNGs have been designed using chaotic systems [46] and circuits [47]. However, several other approaches have been used for designing PRNGs of this class. As examples of these approaches, we can mention approaches based on neural networks [48] and cellular automata [49] as well as those based on elliptic curves [50], internal state permutation [51], and transmission error sampling [52]. These PRNGs can be implemented in software [53] or hardware [47]. In addition to security, other objectives such as power consumption [54] have been considered by researchers while designing PRNGs of this class.

Each CSPRNG is based on an assumption associated with the computational hardness of a mathematical problem (It is unknown whether or not such a PRNG can be constructed without any assumption). For example, deciding quadratic residuosity of prime numbers was posed as a hard problem in [55]. Later on, this problem was proved to be as hard as integer factorization [15]. A CSPRNG based on this problem was proposed in [56]. The authors of [57] proposed a modification to the PRNG introduced in [56]. The modified PRNG depends on the problem of deciding quadratic residuosity modulo composite numbers with unknown factorizations.

2.4. Discrete Logarithmic Cryptographically-Secure Pseudorandom Number Generators (DL-CSPRNGs)

Blum and Micali [14] proposed the first CSPRNG based on exponentiation modulo a prime as a trapdoor function. The inverse of this function requires the calculation of discrete logarithms which is assumed to be a computationally-hard problem. These researchers also coined the notion of hard-core functions, which build a framework for constructing CSPRNGs along with trapdoor functions.

Different variants of DLP have been used in the design of CSPRNGs. Among these variants, one may refer to [22,23,58] as well as Elliptic Curve Discrete Logarithm Problem (ECDLP) [59–61].

2.5. The most relevant: Gennaro's DL-CSPRNG

Our CSPRNG can be considered as a modification to Gennaro's PRNG [22,58], which is an improvement to the one proposed earlier by Patel and Sundaram [23]. The mentioned PRNGs are both based on the hardness of a variant of DLP. This variant of the problem searches for discrete logarithms with short exponents over $\mathbb{Z}_p^*$, where p is a safe prime. A prime number of the form $p = 2q + 1$ is called a safe prime if q is also prime. In such a case, q is referred to as a Sophie Germain prime (named after the French mathematician Sophie Germain). An existing conjecture states that there are an infinite number of Sophie Germain primes, but this is still unproven. The advantage of a safe prime p (from a computational point of view) is that the modulus is as small as possible relative to p.

The technical essence in Gennaro's PSRNG lies upon the following proposition.

Proposition 2.1 ([22]). *Let $A_i = \{(g, g^x) \mid x \in B_i\}$, where $B_i = \{2^i u \mid 0 \leq 2^i u < p\} \cup \{1+2^i u \mid 0 \leq 1+2^i u < p\}$. Then, A_0 and A_{n-c} are computationally indistinguishable on the assumption that DLP over $\mathbb{Z}_p^*$ with short exponents is computationally-intractable in the worst case.*

Let g be a generator of $\mathbb{Z}_p^*$. Let $f_1 : \mathbb{Z}_{p-1} \to \mathbb{Z}_p^*$ and f_2 be the following functions:

$$f_1(s) = (g^{2^{n-c}})^{s-(s \bmod 2^{n-c})} g^{lsb_1(s)} \bmod p,$$
$$f_2(s) = lsb_{n-c-1}(msb_{n-1}(s)),$$

where elements in $\mathbb{Z}_{p-1}$ or $\mathbb{Z}_p^*$ can be identified with their binary representations of the corresponding natural numbers in the standard encoding, $lsb_k(s)$ is the least significant k bits of s and $msb_k(s)$ the most significant k bits of s. Let s_0 be a random seed and $s_i = f_1(s_{i-1})$ and $o_i = f_2(s_i)$ for each i. Then the output sequence $o_1, o_2, \ldots$ is a pseudorandom sequence on the assumption that the DLSE problem is computationally-intractable in the worst case.

The mathematical description of our PRNG is similar to that of Gennaro's, but we eliminate the need for ALU-intensive arithmetic computations, and replace them with register-level logical operations. To this end, we first incorporate some techniques proposed in [62,63]

into the construction of Gennaro's DL-CSPRNG and devise a new PRNG based on DLP over finite groups of prime order. Next, via adopting finite groups of Mersenne prime order, we improve the performance of the new PRNGs.

Specifically speaking, we utilize the multiplicative group $\mathbb{F}_{2^n}^*$ of Mersenne prime order. We take the security parameter n as a Mersenne exponent so that $2^n - 1$ becomes a Mersenne prime. The Mersenne primality plays an important role in our PRNG. The use of Mersenne primality allows us to bridge between the finite field $\mathbb{F}_{2^n}$ and the n-dimensional vector space $(\mathbb{Z}_2)^n$. Some properties of Mersenne primality have already been utilized in the construction of (non-cryptographic) PRNGs, e.g., Mersenne Twister [64]. We assume that p is a safe prime in the sense of van Oorschot and Wiener [65] and $c = \omega(\log n)$. Actually, we show that some operation over the group is computable in almost constant time under the current CPU architecture, though its theoretical cost depends on the security parameter from the viewpoint of computational complexity theory. We dexterously utilize the above property to efficiently generate pseudorandom bits for the current cryptographic systems.

The security of our newly proposed PRNG relies on the hardness of DLSE over $\mathbb{F}_{2^n}^*$ of Mersenne prime order. In general, DLP over $(\mathbb{Z}_2[t]/\varphi(t))^*$ (where $\varphi(t)$ is an irreducible polynomial of degree n) is rather easier than over $\mathbb{Z}_p^*$ (where p is a prime number such that $p = 2q - 1$ and q is also a prime number.) DLP over fields of characteristic 2 has been still an intractable problem (The best algorithm in the asymptotical sense is due to Coppersmith [66] and the best record is given by Thomé [67]).

Let us consider an $n \times n$ matrix over $\mathbb{Z}_2$ such that $\mathrm{ord}(M) = 2^n - 1$. Then we can take $\langle M \rangle$ as a representation of $\mathbb{F}_{2^n}^*$. Matsumoto and Nishimura [64] specifies a matrix M such that applying M to a vector is implementable in several steps of logical and shift CPU operations and $\mathrm{ord}(M) = 2^n - 1$ in order to construct Mersenne Twister. Note that since Mersenne Twister is a linear PRNG, it is not suitable for cryptographic use. DLP over $\langle M \rangle$ is a search problem to find z such that $M_1 = M_2^z$ for given $M_1, M_2 \in \langle M \rangle$. In this paper, we study the following variant: find z such that $y = xM_1^z$ for given a pair x, y of n-dimensional vectors and $M_1 \in \langle M \rangle$. To produce pseudorandom bits, we use the matrix M in the following way:

$$y = xM^z.$$

If we fix a value of z, then M^z is still a linear mapping. We suppose that z can take super-polynomially many values and this enables us to avoid a weakness caused by the linearity. To make full use of the fast implementability of computing xM within several steps of CPU operations, we consider the following computation of xM^z: let $x \leftarrow xM$ and repeat this process z times naively. In the normal sense, this procedure is an exponential-time algorithm and thus considered to be useless. However, since we can take a small value as z and the above procedure is still faster under the concrete parameter setting, we can generate pseudorandom bits with high speed. (Note that our PRNG satisfies the definition of CSPRNG because there exists another polynomial-time (but practically slow) algorithm to generate pseudorandom bits.) Moreover, our variant of the DLP over $\mathbb{F}_{2^n}^*$ is based not on the standard representation (i.e., $(\mathbb{Z}_2[t]/\varphi(t))^*$) but on a different representation defined by some linear mapping M. Thus, we discuss relations between the standard DLP and our variant.

As we mentioned, the operation (i.e., applying M to a vector) is computable in almost constant time. This implies that we can set a larger value on the security parameter. We usually take 1024 or 2048 as a value of the security parameter for the discrete logarithmic cryptosystems. In our PRNG, we can take some value larger than 10 000 as a value of the security parameter without much increase of its running time. Thus, this makes room for relaxing other parameters such as the length of short exponent. In order to demonstrate the performance of our new PRNG, we concretely set the parameters for the practical use and experiment on the efficiency and statistical properties.

2.6. The security-performance tradeoff

CSPRNGs are typically computation-intensive as they depend on arithmetic operations for calculating the trapdoor function as well as the hard-core function. An attempt at resolving the tradeoff between provable security and performance has been made in [68,69] via introducing high-performance, yet cryptographically-secure PRNGs based on the security of block ciphers. Yarrow is another CSPRNG based on block ciphers [70]. This PRNG was superseded by Fortuna [71,72] later on. Other researchers have worked on parallel implementations to achieve higher performance [73]. In another attempt, Gollmann [74,75] proposed an architecture based on cascaded Linear Feedback Shift Registers (LFSRs) to improve performance without compromising security. An LFSR is a shift register with a feedback loop containing some logical XOR gates implementing modulo-2 addition. LFSRs have found their applications in many cryptographic schemes [8,76].

2.7. Motivations

The critical tradeoff between security and performance is the source of many complexities in the design of DL-CSPRNGs. This complexity makes it difficult to enlarge the design space of such PRNGs. Some researchers have attempted to maneuver in the existing tradeoff space via improving performance without compromising security. Solutions provided by these researchers adversely affect other design objectives such as area and power consumption. The above discussions clarify the need for a tradeoff paradigm shift in this realm. However, to the best of our knowledge, there is no research focusing on a such a paradigm shift in this area. This niche motivates our work in this paper.

3. The proposed CSPRNG: Design and implementation

In this section, we introduce, design and implement our proposed CSPRNG. Section 3.1 presents some preliminary discussions. Section 3.2 discusses the computational complexity of the DLP underlies our PRNG. Section 3.3 designs the PRNG. Lastly, Section 3.5 proposes an implementation for the PRNG.

3.1. Preliminaries

As the basis of the security of our PRNG, we assume that some variant of the DLSE problem over $\mathbb{F}_{2^n}^*$ of Mersenne prime order is computationally intractable in the worst case. Namely, if there exists a probabilistic polynomial-time algorithm that distinguishes our pseudorandomness from the true randomness, then there exists a probabilistic polynomial-time algorithm that solves any instances of the variant of the DLSE problem. A Mersenne prime p is of the form $p = 2^{p'} - 1$ where p' is also a prime, which is called a *Mersenne exponent*. For the standard realization of $\mathbb{F}_{2^n}^*$, the following is well known: $(\mathbb{Z}_2[t]/\varphi(t))^*$, where $\varphi(t) \in \mathbb{Z}_2[t]$ is a primitive polynomial of degree n. We write, for each Mersenne exponent n, $\mathsf{G}_n = (\mathbb{Z}_2[t]/\varphi(t))^*$. (Note that whenever we say G_n, some primitive polynomial that forms $\mathbb{F}_{2^n}$ is implicitly assumed.) We note that since each G_n is a multiplicative group of odd prime order, i.e., a cyclic group, where every non-identity element is a generator of the group, DLP and the DLSE problem over G_n are naturally defined.

In this paper, we consider the DLP in a slightly different way. Let $x \in \mathsf{G}_n$ be a non-identity element. We can naturally identify x with an n-dimensional row vector over $\mathbb{Z}_2$. Let M be an $n \times n$ matrix over $\mathbb{Z}_2$. Matsumoto and Nishimura [64] gave a way to construct matrices M such that $\{xM^i : 0 \le i \le 2^n - 2\}$ is isomorphic to G_n. We call such a special matrix *generator matrix*. This means that for every pair x and y of n-dimensional non-zero row vectors over $\mathbb{Z}_2$ there exists some unique exponent i satisfying $y = xM^i$. Let $\mathsf{M}_n = \{M^i : 0 \le i \le 2^n - 2\}$. Computing z such that $y = xM^z$ from given M, x and y is considered as a variant of DLP. Let mDLP denote our variant of the DLP and DLP the DLP over G_n, i.e., the standard DLP. We discuss computational relations between mDLP and DLP in the next subsection. We also denote, by mDLSEP and DLSEP, the short exponent variants of mDLP and DLP, respectively.

3.2. Computational complexity of the underlying DLP variant

Here, we consider the computational complexity of mDLP and mDLSEP. Recall that mDLP is a search problem to find z for given x, M, xM^z. Let I be the identity matrix of dimension n over $\mathbb{Z}_2$. Let $cp(t) = \det(M - tI)$ be the characteristic polynomial of the matrix M. We note that $\mathrm{degree}(cp(t)) = n$ and $cp(M) = O$. Hence

$$\forall i \geq 0, M^i = a(M), \quad \text{where } a(t) = t^i \bmod cp(t)$$

and thus M^i is a linear combination of $\{I, M, M^2, \ldots, M^{n-1}\}$. Since by the assumption that M_n has cardinality $2^n - 1$ it follows that the matrices in $\{I, M, M^2, \ldots, M^{n-1}\}$ are linearly independent. It also follows that given $x, M, y = xM^z$ we can compute M^z without computing z by solving the equation (w.r.t. $c_0, \ldots, c_{n-1}$) below.

$$x \sum_{i=0}^{n-1} c_i M^i = y.$$

Moreover, we can find a polynomial $b(t)$ of degree less than n such that $b(M) = M^z$. Hence mDLP is reducible to finding an integer z such that $t^z = b(t) \bmod cp(t)$. This is exactly the DLP over $(\mathbb{Z}_2[t]/cp(t))^*$. Hence, we have the following.

Theorem 3.1. *Suppose that there exists an efficient algorithm to solve* DLP *on average. Then, there exists an efficient algorithm to solve* mDLP *on average.*

Next, we consider the reducibility from DLP to mDLP. What we have is only the reducibility in the worst case. By considering the construction of a matrix M such that, for all x, $xM = xg$, where g is a generator, we have the following.

Theorem 3.2. *Suppose that there exists an efficient algorithm to solve* mDLP *in the worst case. Then, there exists an efficient algorithm to solve* DLP *in the worst case.*

Thus, we can say that DLP and mDLP are equivalent problems with respect to the worst-case reduction. It is still open whether the equivalence with respect to the average-case reduction holds or not.

We mention the computational complexity of the DLP with short exponents. By a similar argument, we have the following.

Theorem 3.3. *Suppose that there exists an efficient algorithm to solve* DLSEP *on average. Then, there exists an efficient algorithm to solve* mDLSEP *on average.*

Theorem 3.4. *Suppose that there exists an efficient algorithm to solve* mDLSEP *in the worst case. Then, there exists an efficient algorithm to solve* DLSEP *in the worst case.*

3.3. Design

Our PRNG can be regarded as a variant of Gennaro's. We will modify his PRNG stepwise. Recall $c = \omega(\log n)$. (That is, c grows faster than $a \log n$ for any constant a and 2^c grows faster than any polynomial in n.) First, we consider the replacement of the underlying group $\mathbb{Z}_p^*$ for Gennaro's PRNG with the group G_n of Mersenne prime order. Let

$$stat_1(z, g) = (g^{msb_c(z) \| 0^{n-c}}, g),$$
$$out_1(z, g) = lsb_{n-c}(z),$$

where $g \in \mathsf{G}_n$ is a generator and $u \| v$ denotes the concatenation of binary strings u and v. (Remember that any elements in $\mathbb{F}_{2^n}$ can be identified with natural numbers or binary strings from the context.) Start with a random seed $z^{(0)} \in \mathbb{F}_{2^n}$. Let $s_0 = (z^{(0)}, g)$ and set $s_i = (z^{(i)}, g) = stat_1(s_{i-1})$ and $o_i = out_1(s_i)$ for each i. Then $o_1, o_2, \ldots$ is a pseudorandom sequence on the assumption that DLSEP over G_n is computationally intractable in the worst case. Though the above modification is straightforward, we have to prepare some technical claim to guarantee its security. To complete a proof of the security for the above PRNG, the following lemma shown independently by Koshiba and Kurosawa [62] and Gennaro, Krawczyk and Rabin [63] is sufficient instead of Proposition 2.1.

Lemma 3.5 (*[62,63]*). *Let G be a group of (any) prime order q and let $A_i = \{(g, g^x) \mid x \in B_i\}$, where $B_i = \{2^i u \mid 0 \leq 2^i u < q\}$. Then, A_0 and A_{n-c} are computationally indistinguishable on the assumption that DLP over G with short exponents is computationally-intractable in the worst case.*

While the PRNG induced from the pair of functions $stat_1$ and out_1 is cryptographically secure, its speed is as slow as any other number-theoretic PRNGs. Thus, we consider another modification in the following. (Note that the modification is possible since our underlying group is of a Mersenne prime order.) Let

$$stat_2(v \| u, g) = (g^{0^{n-c} \| u}, g),$$
$$out_2(v \| u, g) = msb_{n-c}(v \| u) = v.$$

Using the above pair of functions $stat_2$ and out_2, we can similarly generate a pseudorandom sequence on the same computational assumption. From the technical point of view, this implies that taking the most significant bits is a "hard-core" function of the discrete logarithmic function as well as the consecutive least significant bits when we take groups of Mersenne prime order as underlying group for a PRNG. This property is peculiar to groups of Mersenne prime order and comes from the fact if $w \| b$ satisfies that $y = g^{w \| b}$ for some y then $b \| w$ satisfies that $y = (g^2)^{b \| w}$ and vice versa, where b denotes a bit.

Now, we are ready to describe our new PRNG. Let

$$stat_3(v \| u, x, M) = (xM^{0^{n-c} \| u}, x, M),$$
$$out_3(v \| u, x, M) = msb_{n-c}(v \| u) = v.$$

Start with a random seed $z^{(0)}$ and x, which is a pair of n-dimensional non-zero vectors over $\mathbb{Z}_2$. We also have to select a matrix $M \in \mathsf{M}_n$, which we will show in the next subsection. Let $s_0 = (z^{(0)}, x, M)$ and set $s_i = (z^{(i)}, x, M) = stat_3(s_{i-1})$ and $o_i = out_3(s_i)$ for each i. Then we output $o_1, o_2, \ldots$ as a pseudorandom sequence. We denote this PRNG by SR. Consequently, we obtain the following.

Theorem 3.6. *The PRNG SR is cryptographically-secure on the assumption that* mDLSEP *is computationally intractable in the worst case.*

By Theorem 3.4, the security of SR can be guaranteed even on the standard intractability assumption.

Corollary 3.7. *The PRNG SR is cryptographically-secure on the assumption that* DLSEP *is computationally intractable in the worst case.*

We will show that, in the next subsection, the cost of computing $stat_3$ and out_3 are quite low under the current CPU architecture and thus it is possible to generate pseudorandom sequences with high-speed in a practical sense.

3.4. Seed management

The proposed RNG uses the seed generation algorithm introduced in [77]. This algorithm has been used in some other research works as well [78]. This algorithm is as follows.

1. A 60-digit Hexadecimal ($[0-9, A-F]$) string $H = \overline{h_1 h_2 \cdots h_{60}}$ is chosen manually or randomly. Then H is decomposed into 10 parts H_1 through H_{10}, such that $\forall i \in [1-10] : H_i = \overline{h_{6*(i-1)+1} \cdots h_{6*(i-1)+6}}$.

2. The following calculations are performed considering 32-bit floating-point arithmetic with 10^{-7} precision,

$$x_{n+1} \leftarrow rem\left(\left(1-\alpha x_n^2+\beta x_{n-1}\right) * 101, 1\right), \tag{3.1}$$

$$y_{n+1} \leftarrow rem\left(\gamma \sin\left(\pi x_{n-1} y_{n-1}\right) * 101, 1\right). \tag{3.2}$$

Here n is the number of iteration, and x_n and y_n represent the state of the recursive algorithm, with $x_0 = rem\left(H_7 + H_8, 1\right)$ and $y_0 = rem\left(H_9 + H_{10}, 1\right)$ being initial conditions. Also, *rem* is the remainder operation, and $\alpha = rem\left(H_1 + H_2, 1\right)$, $\beta = rem\left(H_3 + H_4, 1\right)$ and $\gamma = rem\left(H_5 + H_6, 1\right)$ are control parameters.

3. The following computations are performed,

$$X_R = round\left(x_n * \left(2^{7+1} - 1\right)\right), \tag{3.3}$$

$$X_R = round\left(x_n * \left(2^{7+1} - 1\right)\right), \tag{3.4}$$

where *round* is the round-to-the-nearest operation.

In each step of the above algorithm, concatenating x and y creates 16 pseudorandom bit. Thus, seven parallel instances of the algorithm can create a 128-bit seed.

3.5. Implementation

First, we discuss how to select a generator matrix M. We mentioned that Matsumoto and Nishimura [64] gave a way to construct a matrix M such that $\{xM^i : 0 \le i \le 2^n - 2\}$ is isomorphic to G_n. Their construction of such a matrix M is so as to compute xM as simply as possible, especially for 32-bit CPU machines. In order to give a way to implement our PRNG, we review a (non-cryptographic) PRNG, called Mersenne Twister (MT), proposed by Matsumoto and Nishimura [64]. MT is based on the following linear recurrence

$$\mathbf{x}_{k+d} := \mathbf{x}_{k+m} \oplus (\mathbf{x}_k^u \parallel \mathbf{x}_{k+1}^l)A, \quad (k = 0, 1, \ldots), \tag{3.5}$$

where each $\mathbf{x}_i$ is a 32-bit string. There are several constants to specify the MT algorithm; an integer d, which is the degree of the recurrence, an integer r ($0 \le r \le 31$), an integer m ($1 \le m \le d$), and a 32×32 matrix A over $\mathbb{Z}_2$. We have to give $\mathbf{x}_0, \ldots, \mathbf{x}_{d-1}$ as initial seeds. In the right-hand side of the recurrence, $\mathbf{x}_k^u$ means the upper $32 - r$ bits of $\mathbf{x}_k$, and $\mathbf{x}_{k+1}^l$ the lower r bits of $\mathbf{x}_{k+1}$.

Matsumoto and Nishimura [64] choose a matrix A of the following form so that multiplication by A becomes very fast. Actually, their choice is written as

$$A = \begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ a_{31} & a_{30} & \cdots & \cdots & a_0 \end{pmatrix},$$

where the unspecified entries are 0's and each of $a_{31}, \ldots, a_0$ is either 0 or 1, then the calculation of $\mathbf{x}A$ can be done by using bit operations only:

$$\mathbf{x}A = \begin{cases} \mathrm{shift right}(\mathbf{x}) & \text{if } x_0 = 0, \\ \mathrm{shift right}(\mathbf{x}) \oplus \mathbf{a} & \text{if } x_0 = 1, \end{cases} \tag{3.6}$$

where $\mathbf{a} = a_{31} \parallel a_{30} \parallel \cdots \parallel a_0$.

The state transition is written by the following linear mapping M.

$$M : (\mathbf{x}_0^u, \mathbf{x}_1, \ldots, \mathbf{x}_{d-1}) \mapsto (\mathbf{x}_1^u, \mathbf{x}_2, \ldots, \mathbf{x}_d)$$

By a general theory of linear recurrence, each entry of the $(32d - r)$-dimensional vector is a linear recurring sequence satisfying the recurrence corresponding to the characteristic polynomial $\varphi_M(t)$ of the mapping M. The sequence attains the maximal period $2^n - 1 = 2^{32d-r} - 1$ if and only if $\varphi_M(t)$ is primitive, i.e., t generates the multiplicative group $(\mathbb{Z}_2[t]/\varphi_M(t))^*$. Since the characteristic polynomial is of the form

$$\begin{aligned} \varphi_M(t) &= \det(M - tI) \\ &= (t^d + t^m)^{32-r}(t^{d-1} + t^{m-1})^r + a_0(t^d + t^m)^{32-r}(t^{d-1} + t^{m-1})^{r-1} \\ &\quad + \cdots + a_{r-2}(t^d + t^m)^{32-r}(t^{d-1} + t^{m-1}) + a_{r-1}(t^d + t^m)^{32-r} \\ &\quad + a_r(t^d + t^m)^{32-r-1} + \cdots + a_{32-2}(t^d + t^m)^{32-r-1} + a_{32-1} \end{aligned}$$

we have to choose parameters $a_0, \ldots, a_{31}$ and m such that $\varphi_M(t)$ becomes a primitive polynomial. (Though the choice of parameters can be done with time $O(n^3)$, this procedure is actually the most time-consuming part in a practical sense. Fortunately, we can run this part as pre-processing for generating pseudorandom bits.)

A great advantage is that the state vector assumes every bit-pattern in the $(32d - r)$-dimensional vector and appears once in a period, except for the zero state. Consequently, the sequence is uniformly distributed in the n-dimensional space. This also means that MT is not only a pseudorandom 'number' generator but also a pseudorandom 'bit' generator. Since any initial seed except for zero lies on the same orbit, the choice of an initial seed does not affect the randomness for the whole period.

From Eqs. (3.5) and (3.6), it is easy to see that an implementation of "$x \leftarrow xM$" is possible in several steps of logical and shift operations on 32-bit words by using the standard pointer technique in the programming. We note that the computation of "$x \leftarrow xM$" does not depend heavily on the value of n. This means that we can take the value of n as large as we like. We note that the value of n must be a Mersenne exponent and there are many restrictions on the real computer architecture. For example, our generator with the setting $n = 19937$ is efficient enough, while the standard arithmetic computation over 19937-bit integers is not suitable for practical cryptographic purpose. In case of $n = 19937$, parameters $m = 397$ and $\mathbf{a} = \mathtt{0x9908B0DF}$ for example make $\varphi_M(t)$ primitive.

For more details for the MT algorithm, see [64]. Some variants of MT and parameters for 64-bit CPUs can be found in [79].

To implement our PRNG $\mathcal{SR}$, we have to mention a way to compute xM^z, where $|z| = c$. For the parameter $n = 19937$, we may set c as small as 14. We consider the following process: let $x \leftarrow xM$ and repeat this process z times naively. The former part "$x \leftarrow xM$" can be implemented within several steps of CPU instructions and the number of repetitions is at most 2^{14}. We will see that this implementation is practically good in the next section.

4. Tradeoff space

Our main purpose in this paper was to reshape the tradeoff space in DL-CSPRNGs. In this section, we evaluate the impact of design parameters on the design objectives of our proposed generator. This way, we clarify the tradeoff space in the proposed generator.

4.1. A minor modification

For our PRNG $\mathcal{SR}$, the value of c must be small to generate pseudorandom bits fast. However, if c is too small then the mDLSEP becomes easier. This means that it is difficult to choose a good value of c. To overcome this difficulty, we consider a further minor heuristic modification. Let

$$\begin{aligned} stat_4(v \parallel u, x, M) &= (\pi(x)M^{0^{n-c}\parallel u}, \pi(x), M), \\ out_4(v \parallel u, x, M) &= v, \end{aligned}$$

where π is an efficiently computable length-preserving permutation. We may use another generator matrix M', which can be selected by the same way to select M, as a possible candidate of π and compute $\pi(x) = xM'$. Using the pair of functions $stat_4$ and out_4, we can similarly generate a pseudorandom sequence on the same computational assumption. We denote, by $\mathcal{SR}'$, the PRNG induced from the functions $stat_4$ and out_4.

4.2. Tradeoffs

As we have seen, our PRNGs $\mathcal{SR}$ and $\mathcal{SR}'$ are variants of Gennaro's PRNG. A typical value for the security parameter n in case of Gennaro's PRNG and other number-theoretic ones is either $n = 1024$ or $n = 2048$. As we know, the value $n = 1024$ results from the balance of the security and the speed performance under the current computers. The bottleneck with respect to the speed of usual number-theoretic PRNG is the heaviness of arithmetic computation over integers of n-bit string. On the other hand, our PRNGs $\mathcal{SR}$ and $\mathcal{SR}'$ do not require arithmetic computation. Thus, we can take a larger value for n. Even if we take a large value for n, the speed for generating pseudorandom bits is almost constant as long as we keep the parameter c small. Actually, we implemented $\mathcal{SR}'$ with $n = 19937$ and $c = 14$. The choice $n = 19937$ is so as to select other MT parameters in reasonable time, say, a day; the choice $c = 14$ is so as to generate pseudorandom bits fast to be used in fast cryptographic primitives. Furthermore, we should mention the length $c = 14$ of the exponents. By taking a quite large value of security parameter n, we pay the price for the short exponents. Fortunately, due to the construction of our PRNG $\mathcal{SR}'$, we can avoid the increase of the running cost for generation of pseudorandom bits, even when the security parameter is pretty large. In short, we can say that our PRNG dexterously introduces advantageous points in a new tradeoff which has never appeared in the literature before.

Exactly speaking, the security is guaranteed only when generator matrices are uniformly distributed. For the sake of fast implementation, we have to compromise the provable security. As a generator matrix, we use a fixed matrix M. (Note that what we mean by "fixing a matrix M" is the use of the special matrix that comes from the MT algorithm though the provable security requires that generator matrices should be distributed over M_n. However, M_n implicitly assumes its characteristic polynomial and we can choose MT parameters so as to change the underlying characteristic polynomial. Hence, "fixing a matrix M" does not mean "considering only one matrix M".)

By taking other concrete parameters into account, we choose the following setting.

$$stat(z, x) = (\pi(x)M^{lsb_{14}(z)}, \pi(x)),$$
$$out(z, x) = msb_{4096}(z).$$

We adopt another fixed linear mapping M' as $\pi(x) = xM'$. Due to the sacrifice of too short exponents, we abandon some hard-core bits. In this case, we use a fraction of the hard-core bits less than a quarter.

For an implementation of the exponentiation $y = xM^{lsb(z)}$, we multiply the base point x by the matrix M naively $lsb(z)$ times. Since $|lsb(z)| = \omega(\log n)$, this algorithm is a super-polynomial-time one. (We note that there is a polynomial-time algorithm to compute $y = xM^{lsb(z)}$.) However, a super-polynomial-time algorithm with concrete parameters may be faster than any other polynomial-time algorithm with the same parameters and this is the case.

5. Evaluations

In this section, we evaluate our proposed generator in two different ways. In Section 5.1, we evaluate the performance of the generator. In Section 5.2, we apply statistical randomness tests.

5.1. Performance evaluation

Under those parameters discussed in the previous section, we have prototypically implemented $\mathcal{SR}'$ in C language (without any special library) on an 800 MHz Pentium III. We ran $\mathcal{SR}'$ using randomly chosen seeds and obtained on average 55.0×10^6 pseudorandom bits per second. A comparison with state-of-the-art RNGs implemented on Pentium shows that our RNG is comparable with them in terms of performance. We use a rough measure for this comparison to which we refer as the *Bit Per Clock Pulse* (BPCP). BPCP is defined as the number of pseudorandom bits per CPU clock pulse. This measure is equal to the proportion of the number of generated pseudorandom bits per second to the clock rate of the CPU. This measure is equal to $\frac{55.0\times10^6}{800\times10^6} = 0.06875$. Aiello, Rajagopalan and Venkatesan proposed a fast PRNG based on block ciphers [69] and reported that their sample construction of PRNG based on DES and MD5 ran at about 20 Mb/s on a 233 MHz Pentium II. BPCP is equal to 0.08584 for their proposed PRNG. As seen from the BPCPs, our RNG reduces the performance by 20% compared to the one proposed in [69]. Part of this reduction can be compensated using compiler optimization techniques. Moreover, a more accurate comparison measure may reveal that the real performance reduction is less than 20%. There is also a more recent RNG implemented on Pentium (using C) that can be compared with our $\mathcal{SR}'$. This RNG has been reported in [80]. Although the authors of [80] have not directly reported the performance of their RNG, a simple calculation shows that they have managed to generate 124×10^6 pseudorandom bits per second on a 1.6 GHz Pentium. This leads to a BPCP of 0.0775. Compared to the latter RNG, our $\mathcal{SR}'$ leads to a performance reduction of 11%. Thus, we can state that though we utilize a number-theoretic assumption, our generator $\mathcal{SR}'$ produces pseudorandom bits pretty fast.

5.2. Randomness test

5.2.1. Evaluation environment and methodology

In the presence of a sufficiently-large security parameter, the existence of provable security eliminates the need for statistical randomness tests in the design of CSPRNG. However, in our case, we have fixed a parameter called the generator matrix. Thus, we have considered randomness test to show that we have not sacrificed randomness in our security-performance tradeoff paradigm shift.

Our PRNG generator with the concrete parameters was subject to a typical statistical test, the National Institute of Standards and Technology statistical test suite for random and pseudorandom number generators for cryptographic applications (NIST SP 800-22) [81]. This suite contains 15 delicately-chosen statistical tests that focus on a variety of different types of non-randomness that could exist in a sequence. Each test returns a level of significance referred to as the *P-Value*, which is compared with 0.01 taken as the *threshold significance*. A P-value above 0.01 returned by a given test, indicates that the stream has been deemed random by the test. The 15 tests in the NIST SP 800-22 test suite are briefly introduced in Table 1.

Some research works, especially those focusing on chaotic encryption, have used some other randomness tests [82,83]. However, NIST SP 800-22 looks quite enough for non-chaotic RNGs (because of its comprehensiveness).

For each test in the NIST SP 800-22 suite, we randomly chose 300 seeds and generated 300 binary sequences of 1 024 000 bits using our CSPRNG $\mathcal{SR}'$. The reference distribution for each test statistic is either a normal distribution or a χ^2 distribution.

This setting is the same as the randomness tests for the AES selection. We note that we do not have to pay much attention to select seeds since pseudorandom sequences generated by our RNG are not sensitive to the seeds. However, the procedure for seed management has been discussed in Section 3.4.

The proposed DL-CSPRNG has been implemented using C and the NIST randomness tests have been conducted on the generated stream of random numbers using Python 3.6.

5.2.2. Evaluation results

The P-Value obtained from each of the 15 tests in the NIST SP 800-22 suite can be found in Table 2.

The fourth entry of each row in Table 2 shows whether the test mentioned in the second entry has deemed the RNG random or not. As seen in the table, our CSPRNG has been deemed random by all tests!

Table 1
The tests in the NIST test suite.

Test	Explanation
Frequency test (Monobit) and Frequency test within a block	Check the significance of the deviation of global and block-level bias of 0s to 1s.
Runs test and Test for longest runs of ones in a block	Check the number of global and block-level runs of constant bit values in the input stream. The goal is to check the stream oscillate significantly faster or slower than expected.
Binary matrix rank test	Considers the input stream as a series of matrices and checks the significance of linear dependence between the matrices.
Discrete Fourier transform (Spectral) test	Checks the periodic features of the stream using a frequency-domain analysis.
Non-overlapping template matching test	Searches for some particular, aperiodic, patterns in the stream.
Overlapping template matching test	Searches for predefined target substrings in the stream.
Maurer's universal statistical test	Checks whether the stream can be significantly compressed in a lossless way.
Linear complexity	Calculates the linear complexity of the stream and compares its deviation from some predefined random streams.
Serial test	Searches in the input string for some specific patterns of fixed lengths. Checks if the patterns occur in the stream significantly more/less frequently than expected.
Approximate entropy test	Searches the input stream, calculates the occurrence frequencies of patterns of lengths m and $m+1$ (for different values of m) and calculates their difference.
Cumulative sums test	Searches for overall bias towards ones or zeros in the input stream via converting all 0s to -1s and calculative cumulative sums of the result.
Random excursions test and Random excursions variant test	Check whether the number of visits to a specific state within a random walk exceeds predefined thresholds.

Table 2
NIST test results.

No.	Test	P-Value	Random
1	Frequency test (Monobit)	0.10139947630582019	Yes
2	Frequency test within a block	0.11022750018311041	Yes
3	Run test	0.34891430107143900	Yes
4	Longest run of ones in a block	0.26514113400715193	Yes
5	Binary matrix rank test	0.41129939192050958	Yes
6	Discrete Fourier transform (Spectral) test	0.52964558440337472	Yes
7	Non-overlapping template matching test	0.24301525001550438	Yes
8	Overlapping template matching test	0.67120034216812000	Yes
9	Maurer's universal statistical test	0.31481563123129101	Yes
10	Linear complexity test	0.32290109224410570	Yes
11	Serial test	0.20209006607334941	Yes
12	Approximate entropy test	0.41175253945781126	Yes
13	Cumulative sums test	0.68170314140237612	Yes
14	Random excursion test	0.19611408410552007	Yes
15	Random excursion variant test	0.38423933203735900	Yes

6. Conclusions and further works

In this paper, we first showed that (our variant of) Discrete Logarithm with Short Exponents (DLES) and the standard DLP program are reducible to each other. This confirms the hardness of DLES. Then we presented a new sight on the DLP that allows incomparably fast exponentiations using Mersenne primes. Next, we replaced arithmetic operations by logical operations in a commonly-used DLES-based PRNGs to introduce an improved variant. We evaluated the performance and the randomness of our proposed generator in addition to proving its security. Our most important achievement in this research is a paradigm shift in the security-performance tradeoff in the design of DL-CSPRNG. This is achieved thanks to the constant accomplishment time of logical operations in the state-of-the-art CPUs. Our work in this paper can be continued via reshaping the tradeoff space in CSPRNGs based on other DLP variants such as Elliptic Curve Discrete Logarithm Problem (ECDLP).

CRediT authorship contribution statement

Takeshi Koshiba: Formal analysis, Writing – original draft. **Behrouz Zolfaghari:** Formal analysis, Writing – original draft. **Khodakhast Bibak:** Formal analysis, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

The authors would like to thank the editor and the referees for carefully reading the paper, and for their useful comments which helped improve the paper. All authors have read and agreed to the published version of the manuscript.

Funding

TK is supported in part by a JSPS Grant-in-Aids for Scientific Research (A) No. 21H04879, and for Challenging Exploratory Research No. 19K22849 and MEXT Quantum Leap Flagship Program (MEXT Q-LEAP) Grant No. JPMXS0118067285 and JPMXS0120319794.

References

[1] Shimizu K, Suzuki T. Finely tunable bitcuboid-based encryption with exception-free signed binarization for jpeg standard. IEEE Trans Inf Forensics Secur 2021;16(1):4895–908.
[2] Baee MAR, Simpson L, Boyen X, Foo E, Pieprzyk J. Ali: Anonymous lightweight inter-vehicle broadcast authentication with encryption. IEEE Trans Dependable Secur Comput 2022;1, (Early Access Article).
[3] Ge C, Susilo W, Baek J, Liu Z, Xia J, Fang L. A verifiable and fair attribute-based proxy re-encryption scheme for data sharing in clouds. IEEE Trans Dependable Secur Comput 2021;1, (Early Access Article).
[4] Cilardo A, Mazzocca N. Exploiting vulnerabilities in cryptographic hash functions based on reconfigurable hardware. IEEE Trans Inf Forensics Secur 2013;8(5):810–20.
[5] Zhang P, Huang T, Sun X, Zhao W, Liu H, Lai S, Liu JK. Privacy-preserving and outsourced multi-party k-means clustering based on multi-key fully homomorphic encryption. IEEE Trans Dependable Secur Comput 2022;1–12, (Early Access Article).
[6] Kaminaga M, Suzuki T, Fukase M. Determining the optimal random-padding size for rabin cryptosystems. IEEE Trans Inf Forensics Secur 2019;14(8):2232–42.
[7] Zhou J, Liu X, Au OC, Tang YY. Designing an efficient image encryption-then-compression system via prediction error clustering and random permutation. IEEE Trans Inf Forensics Secur 2014;9(1):39–50.
[8] Zolfaghari B, Bibak K, Koshiba T. From random numbers to random objects. Entropy 2022;24(7).
[9] Liu D, Liu Z, Li L, Zou X. A low-cost low-power ring oscillator-based truly random number generator for encryption on smart cards. IEEE Trans Circuits Syst II 2016;63(6):608–12.
[10] Callegari S, Rovatti R, Setti G. Embeddable adc-based true random number generator for cryptographic applications exploiting nonlinear signal processing and chaos. IEEE Trans Signal Process 2005;53(2):793–805.
[11] Zolfaghari B, Bibak K, Nemati HR, Koshiba T, Mitra P. Statistical trend analysis on physically unclonable functions: An approach via text mining. CRC Press; 2021.

[12] Li S, Liu Y, Ren F, Yang Z. Design of a high throughput pseudo-random number generator based on discrete hyper-chaotic system. IEEE Trans Circuits Syst II: Express Briefs 2022;1, (Early Access Article).

[13] Yao ACC. Theory and applications of trapdoor functions. In: Proceedings of the 23rd annual ieee symposium on foundations of computer science. Chicago, IL, USA; 1982.

[14] Blum M, Micali S. How to generate cryptographically strong sequences of pseudo-random bits. SIAM J Comput 1984;13(4):850–64.

[15] Vazirani U, Vazirani V. Efficient and secure pseudo-random number generation. In: Proceedings of 5th annual symposium onfoundations of computer science. Singer Island, FL, USA; 1984.

[16] Yao A. Theory and applications of trapdoor functions. In: Proceedings of the 23rd IEEE symposium on foundations of computer science. FOCS, Chicago, IL, USA; 1982.

[17] Håstad J, Impagliazzo R, Levin LA, Luby M. A pseudorandom generator from any one-way function. SIAM J Comput 1999;28(4):1364–96.

[18] Frey G, Muller M, Ruck H-G. The tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. IEEE Trans Inform Theory 1999;45(5):1717–9.

[19] Meidl W, Winterhof A. Lower bounds on the linear complexity of the discrete logarithm in finite fields. IEEE Trans Inform Theory 2001;47(7):2807–11.

[20] Boudot F, Gaudry P, Guillevic A, Heninger N, Thomé E, Zimmermann P. The state of the art in integer factoring and breaking public-key cryptography. IEEE Secur Priv Mag 2022;20(2):80–6.

[21] Yasuda M, Shimoyama T, Kogure J, Izu T. Computational hardness of ifp and ecdlp. Appl Algebra Eng Commun Comput 2016;27(6):493–521.

[22] Gennaro R. An improved pseudo-random generator based on discrete log. In: Proceedings of advances in cryptology. CRYPTO, Santa Barbara, California, USA; 2000.

[23] Patel S, Sundaram GS. An efficient discrete log pseudo random generator. In: Proceedings of advances in cryptology. CRYPTO, Santa Barbara, California, USA; 1998.

[24] Park J, Kim BJ, Sim J-Y. A pvt-tolerant oscillation-collapse-based true random number generator with an odd number of inverter stages. IEEE Trans Circuits Syst II: Express Briefs 2022;1, (Early Access Article).

[25] Ray B, Milenković A. True random number generation using read noise of flash memory cells. IEEE Trans Electron Devices 2018;65(3):963–9.

[26] Anandakumar NN, Sanadhya SK, Hashmi MS. Fpga-based true random number generation using programmable delays in oscillator-rings. IEEE Trans Circuits Syst II 2020;67(3):570–4.

[27] Balachandran GK, Barnett RE. A 440-na true random number generator for passive rfid tags. IEEE Trans Circuits Syst I Regul Pap 2008;55(11):3723–32.

[28] Johnson AP, Chakraborty RS, Mukhopadyay D. An improved dcm-based tunable true random number generator for xilinx fpga. IEEE Trans Circuits Syst II 2017;64(4):452–6.

[29] Pareschi F, Setti G, Rovatti R. Implementation and testing of high-speed cmos true random number generators based on chaotic systems. IEEE Trans Circuits Syst I Regul Pap 2010;57(12):3124–37.

[30] Ma Y, Chen T, Lin J, Yang J, Jing J. Entropy estimation for adc sampling-based true random number generators. IEEE Trans Inf Forensics Secur 2019;14(11):2887–900.

[31] Guo Y, Cai Q, Li P, Zhang R, Xu B, Shore KA, Wang Y. Ultrafast and real-time physical random bit extraction with all-optical quantization. Adv Photon 2022;4(3):1–7.

[32] Guo Y, Cai Q, Li P, Jia Z, Xu B, Zhang Q, Zhang Y, Zhang R, Gao Z, Shore KA, Wang Y. 40 Gb/s quantum random number generation based on optically sampled amplified spontaneous emission. APL Photon 2021;6(6):1–9.

[33] Li P, Wang Y, Zhang J-Z. All-optical fast random number generator. Opt Express 2019;18(19):20 360–9.

[34] Sunar B, Martin WJ, Stinson DR. A provably secure true random number generator with built-in tolerance to active attacks. IEEE Trans Comput 2007;56(1):134–45.

[35] Wieczorek PZ, Gołofit K. Dual-metastability time-competitive true random number generator. IEEE Trans Circuits Syst I Regul Pap 2014;61(1):134–45.

[36] Cusick T. Properties of the x/sup 2/ mod n pseudorandom number generator. IEEE Trans Inform Theory 1995;41(4):1155–9.

[37] Neuman, Merrick. Autocorrelation peaks in congruential pseudorandom number generators. IEEE Trans Comput 1976;C-25(5):457–60.

[38] Wen S, Zeng Z, Huang T, Zhang Y. Exponential adaptive lag synchronization of memristive neural networks via fuzzy method and applications in pseudorandom number generators. IEEE Trans Fuzzy Syst 2014;22(6):1704–13.

[39] Wiese K, Hendriks A, Deschenes A, Youssef B. P-rnapredict-a parallel evolutionary algorithm for rna folding: effects of pseudorandom number quality. IEEE Trans NanoBioscience 2005;4(3):219–27.

[40] Garcia-Bosque M, Pérez-Resa A, Sánchez-Azqueta C, Aldea C, Celma S. Chaos-based bitwise dynamical pseudorandom number generator on fpga. IEEE Trans Instrum Meas 2019;68(1):291–3.

[41] Hortensius P, McLeod R, Pries W, Miller D, Card H. Cellular automata-based pseudorandom number generators for built-in self-test. IEEE Trans Comput-Aided Des Integr Circuits Syst 1989;8(8):842–59.

[42] Gonzalez-Diaz VR, Pareschi F, Setti G, Maloberti F. A pseudorandom number generator based on time-variant recursion of accumulators. IEEE Trans Circuits Syst II 2011;58(9):580–4.

[43] Camp W, Lewis T. Implementing a pseudorandom number generator on a minicomputer. IEEE Trans Softw Eng 1977;SE-3(3):259–62.

[44] Wang C, Pei D. A vlsi design for computing exponentiations in gf(2/sup m/) and its application to generate pseudorandom number sequences. IEEE Trans Comput 1950;39(2):258–62.

[45] Li S, Liu Y, Ren F, Yang Z. Design of a high throughput pseudo-random number generator based on discrete hyper-chaotic system. IEEE Trans Circuits Syst II: Express Briefs 2022;1, (Early Access Article).

[46] Bernardini R, Cortelazzo G. Tools for designing chaotic systems for secure random number generation. IEEE Trans Circuits Syst I 2001;48(5):552–64.

[47] Bernstein G, Lieberman M. Secure random number generation using chaotic circuits. IEEE Trans Circuits Syst 1990;37(9):1157–64.

[48] Karras D, Zorkadis V. Overfitting in multilayer perceptrons as a mechanism for (pseudo)random number generation in the design of secure electronic commerce systems. In: Proceedings of IEEE/AFCEA EUROCOMM 2000. Information systems for enhanced public safety and security. Munich, Germany; 2000.

[49] Hameed MUA, Eldin AMB. A cellular automata random number generator for cryptographic applications. In: Proceedings of international conference on computer engineering & systems. Cairo, Egypt; 2007.

[50] Tian X, Ding R, Wu X, Bai G. Hardware implementation of a cryptographically secure pseudo-random number generators based on koblitz elliptic curves. In: Proceedings of IEEE 3rd international conference on electronics technology. ICET, Chengdu, China; 2020.

[51] Williams B, Hiromoto RE, Carlson A. A design for a cryptographically secure pseudo random number generator. In: Proceedings of 10th IEEE international conference on intelligent data acquisition and advanced computing systems: technology and applications. IDAACS, Metz, France; 2019.

[52] Francillon A, Castelluccia C. Tinyrng: A cryptographic random number generator for wireless sensors network nodes. In: Proceedings of 5th international symposium on modeling and optimization in mobile, Ad hoc and wireless networks and workshops. Limassol, Cyprus; 2007.

[53] McEvoy R, Curran J, Cotter P, Murphy C. Fortuna: Cryptographically secure pseudo-random number generation in software and hardware. In: Proceedings of IET irish signals and systems conference. Dublin, Ireland; 2006.

[54] Lan J, Goh WL, Kong ZH, Yeo KS. A random number generator for low power cryptographic application. In: Proceedings of international soc design conference. Incheon, Korea (South); 2010.

[55] Blum M. Coin flipping by telephone:a protocol for solving impossible problems. ACM SIGACT News 1983;15(1):23–7.

[56] Blum L, Blum M, Shub M. A simple unpredictable pseudo-random number generator. SIAM J Comput 1986;15(2):364–83.

[57] Goldwasser S, Micali S. Probabilistic encryption. J Comput Syst Sci 1982;28(2):270–99.

[58] Gennaro R. An improved pseudo-random generator based on the discrete logarithm problem. J Cryptol 2005;18(1):91–110.

[59] Benssalah M, Djeddou M, Drouiche K. Pseudo-random sequence generator based on random selection of an elliptic curve. In: Proceedings of international conference on computer, information and telecommunication systems. CITS, Gijon, Spain; 2015.

[60] Kono Y, Nogami Y, Kusaka T. Experimental evaluation of the efficiency of associative rational points for random walks on ecdlp. In: Proceedings of 14th international symposium on communications and information technologies. ISCIT, Incheon, Korea (South); 2014.

[61] Reyad O, Karar M, Hamed K. Random bit generator mechanism based on elliptic curves and secure hash function. In: Proceedings of international conference on advances in the emerging computing technologies. AECT, Al Madinah Al Munawwarah, Saudi Arabia; 2019.

[62] Koshiba T, Kurosawa K. Short exponent diffie-hellman problems. In: Proceedings of the 7th workshop on theory and practice in public key cryptography. PKC, Singapore; 2004.

[63] Gennaro R, Krawczyk, Rabin TH. Secure hashed diffie-hellman over non-ddh groups. In: Proceedings of advances in cryptology. EUROCRYPT, Interlaken, Switzerland; 2004.

[64] Matsumoto M, Nishimura T. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans Model Comput Simul 1998;8(1):3–30.

[65] van Oorschot PC, Wiener MJ. On Diffie-Hellman key agreement with short exponents. In: Proceedings of advances in cryptology. EUROCRYPT, Saragossa, Spain; 1996.

[66] Coppersmith D. Fast evaluation of logarithms in fields of characteristic two. IEEE Trans Inform Theory 1984;30(4):587–94.

[67] Thomé E. Computation of discrete logarithms in $\mathbb{F}_{2^{607}}$. In: Proceedings of advances in cryptology. ASIACRYPT, Auckland, New Zealand; 2001.

[68] Aiello W, Rajagopalan SR, Venkatesan R. Design of practical and provably good random number generators. J Algorithms 1998;29(2):358–89.

[69] Aiello W, Rajagopalan S, Venkatesan R. High-speed pseudorandom generation with small memory. In: Proceedings of the 6th international workshop on fast software encryption. FSE, Rome, Italy; 1999.

[70] Kelsey J, Schneier B, Ferguson N. Notes on the design and analysis of the yarrow cryptographic pseudorandom number generator. In: Proceedings of sixth annual workshop on selected areas in cryptography. Kingston, Ontario, Canada; 1999.
[71] Ferguson N, Schneier B. Practical cryptography. 1st ed.. Wiley Publishing, Inc.; 2003.
[72] McEvoy R, Curran J, Cotter P, Murphy C. Fortuna: Cryptographically secure pseudo-random number generation in software and hardware. In: Proceedings of IET irish signals and systems conference. Dublin, Ireland; 2006.
[73] Keller J, Spenger G, Wendzel S. Ant colony-inspired parallel algorithm to improve cryptographic pseudo random number generators. In: Proceedings of IEEE security and privacy workshops. SPW, San Jose, CA, USA; 2017.
[74] Gollmann D. Pseudorandom properties of cascaded connections of clock controlled shift registers. In: Proceedings of advances in cryptology (Eurocrypt). Paris, France; 1985.
[75] Chambers WG, Gollmann D. Lock in effect in cascades of clock controlled shift register. In: Proceedings of advances in cryptology (Eurocrypt). Houthalen, Belgium; 1989.
[76] Nandi S, Krishnaswamy S, Zolfaghari B, Mitra P. Key-dependant feedback configuration matrix of σ-lfsr and resistance to some known plaintext attacks. IEEE Access 2022;10:44 840–54.
[77] Murillo-Escobar MA, Cruz-Hernández C, Abundiz-Pérez F, López-Gutiérrez RM, Acosta Del Campo OR. A RGB image encryption algorithm based on total plain image characteristics and chaos. Signal Process 2015;109:119–31.
[78] Murillo-Escobar D, Ángel Murillo-Escobar M, Cruz-Hernández C, Arellano-Delgado A, López-Gutiérrez RM. Pseudorandom number generator based on novel 2d hénon-sine hyperchaotic map with microcontroller implementation. Nonlinear Dynam 2022;1–17, vol. Published Online.
[79] Nishimura T. Ables of 64-bit mersenne twisters. ACM Trans Model Comput Simul 2000;10(4):348–57.
[80] Stošić BD. Fast random number generation using 128-bit multimedia extension registers on pentium class machines. Comm Statist Simulation Comput 2008;37(2):1–4.
[81] Rukhin A, Soto J, Nechvatal J, Smid M, Barker E, Leigh S, Levenson M, Vangel M, Banks D, Heckert A, Dray J, Vo S. A Statistical test suite for random and pseudorandom number generators for cryptographic applications. Tech. Rep. NIST Special Publication 800-22, U.S: National Institute of Standards and Technology; 2010.
[82] Murillo-Escobar MA, Cruz-Hernández C, Cardoza-Avendaño L, Méndez-Ramírez R. A novel pseudorandom number generator based on pseudorandomly enhanced logistic map. Nonlinear Dynam 2017;87:407–25.
[83] Barani MJ, Ayubi P, Valandar MY, Irani BY. A new pseudo random number generator based on generalized newton complex map with dynamic key. J Inf Secur Appl 2020;53:1–25.