國 立 交 通 大 學

統計學研究所

碩 士 論 文

在 64 及 128 位元電腦上搜尋並評估高階遞迴亂數產生器

Computer Search and Evaluation of Large-Order Multiple Recursive Generators for 64-bit and 128-bit CPUs

研 究 生：施柏成

指導教授：盧鴻興 教授

中 華 民 國 九 十 八 年 六 月

在 64 及 128 位元電腦上搜尋並評估高階遞迴亂數產生器

Computer Search and Evaluation of Large-Order Multiple Recursive Generators for 64-bit and 128-bit CPUs

研 究 生：施柏成　　Student：Bo-Chen Shih

指導教授：盧鴻興　　Advisor：Horng-Shing Lu

國 立 交 通 大 學 理 學 院
統 計 研 究 所
碩 士 論 文

A Thesis
Submitted to Institute of Statistics
College of Science
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master
in
Statistics
June 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年六月

學生：施柏成　　　　　　　　　　　　　　指導教授：盧鴻興　博士

國立交通大學統計學研究所

摘　　要

許多便捷的高階遞迴亂數產生器已經在許多文獻上被發現，但是絕大部分只適用於 32 位元電腦。隨著科技進步，64 位元以上的電腦將會越來越受歡迎，因此搜尋 64 位元以上的亂數產生器是非常必要的。在本篇論文裡，我們將遵循類似搜尋 32 位元的便捷高階遞迴亂數產生器的方法去尋找 64 位元的便捷高階遞迴亂數產生器。並利用 TestU01 這套軟體去測試它們是否能通過許多統計上的嚴格檢驗，結果顯示我們找出的 64 位元便捷高階遞迴亂數產生器絕大部分都痛過了測試並證明它們具有許多良好性質。除此之外，為了方便讀者們使用，我們也設計了一個網頁供讀者們上網使用本篇論文所發現的亂數產生器的參數和它們的程式碼。

Abstract

Several portable and efficient Multiple Recursive Generators (MRGs) have been found in the literature and they are mostly for 32-bit computers. As the 64-bit (and eventually 128-bit) computers becomes more and more popular, there is a need to search for random number generators suitable for these computing platforms. In this thesis, we follow a similar approach taken by the search algorithm for 32-bit generators and we find several 64-bit and 128-bit generators that are efficient with extreme long period lengths. We test their empirical performance with TestU01 packages. The results showed that all of them passed the stringent tests. In addition, Through an extensive computer search, we have found several large order MRGs and we list them in this thesis. In addition to these nice property, all of these generators have a property of equi-distribution over a high dimensional space. Following a similar approach for 32-bit MRGs, we also construct a collection of MRGs from a single MRG found which are useful for parallel simulation of 64-bit or 128-bit applications. For the convenience of interested users, we design a website program to automatically offer the parameters and the associated program codes for our generators.

誌　　謝

本篇論文的完成，除了感謝盧鴻興教授的細心指導外，特別感謝鄧利源教授在研究上給予我許多的指導、建議以及啟發，讓我了解到學習的愉悅，與鄧教授討論研究的時間，是我這輩子在學習過程中最快樂也是最充實的時光。

從前的我從沒想過我能繼續讀到碩士，但一句對著一位女孩說的承諾，使我從一個即將被大學退學的學生在半年內努力考上了交大研究所。在碩士兩年的生涯裡，感謝室友義閔、新樺、銘勳以及庭瑋陪伴我這兩年的碩士生涯，也感謝欽友、孟樵…等 410 研究室的同學陪我渡過許多快樂的時光，也感謝我的好朋友范銘隆給予我許多建議，讓我能適時的改正自己求學的態度和缺點，還有感謝我的女朋友張孟婷陪伴我度過碩士最忙碌的時刻，並在我累時適時的給予我鼓勵，最後感謝我的父母以及哥哥對我的支持，讓我在求學生涯裡能夠無憂無慮。

最後，我要對那女孩說，三年前的那句承諾：我會一年比一年更好，我這輩子永遠都不會忘記並且實現它。

柏成 於交通大學統計學研究所

中華民國九十八年六月

Contents

1 Introduction **1**

1.1 LCG and MRG 1

1.2 Some 64-bit random number generators 2

2 Computer Search of MRGs of Large Order **4**

2.1 Efficient Search Algorithm for Large Order MRGs 4

2.2 Prime modulus for MRGs with CPUs larger than 32 bits 5

3 Efficient and Portable MRGs **6**

3.1 DX-k-s generators 6

3.2 DL-k generators 6

3.3 DS-k generators 7

3.4 DT-k generators 7

3.5 Comparing DX, DL, DS and DT generators 8

4 Tables of Efficient MRGs **10**

4.1 63-bit and 64-bit efficient MRGs 10

4.2 127-bit and 128-bit efficient MRGs 11

4.3 Selecting MRGs 11

5 Program Implementation and Empirical Evaluations **12**

5.1 Initialization and Generation 12

5.2 Empirical Evaluation 12

5.3 Comparison with other 64-bit generators 13

6 Parallel MRGs **14**

6.1 Need for parallel generators 14

6.2 Common method to design parallel MRGs 14

6.3 Constructing MRGs with different multipliers 14

7 Web Programming for MRGs and Parallel MRGs **17**

7.1 Retrieving parameters for the maximum period MRGs 17

7.2 Retrieving programs for parallel MRGs 20

7.3 Retrieving TestU01 code for the maximum period MRGs 22

8 Conclusion **23**

List of Tables

1 List of k and c for non-Sophine-Germain vs. Sophine-Germain prime, where $p = 2^d - c$ 26

2 List of B for DX-k-1 generator with $p = 2^{63} - c$ 27

3 List of B for DX-k-2 generator with $p = 2^{63} - c$ 28

4 List of B for DX-k-3 generator with $p = 2^{63} - c$ 29

5 List of B for DX-k-4 generator with $p = 2^{63} - c$ 30

6 List of B for DL-k generator with $p = 2^{63} - c$ 31

7 List of B for DS-k generator with $p = 2^{63} - c$ 32

8 List of B for DT-k generator with $p = 2^{63} - c$ 33

9 List of B for DX-k-1 generator with $p = 2^{64} - c$ 34

10 List of B for DX-k-2 generator with $p = 2^{64} - c$ 35

11 List of B for DX-k-3 generator with $p = 2^{64} - c$ 36

12 List of B for DX-k-4 generator with $p = 2^{64} - c$ 37

13 List of B for DL-k generator with $p = 2^{64} - c$ 38

14 List of B for DS-k generator with $p = 2^{64} - c$ 39

15 List of B for DT-k generator with $p = 2^{64} - c$ 40

16 List of B for DX-k-1 generator with $p = 2^{127} - c$, where x = 92233720368547 . . 41

17 List of B for DX-k-2 generator with $p = 2^{127} - c$, where x = 92233720368547 . . 42

18 List of B for DX-k-3 generator with $p = 2^{127} - c$, where x = 92233720368547 . . 43

19 List of B for DX-k-4 generator with $p = 2^{127} - c$, where x = 92233720368547 . . 44

20 List of B for DL-k generator with $p = 2^{127} - c$, where x = 92233720368547 45

21 List of B for DS-k generator with $p = 2^{127} - c$, where x = 92233720368547 46

22 List of B for DT-k generator with $p = 2^{127} - c$ 47

23 List of B for DX-k-1 generator with $p = 2^{128} - c$, where x = 184467440737095 . . 48

24 List of B for DX-k-2 generator with $p = 2^{128} - c$, where x = 184467440737095 . . 49

25 List of B for DX-k-3 generator with $p = 2^{128} - c$, where x = 184467440737095 . . 50

26 List of B for DX-k-4 generator with $p = 2^{128} - c$, where x = 184467440737095 . . 51

27 List of B for DL-k generator with $p = 2^{128} - c$, where x = 184467440737095 . . . 52

28 List of B for DS-k generator with $p = 2^{128} - c$, where x = 184467440737095 . . . 53

29 List of B for DT-k generator with $p = 2^{128} - c$ 54

30 Results of Crush test module on DL(63), DL(64), DL(127) and DL(128) generators 55

31 Results of Crush test module on DS(63), DS(64), DS(127) and DS(128) generators 56

32 Results of Crush test module on DT(63), DT(64), DT(127) and DT(128) generators 57

1 Introduction

Computer simulation has become a necessary part in most of the scientific studies. The quality of such computer simulation depends heavily on the quality of the random number generators used. Most of large-scale scientific research studies demand the generation of large quantity of random numbers which should have the property of extreme long period length and great empirical performances. Therefore, it is important to find good random number generators for various scientific studies such as Monte Carlo simulation and computer modeling.

1.1 LCG and MRG

Linear Congruential Generator (LCG) proposed by Lehmer [1951] is the best-known uniform random number generator. It is defined recursively as:

$$X_i = BX_{i-1} \bmod p\ ,\ i \geq 1, \quad U_i = X_i/p, \tag{1}$$

where the multiplier B and the prime modulus p are some positive integers, and X_0 is any non-zero initial seed. The maximum period length of an LCG is $p-1$ if the multiplier B is suitably chosen. Such B is called a *primitive root* of the set $\mathbb{Z}_p = \{\ 0, 1, 2, \cdots, p-1\ \}$ for a prime p. An integer B is a primitive root modulo for a prime number p, if and only if $B^{(p-1)/q} \neq 1 \bmod p$ for any prime factor q of $p-1$. The most well-known LCG for 32-bit computers is the prime modulus $p = 2^{31}-1$ and the multiplier $B = 16807 = 7^5$. LCG has been a popular random number generator because of its simplicity and generating efficiency. However, LCG has a short period length (by today's standard) and it has a questionable empirical performance which make it unreliable for a large scale simulation study. Next, we describe some popular generating algorithms proposed recently to replace LCG.

Multiple recursive Generator (MRG) is an extension of Linear Congruential Generator and has been used widely in Random number generators. It is based on the k-th order linear recurrence

$$X_i = (\alpha_1 X_{i-1} + \cdots + \alpha_k X_{i-k}) \bmod p,\ i \geq k \tag{2}$$

where the multipliers $\alpha_1, \alpha_2, \cdots, \alpha_k$ and prime modulus p are positive integers. $X_0, X_1, \cdots, X_{k-1}$ are k initial seeds which not all are zeros. For Multiple recursive Generator in (2), the corresponding characteristic polynomial is the form :

$$f(x) = x^k - \alpha_1 x^{k-1} - \cdots - \alpha_{k-1} x - \alpha_k, \tag{3}$$

If $f(x)$ is a primitive polynomial modulo p, then the maximum period length of MRG of order k is $p^k - 1$. $f(x)$ is a k-th degree primitive polynomial if there is a solution for which $f(x) = 0$

has a primitive root over the finite field of p^k elements. One can check check whether $f(x)$ is a primitive polynomial using the conditions given below:

1. $f(x)$ in (3) is irreducible of degree $k > 1$,

2. $f(x)$ is not a divisor of $x^m - 1$ for any $m < p^k - 1$.

We will discuss the issue of checking a k-th degree primitive polynomial later.

A maximum-period MRG of order k enjoys a nice equi-distribution property up to k dimensions: every t-tuple ($t \leq k$) of integers between 0 and $p-1$ appears exactly the same number of times over the entire period $p^k - 1$ with the exception that the all-zero tuple appears one time less. See, for example, Lidl and Niederreiter [1994].

Several efficient and portable large order MRGs for 32-bit CPUs have been found recently in the literature. They have been shown to have excellent empirical performance and high efficiency for the generating speed. With the progress of science and technology, we believe that 64-bit (or larger) CPUs will become more and more popular in the near future. Hence, it is necessary to find good random number generators for 64-bit CPUs.

1.2 Some 64-bit random number generators

One straightforward way to construct a 64-bit generator is to consider a 64-bit LCG by choosing a prime modulus $p = 2^{63} - c$ or $p = 2^{64} - c$ and a appropriate multiplier B to achieve a maximum period length of $p - 1$. For example, L'Ecuyer [1993] found some good parameters for 64-bit LCGs using the spectral test and Beyer ratios as selection criteria:

1. $B = 2307085864,\ p = 2^{63} - 25$,

2. $B = 13891176665706064842,\ p = 2^{64} - 59$.

It is well-known that LCGs have poor high dimension lattice structure. Marsaglia [1968] was the first to point out that successive t-tuple in the output sequence by an LCG with the modulus m lies in a simple lattice structure which will fall on at most $(t!m)^{1/t}$ hyperplanes.

In addition, Hormann [1994] suggested that it is not recommended to use LCG as generate random numbers by the ratio of uniforms method, proposed by Kinderman and Monahan [1997]. For more discussion, see Gentle [2003].

To the best of our knowledge, there are only few other 64-bit random number generators proposed in the literature. Nishimura [2000] proposed a 64-bit generator which is an extension of 32-bit MT19937. Several new parameters are also listed. The period length is still $2^{19937} - 1 \approx$

10^{6001}, but the dimension of equi-distribution property is decreased from 623 (for 32-bit) to 311 (for 64-bit).

L'Ecuyer [1997] proposed a popular generator for 64-bit CPUs, called $MRG63k3a$, by combing the following two generators:

$$X_i = 1754669720 X_{i-2} - 3182104042 X_{i-3} \bmod (2^{63} - 6645), \tag{4}$$

$$Y_i = 31367477935 Y_{i-1} - 6199136374 Y_{i-3} \bmod (2^{63} - 21129), \tag{5}$$

$$Z_i = (X_i - Y_i) \bmod (2^{63} - 6645). \tag{6}$$

The corresponding uniform (0,1) generator is

$$U_i = Z_i^*/(m_1 + 1), \quad m_1 = 2^{63} - 6645, \tag{7}$$

where $Z_i^* = Z_i$, if $Z_i \geq 0$ and $Z_i^* = m_1$, if $Z_i = 0$. The period length is about $10^{113.5}$.

In this thesis, we perform a computer search for 64-bit or 128-bit MRGs and we then compare these large order efficient MRGs with previously proposed 64-bit random number generators.

In chapter 2, we discuss some key issues for the search of large order MRGs. In particular, we describe the efficient algorithm proposed by Deng [2004] to avoid two search bottlenecks in the classical search algorithm. In chapter 3, we then describe a series of efficient and portable Multiple Recursive Generators (MRGs) proposed recently by Deng and his co-authors. We then describe some criteria of selecting these MRGs. In chapter 4, we extend the computer search of Multiple Recursive Generators (MRGs) from 32-bit generators to 64-bit and 128-bit generators and we tabulate them in several tables. In chapter 5, we utilize TestU01 packages to evaluate the empirical performance of these generators found. We also compare them with other 64-bit generators proposed in the literature. The empirical test results show that our generators are great choices for 64-bit and 128-bit CPUs. In chapter 6, we discuss the issue of performing parallel simulation using these newly found MRGs as the backbone generators. In particular, we first describe some automatic generation method (mostly for 32-bit generators) proposed in Deng and Xu [2003], Deng, Li and Shiau [2009] and Deng, Shiau, and Tsai [2009]. We then perform a similar steps for the parallel simulation for 64-bit or 128-bit CPUs. In chapter 7, we introduce our web program which can provide the required parameters or the associated program codes for our generators. Interested users can obtain the desired generator in C language directly by downloading the codes produced from our web program. In addition, they can implement the MRGs found in different programming by getting the required parameters provided in our web program. For applications to run simulations in parallel on several processors, our web can provide users to obtain many generators simultaneously.

2 Computer Search of MRGs of Large Order

As mentioned earlier, the MRG for the corresponding primitive characteristic k-th degree polynomial will achieve the maximum period of $p^k - 1$. As the order k becomes larger and larger, the work of computer checking of k-th degree primitive polynomial becomes harder and harder. Next, we describe the efficient algorithm proposed by Deng [2004] for checking k-th degree primitive polynomial.

2.1 Efficient Search Algorithm for Large Order MRGs

A set of necessary and sufficient conditions under which $f(x)$ in (3) is a primitive polynomial has been given in Alanen and Knuth [1964] and Knuth [1998]:

AK(i) $(-1)^{k-1}\alpha_k$ must be a primitive root mod p.

AK(ii) $x^R = (-1)^{k-1}\alpha_k \bmod (f(x), p)$, where $R = (p^k - 1)/(p - 1)$.

AK(iii) For each prime factor q of R, the degree of $x^{R/q} \bmod (f(x), p)$ is positive.

However, it is difficult to check the conditions in practice, especially when the values of k and p are large. Alternatively, Deng [2004] proposed an efficient algorithm that bypasses the difficulty of factoring a large number and provided an early exit strategy for a failed search to achieve a better efficiency:

Algorithm GMP Given a prime order k, choose a prime modulus p such that $R(k,p) = (p^k - 1)/(p - 1)$ is also a prime number. Let $f(x)$ be as in (3).

(i) α_k must be a primitive element mod p. If this condition is met, then go to the next step.

(ii) Initially, let $g(x) = x$. For $i = 1, 2, 3, \cdots, \lfloor k/2 \rfloor$, do

1. $g(x) = g(x)^p \bmod f(x)$;
2. $d(x) = \gcd(f(x), g(x) - x)$;
3. if $d(x) \neq 1$, then $f(x)$ cannot be a primitive polynomial.

If all the loops in Step (ii) have been passed, then $f(x)$ is a primitive polynomial.

Hence, we choose the smallest prime order k for each interval from 101 up to 2003. For each value of k and d, we find the smallest c for a prime $p = 2^d - c$ such that $R(k,p) = (p^k - 1)/(p - 1)$ is a probable-prime number. We recommend utilizing the free packages of PFGW (http://www.fermatsearch.org/index.html) to verify the primality of $R(k,p)$. PFGW provids a quick probable-prime test for large numbers. The probability of making false positive error can be smaller than 10^{-200} with several independent probabilistic tests. It is much smaller than a computer error or hardware error. Hence, it can be safely accepted as a prime number. Even if $R(k, p)$ is not a prime, L'Ecuyer [1997] showed that we only have a tiny chance(say, 10^{-50} or less) of misclassifying a non-primitive polynomial. Hence, throughout this paper, we are following this procedure to find prime modulus p as described here.

In addition, we require that both p and $Q \equiv (p-1)/2$ are prime numbers. If this condition is satisfied, such Q is called Sophine-Germain prime number and p is usually called a "safe prime" in the area of cryptography. But there is no particular strong advantage to choose a "safe prime" in the area of computer simulation. MRGs with a non-Sophine-Germain prime have a advantage of a larger prime modulus than its counterpart with a Sophine-Germain prime. Hence, it has a slightly longer period length of $(p^k - 1)$.

A series of efficient and portable MRGs were proposed by Deng and Xu [2003], Deng [2004] and Deng [2005]. Particularly, the maximum period length of MRG, proposed in Deng [2005], is approximately 10^{14903}.

Next, we describe the key issues for the computer search of efficient large order MRGs for 64-bit or 128-bit CPUs.

2.2 Prime modulus for MRGs with CPUs larger than 32 bits

As the computer's architectures of CPUs are moving from 32 bits to 64 bits(or beyond), 64-bit computing will become the mainstream in the future. Hence, developing good random number generators for 64-bit and 128-bit computer system is imperative.

It is straightforward to apply the search algorithm for large order MRGs with prime modulus suitably chosen. It should be of the size can be stored in a computer word. Specifically, we select a prime modulus, p, that is slightly smaller than 2^d, where the choice of d depends on the word size of a CPU. Generally speaking, we choose $d = e - 1$ for a signed integer and $d = e$ for an unsigned integer in a computer word of size e. In this thesis, we consider two computer word sizes: $e = 64$ and $e = 128$. Specially, we choose the prime modulus p to be of the form $p = 2^{63} - c$ and $2^{64} - c$ for 64-bit CPUs; $p = 2^{127} - c$ and $p = 2^{128} - c$ for 128-bit CPUs.

Table 1 lists non-Sophine-Germain prime p and Sophine-Germain prime p which satisfy the condition that $(p^k - 1)/(p - 1)$ is a prime for the order k from 101 to 2003.

3 Efficient and Portable MRGs

When k is large, a general MRG may be less efficient because it needs several multiplications whereas an LCG needs only one multiplication. To improve the efficiency of MRGs, many authors considered only two nonzero coefficients α_j and α_k $(1 \leq j < k)$ of the MRG in (2). For example, see, L'Ecuyer and Blouin [1988], L'Ecuyer, Blouin and Couture [1993].

Deng and Lin [2000] proposed a Fast MRG (FMRG) which has a slightly simpler form and requires a single multiplication. Extending the idea of FMRG, Deng and Xu [2003] introduced a class of DX generators which is a special MRG in (2) that has s nonzero coefficients, all of them being equal, and the nonzero coefficients indices are about $k/(s-1)$ apart. We give more detailed discussion next.

3.1 DX-k-s generators

Deng and Xu [2003] and Deng [2005] proposed DX generators as a system of portable, efficient and maximal period MRGs where coefficients of the nonzero multipliers are the same:

1. DX-k-1[FMRG]$(\alpha_1 = 1, \alpha_k = B)$
$$X_i = X_{i-1} + BX_{i-k} \bmod p, \quad i \geq k, \tag{8}$$
2. DX-k-2$(\alpha_1 = \alpha_k = B)$
$$X_i = B(X_{i-1} + X_{i-k}) \bmod p, \quad i \geq k, \tag{9}$$
3. DX-k-3$(\alpha_1 = \alpha_{\lceil k/2 \rceil)} = \alpha_k = B)$
$$X_i = B(X_{i-1} + X_{i-\lceil k/2 \rceil} + X_{i-k}) \bmod p, \quad i \geq k, \tag{10}$$
4. DX-k-4$(\alpha_1 = \alpha_{\lceil k/3 \rceil)} = \alpha_{\lceil 2k/3 \rceil)} = \alpha_k = B)$
$$X_i = B(X_{i-1} + X_{i-\lceil k/3 \rceil} + X_{i-\lceil 2k/3 \rceil} + X_{i-k}) \bmod p, \quad i \geq k, \tag{11}$$

where $X_0, X_1, \cdots, X_{k-1}$ are initial seeds. Here the notation $\lceil x \rceil$ is the ceiling function of a number x, which return the smallest integer $\geq x$. For the class DX-k-s, s is the number of terms with nonzero coefficient B.

3.2 DL-k generators

According to L'Ecuyer [1997], a necessary (but not sufficient) condition for a "good" MRG is the sums of coefficients, $\sum_{i=1}^{k} \alpha_i^2$, should be large. This indicates that we should search MRGs with many nonzero terms as possible and retain efficiency and portability. Deng and Li [2005] and Deng, Li, Shiau, and Tsai [2008] considered a DL-k generator, with $\alpha_i = B$ for $i = 1, 2, \cdots, k$, as:

$$X_i = B(X_{i-1} + X_{i-2} + \cdots + X_{i-k}) \bmod p, \quad i \geq k, t \geq 1. \tag{12}$$

By using high-order recurrence, DL generator can be simplified and implemented efficiently as:

$$X_i = X_{i-1} + B(X_{i-1} - X_{i-(k+1)}) \bmod p, \quad i \geq k+1. \tag{13}$$

where $X_0, X_1, \cdots, X_{k-1}$ are initial seeds and X_k is computed by (12).

3.3 DS-k generators

Deng, Li, Shiau, and Tsai [2008] also considered another class of generators with many nonzero coefficients, called DS generators. It defined as :

$$X_i = B \sum_{j=1, j\neq d}^{k} X_{i-j} \bmod p. \tag{14}$$

which can be efficiently implemented by

$$X_i = X_{i-1} + B(X_{i-1} - X_{i-d} + X_{i-d-1} - X_{i-k-1}) \bmod p, \quad i \geq k+1. \tag{15}$$

where $X_0, X_1, \cdots, X_{k-1}$ are initial seeds and X_k is computed by (14). The parameter d of the zero-coefficient index can be chosen arbitrarily. For simplicity, we refer the case of $d = \lceil k/2 \rceil$ as the DS-k generators.

The main motivation behind DS generators is to further improve the lattice structure of the DL generators over a space of dimension beyond k. Comparing (13) and (15), we can see the high-order implementation of DS-k generators has a more complex recurrence than DL-k generators. Therefore, DS-k generators may have a better lattice structure for dimensions than the described generators previously.

3.4 DT-k generators

DX, DL and DS generators all have a zero-state problem. Specifically, when the k-dimensional state vector for the recurrence is close to the zero vector, the subsequent numbers generated may stay within a neighborhood of zero for quite many of them before they can break away from the near-zero land. Hence, Deng, Shiau, and Tsai [2009] considered a new class of MRGs, called DT generators, which has many non-zero terms with unequal weights on each term such that it can avoid above situation:

$$X_i = B^k X_{i-1} + B^{k-1} X_{i-1} + \cdots + B X_{i-k} \bmod p, \quad i \geq k. \tag{16}$$

where $X_0, X_1, \cdots, X_{k-1}$ are initial seeds and X_k is computed by (16). Like DL and DS generators, DT generators can be efficiently implemented as:

$$X_i = ((B^{-1} + B^k) X_{i-1} - X_{i-k-1}) \bmod p, \quad i \geq k+1. \tag{17}$$

where $D \equiv (B^{-1} + B^k) \bmod p$ can be pre-computed.

To design portable MRGs, a common method is to impose a limit on the multiplier B for the generators. General speaking, the larger B is better for the generators but in fact, it is difficult to possess a portable and efficient implementation when B is large. In this thesis, we choose $B \leq 2^b$ for DX(d), DL(d) and DS(d) generators where $b = \lfloor d/2 \rfloor$. Here, $\lfloor x \rfloor$ means the floor function of x which returns the largest integer $\leq x$. This is a common technique to ensure that the integer operation would not beyond the bit's limit of the computer when $B < \sqrt{p}$. See more details in L'Ecuyer [1988]. And we also choose the smallest B for DX(d), DL(d), DS(d) and DT(d) generators. Although the generators with small parameter B are not recommended to use generally. But they can be useful for easier generation of parallel MRGs, see Deng, Shiau and Tsai [2009].

3.5 Comparing DX, DL, DS and DT generators

As mentioned, all maximum period MRGs of order k have a nice equi-distribution property up to k dimensions. Namely, every t-tuple ($1 \leq t \leq k$) of integers between 0 and $p-1$ appears the same number of times (p^{k-t}) over its entire period $p^k - 1$, with the exception of the all-zero tuple which appears one times less ($p^{k-t} - 1$). See, for example, Lidl and Niederreiter [1994, Theorem 7.43]. Therefore, DX, DL, DS and DT have the property of equi-distribution over dimensions up to k. In addition, all of the proposed generators are shown to pass some stringent empirical tests. The differences may be shown over the dimension larger than k like the spectral test which measures the minimum distance between two successive parallel hyperplanes over a dimension $> k$. However, to the best of our knowledge, it is computationally hard to perform such spectral test on a large dimension. Since the equi-distribution property of maximal period MRGs of order k as mentioned earlier is clearly a stronger condition than the spectral test, it is important to find MRGs with large order k.

The main motivation for proposing DX generator is the computing efficiency. It was achieved by setting lot of terms in the recurrence equation (2) to be zero. This leads to a potential program of "escaping from near-zero state". For example, if k-dimensional state vector is of the form

$(0, 0, \cdots, 0, 0, v, 0)'$, for any integer v, then the DX generator will produce a long sequence of zeros. DL and DS generators were proposed in Deng, Li, Shiau and Tsai [2008] with many non-zero terms with the same coefficients. One advantage is that they can escape quickly from such a near-zero k-dimensional state vector. However, DL and DS generators can stay with near-zero state for a long time (when k is large) with a k-dimensional state vector of the form $(0, 0, \cdots, 0, -v, v)'$, for any integer v.

The DT generators in (16) can escape quickly from near zero state like $(0, 0, \cdots, 0, 0, v, 0)'$ or $(0, 0, \cdots, 0, -v, v)'$. However, with given multiplier B for the DT generator, it could also stay in near-zero states for a long time, when k is large and the state vector is of the form $(0, \cdots, 0, -v, Bv)'$. But this most likely would only happen when one purposely chooses an initial state vector of the above form with the pre-specified multiplier B.

In our opinion, no single generator should be used for every computer simulation. One should try various types of generators like DX, DL, DS and DT generators and with various order k to ensure a consistent simulation result.

4 Tables of Efficient MRGs

With tables of non-Sophine-Germain primes and Sophine-Germain primes given, we can then search for two types of parameters for efficient and portable MRGs with different orders k from 101 to 2003.

Following the notation in Deng, Lu and Chen [2009], we will use DX(d), DL(d), DS(d) and DT(d) to denote the corresponding generators with $p = 2^d - c$ for some c. Specifically, for generators for 64-bit CPUs, we consider the prime modulus $p = 2^{63} - c$ or $p = 2^{64} - c$. For 128-bit CPUs, we consider the generators with prime modulus $p = 2^{127} - c$ or $p = 2^{128} - c$.

4.1 63-bit and 64-bit efficient MRGs

For 63-bit efficient and portable MRGs, we first search Sophine-Germain prime modulus. From $p = 2^{63} - 1$ downward, we choose the maximum prime modulus p that satisfies both $R(k, p) = (p^k - 1)/(p - 1)$ and $(p - 1)/2$ are prime numbers where the order k is from 101 to 2003. Once k and p have been selected, we then use the GMP algorithm to find the multipliers, minimum B and $B < \sqrt{p}$. For the minimum multiplier B, we search from low bound $B = 2$ upward. And then we search the multiplier $B < \sqrt{p}$ from the upper bound $B = 2^{31} - 1$ downward. Latter, we search Non-Sophine-Germain prime modulus. Similar to Sophine-Germain primes, we use the same method to find the multipliers, minimum B and $B < \sqrt{p}$. Deng had found the multiplier $B < \sqrt{p}$ of Sophine-Germain primes for DX(63), DX(64), DX(127) and DX(128) from order k=101 up to 1511. We continue to search the multipliers, minimum B and $B < \sqrt{p}$, for non-Sophine-Germain primes and Sophine-Germain primes from the order $k = 101$ to 2003 for DL, DS and DT generators. In addition, we also search the multipliers, minimum B and $B < \sqrt{p}$, for Sophine-Germain primes from the order $k = 1601$ to 2003 and for non-Sophine-Germain primes from the order $k = 101$ to 2003 for DX generators. The period lengths of searched generators ranges from 10^{1915} to 10^{37987}. We list these generators in Table 2-8.

Similarly, for 64-bit efficient and portable MRGs, we first search the maximum Sophine-Germain prime modulus of the form $p = 2^{64} - c$ for each prime order k. For each k and p selected, we then search the multipliers, minimum B and $B < \sqrt{p}$ (more precisely, $B < 2^{32}$). Following the same procedure, we search the same parameters p and B for non-Sophine-Germain prime modulus. Finally, We find 320, 80, 80 and 40 generators for DX(64)-k-s, DL(64)-k, DS(64)-k and DT(64)-k respectively. The period lengths ranges from 10^{1946} to 10^{38590}. We list these generators in Table 9-15.

4.2 127-bit and 128-bit efficient MRGs

Like 64-bit generators, we use the same procedure to search for 128-bit generators with prime modulus of the form $p = 2^{127} - c$ and $p = 2^{128} - c$. We find 320, 80, 80 and 40 generators for DX(127)-k-s, DL(127)-k, DS(127)-k and DT(127)-k respectively. The period lengths ranges from 10^{3861} to 10^{76576}. We list these generators in Table 16-22. And we find 320, 80, 80 and 40 generators for DX(128)-k-s, DL(128)-k, DS(128)-k and DT(128)-k respectively. The period lengths ranges from 10^{3892} to 10^{77179}. We list these generators in Table 23-29.

Among the generators listed in table 2-29, the shortest period length is close to 10^{1915} and longest period length is close to 10^{77179}. It is worth mentioning that the 128-bit efficient and portable MRGs with order $k = 2003$ have the longest period length of 10^{77179} and the property of equi-distribution over their dimensions is up to 2003.

4.3 Selecting MRGs

Generally speaking, one can improve the MRGs with better theoretical and empirical property by increasing the prime modulus p which is of size 2^d, recurrence order k, and number of non-zero terms.

Clearly, increasing the order k will enlarge the period length of MRGs and the dimensions of equi-distribution. However, it will increase proportionally the memory requirement to store the k value of the current state.

According to L'Ecuyer [1997], a necessary (but not sufficient) condition for a "good" MRG (better lattice structure over dimension larger than k) is that the sums of squares of coefficients, $\sum_{i=1}^{k} \alpha_i^2$, should be large. Deng and George [1990] and Deng, Lin, Wang and Yuan [1997] also recommended that good MRGs should posses many terms with large coefficients from a statistical viewpoint. Hence, it indicates that we should increase the non-zero terms and the multiplier B as much as possible. Finally, increasing the prime modulus p of size 2^d can increase the density of possible generated points over any small interval. For example, the density in the 64-bit of efficient and portable MRGs is 10^9 times larger than the density in 32-bit of efficient and portable MRGs. Therefore , it is preferred to have larger value of d.

In summary, when we increase the values of d, k and non-zero terms for MRGs, we expect to improve the generators' performance. However, it may have some drawbacks such as increasing the difficulty of portable implementation, increasing the memory requirement, or decreasing the generating efficiency.

5 Program Implementation and Empirical Evaluations

While 64-bit CPUs will become more and more popular in the near future, 128-bit CPUs are not to be popular any time soon. Both 64-bit and 128-bit CPUs are not the mainstream today. However, we can still utilize multi-precision software packages (GMP or NTL) to implement these efficient MRGs for any 64-bit or 128-bit generators found at the expense of a slower generating speed. Generally speaking, the speed in generating times deeply depends on the actual hardware available and the software used.

5.1 Initialization and Generation

As explained by Deng and Xu [2003], any k initial values can be selected as a starting initial seeds for any efficient and portable MRGs, except all zeros. For convenience, we utilize LCG with the same multiplier B of efficient and portable MRGs to generate the initial seeds. According to Matsumoto, Wada, Kuramoto, and Ashihara [2007], "initialing the MRG with an LCG is a bad idea, because the structure of LCG may easily show up in some initial segment of the output." Hence, one can also use a lower order of MRG in place of LCG to generate initial seeds. But in our opinion, a good RNG should not depend on a particular choice of initial seeds. In fact, efficient and portable MRGs have passed extensive tests in TestU01 even with "bad" initial seeds.

We can implement 63-bit or 64-bit efficient and portable MRGs for 64-bit CPUs directly. But if we want to implement 64-bit or 128-bit generators for 32-bit CPUs, we should utilize the multi-precision packages such as NTL (http://www.shoup.net/ntl/) or GMP (http://gmplib.org/) in C or C++.

5.2 Empirical Evaluation

TestU01 is a software library, implemented in C language, and offering a collection of utilities for the empirical statistical testing of uniform random number generators. It was developed by L'Ecuyer with the source code. Since TestU01 is the most stringent and comprehensive test suite by far, we utilize it with the general multi-precision package of GMP (http://gmplib.org/) to evaluate the empirical performance in this thesis. One can download this package from http://www.iro.umontreal.ca/˜simardr/testu01/tu01.html. TestU01 has three predefined test modules. The *SmallCrush* suite contains 15 tests, so it yields 15 p-values. If the tests in *SmallCrush* have been passed, we continue utilize *Crush* tests which contain 144 tests.

DX(63), DX(64), DX(127) and DX(128) generators have been tested in Deng, Lu and Chen [2009]. We continue the testing on the other type of efficient and portable MRGs. There are 80

(DL and DS) and 40 (DT) generators for efficient and portable MRGs in Table 6-29. For each generator, we use *Crush* tests to evaluate their empirical performance. From $k = 101$ to 2003, we choose multipliers $B < \sqrt{p}$ for Sophine-Gemain primes for DL and DS generators with two different starting seeds of 123 and 12345 to apply *Crush* test. In addition, we also choose the minimum B for Sophine-Gemain primes for DT generators with two different starting seeds of 123 and 12345 to apply *Crush* test. Therefore, we get 5760 ($= 144 \times 20 \times 2$) p-values in Table 6-29. The number of tests with p-value outside the range [0.001,0.999] are tabulated in Table 30-32.

From Tables 30-32, we can see that DL, DS and DT all have excellent empirical performances. None of their p-values are very close to 0 or 1. Specifically, there is no efficient and portable MRG found to obviously fail by the TestU01 battery of tests.

5.3 Comparison with other 64-bit generators

We take MT19937 and $MRG63k3a$ to compare with our efficient and portable MRGs. Since MT19937 and $MRG63k3a$ are available in 64-bit CPUs, we use 64-bit efficient and portable MRGs to compare with them. In terms of period length, the longest period for efficient and portable MRGs of 64 bit is 10^{38590} which is far more than MT19937($2^{19937} - 1 \approx 10^{6001}$) and $MRG63k3a$ ($10^{113.5}$). In terms of the property of equi-distribution, efficient and portable MRGs for $k = 2003$ have the property of equi-distribution over dimensions up to 2003 which is also far more than MT19937(311 with 64-bit words). In contrast, $MRG63k3a$ has a (relatively short) period of $10^{113.5}$, so it has “reasonable” (but not exact) equi-distribution property over (relatively) low dimensional space.

6 Parallel MRGs

6.1 Need for parallel generators

To speed up the simulation process, we need a "good" systematic method to construct and parallelize the baseline random number generators so that they can run simultaneously on several computers or processors. The resulting parallel random number generator (PRNG), in theory, should have good theoretical properties and great empirical performances, from the perspectives of both "within" and "between" generators.

6.2 Common method to design parallel MRGs

A fairly common strategy to generate different sequences for different processors is using a skip-ahead scheme on the same RNG. However, for a large order MRG, skip-ahead scheme is slow. Recently, Deng, Li and Shiau [2009] proposed a class of PRNGs using the DX-k generators proposed by Deng and Xu [2003] as the baseline generators.

6.3 Constructing MRGs with different multipliers

If $f(x)$ is an irreducible polynomial, then $c^{-k}f(cx)$ and $x^k f(c/x)$ are also irreducible polynomials for any non-zero constant c. Using this fact, Deng [2004] gave the following theorem:

Theorem 1 *Let $R(k,p) = (p^k - 1)/(p-1)$ be a GMP and c be any non-zero intenger. Let $f(x)$ in (3) be a primitive polynomial and define*

$$G(x) = c^{-k}f(cx) = x^k - G_1x^{k-1} - G_2x^{k-2} - \cdots - G_k \ mod\ p, \tag{18}$$

$$H(x) = -\alpha_k^{-1}x^k f(c/x) = x^k - H_1x^{k-1} - H_2x^{k-2} - \cdots - H_k \ mod\ p, \tag{19}$$

where $G_j = c^{-j}\alpha_j$ mod p and $H_j = -\alpha_k^{-1}\alpha_{k-j}c^j$ mod p, for $j = 1, 2, \cdots, k$, $\alpha_0 = -1$. If the constant term G_k $(= c^{-k}\alpha_k)$ is a primitive element modulo p, then both $G(x)$ and $H(x)$ are k-th degree primitive polynomials.

There are two advantages in considering $G(x)$ and $H(x)$:

1. They can be calculated very efficiently from the polynomial $f(x)$.

2. Both of them have exactly the same number of nonzero terms as that in $f(x)$.

Hence, if $f(x)$ is a characteristic polynomial of a MRG which is implemented efficiently, then the induced generator with characteristic polynomial $G(x)$ or $H(x)$ can be also implemented efficiently.

Based on Theorem 1, many maximum-period MRGs (corresponding to $G(x)$ or $H(x)$) can be constructed quickly from a single maximum-period MRG (corresponding to $f(x)$). But there are some defects here:

1. We need to check the primitive-root condition of the constant term (G_k or H_k)

2. It appears that the sequential selection of c should be randomized.

3. The selection of c causes no guarantee for the generated MRGs to be distinct.

Next, we describe an automatically generating algorithm proposed by Deng, Li and Shiau [2009] to overcome above drawbacks.

Algorithm AGM Let $R(k,p) = (p^k - 1)/(p-1)$ be a GMP and $f(x)$ in (3) be a primitive polynomial corresponding to a efficient and portable MRG in which the nonzero coefficient is B as in equations (8), (9), (10), (11), (12), (14) and (16). Let R be an integer with $\gcd(R, p-1) = 1$. The following procedure will randomly generate a sequence of maximum-period MRGs.

1. Whenever a new processor is initiated, generate r_n by the following equation:

$$r_n = Rr_{n-1} \bmod (p-1),\ n \geq 1\ with\ r_0 = 1. \tag{20}$$

2. Calculate d_n and then c_n for the new processor by

$$d_n = k^{-1}(r_n + 1)\ mod\ (p-1)\ and\ c_n = B^{d_n}\ mod\ p \tag{21}$$

3. With the given c_n, compute the primitive polynomial G(x) or H(x) by

$$G(x) = c_n^{-k} f(c_n x)\ mod\ p, \tag{22}$$

$$H(x) = -B^{-1} x^k f(c_n/x)\ mod\ p. \tag{23}$$

4. The new processor can use the newly constructed maximum-period MRG of order k corresponding to the characteristic polynomial G(x) or H(x) as follows:

$$X_i = G_1 X_{i-1} + \cdots + G_k X_{i-k}\ mod\ p, \tag{24}$$

$$X_i = H_1 X_{i-1} + \cdots + H_k X_{i-k}\ mod\ p, \tag{25}$$

The above automatic generating method is most suitable for Sophine-Germain primes because we can choose R (in step 1 of the AGM algorithm) to achieve a larger possible period when p is a Sophine-Germain prime. Hence, a long cycle of different primitive polynomials $G(x)$ or

$H(x)$ can be generated, which in turn can produce different maximum-period MRGs. When p is a non-Sophine-Germain prime, the AGM algorithm can still be used as long as gcd$(k, p-1)$=1 so that k^{-1} exists under modulus $(p-1)$. One slight drawback is the number of different maximum-period MRGs can be produced is less than those with Sophine-Germain prime modulus. Hence, our web utilized the multiplier B for Sophine-Germain primes from Table 2-29 to generate parallel MRGs.

Previous algorithm is mainly for the case when the prime modulus p is of a Sophine-Germain prime. Deng, Shiau and Tsai [2009] proposed another automatic generation algorithm for the case of non-Sophine-Germain prime. We describe briefly their algorithm below:

Algorithm AGM for non-Sophine-Germain prime. Let $f(x)$ in (3) be a primitive polynomial corresponding to a DL or a DT generator in which the nonzero coefficient is B as in equations (12) or (16). The steps below will randomly generate a set of maximum-period MRGs.

1. Whenever a new processor is initiated, continue generating a new d_n via a simple LCG: $d_n = W d_{n-1} \bmod p$ until $\gcd(kd_n - 1, p-1) = 1$, where W is a primitive element modulus p.
2. Compute $c_n = B^{d_n} \bmod p$ and the primitive polynomial $G(x)$ by
$$G(x) = c_n^{-k} f(c_n x) \equiv x^k - G_1 x^{k-1} - G_2 x^{k-2} - \cdots - G_k \bmod p. \tag{26}$$
and
$$H(x) = -B^{-1} x^k f(c_n/x) \ mod \ p \equiv x^k - H_1 x^{k-1} - H_2 x^{k-2} - \cdots - H_k \bmod p. \tag{27}$$
3. The new processor can use the newly constructed maximum-period MRG corresponding to the characteristic polynomial $G(x)$ in (26) and $H(x)$ in (27) as in equations (24) and (25), respectively.

When the baseline generator is a DL or DT generator, the previously generated MRGs have k nonzero terms. Deng, Shiau and Tsai [2009] presented efficient implementations for the constructed MRGs by a $(k+1)$ recurrence equation.

For these 64-bit and 128-bit generators found (DX, DL, DS and DT), we can certainly follow a similar procedure to construct many efficient MRGs using the same algorithm as described. Since these automatic generation algorithm which may involve some tedious arithmetic operations over a finite field. In the next chapter, we describe our effort to implement the automatic generation method via a web programming so that it can provide interested users with the computer programs for the desired efficient MRGs or automatic generation of MRGs.

7 Web Programming for MRGs and Parallel MRGs

For easy access to the program codes and/or the required parameters for efficient and portable MRGs, we have provided a web page at `http://140.113.114.152/main.html`. This is a temporary site for the experimental purpose only. We are in the process of finding a more permanent website which can be linked to the main home page at `http://www.stat.nctu.edu.tw/` in the near future. For the method of generating parallel MRGs, please see the discussion as before. In addition, we also offer the codes necessary in order to use TestU01 packages. Therefore, the users can easily test the empirical performance of the corresponding MRGs.

We hope to continually improve our web program so that it will provide a more user-friendly interface and more functionality on the information provided. Currently, our web program has four different options as shown below:

Figure 1: Four different options

We explain these four main options for our program in a greater detail: (1) Multiple Recursive Generators which the user can retrieve the parameters for the generator requested (2) Parallel Multiple Recursive Generators which can produce the parameters from a single generator, (3) TestU01 of Multiple provide the program in TestU01 form for the generator requested and (4) TestU01 of Parallel Multiple which can produce the TestU01 program from a single generator. AS shown in Figure 1, users can choose these options by clicking the option for retrieving the program codes, parameters for the efficient and portable MRGs, TestU01 program code for the MRGs or parallel MRGs.

Next, we explain the four options in details for the users inputs and program outputs produced.

7.1 Retrieving parameters for the maximum period MRGs

The first option offer program codes in C of efficient and portable MRGs as shown in Figure 2 is displayed after the first option is selected.

The first input box decides the initial seed chosen from the user for the given efficient MRGs. Here, we use an LCG with multiplier $B = 16807$ and the B given in the corresponding MRG. Using the initial seed input in the first input box, the web program produce an initialization program to generate the initial $X_0, X_1, \cdots, X_{k-1}$. Specifically, $X_i = BX_{i-1} \bmod p$ for $i = 2, 3, \cdots, k$ with X_0 being the initial seed selected. The prime modulus p is selected automatically

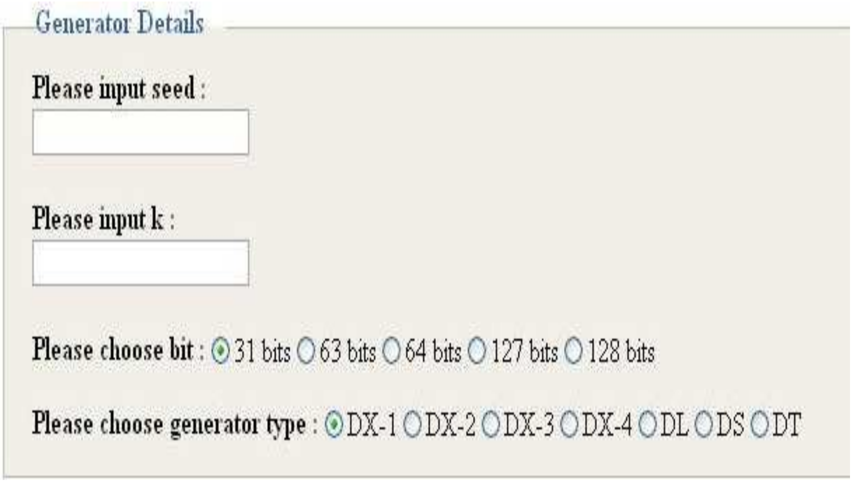

Figure 2: Interface for Efficient and Portable MRGs

from the database with k set by the user who enters second input box for the order k. When the user enters an approximate value of order k, the web program will automatically find the nearest prime order k which have been stored in the database. The values for the order k available are ranging from 101 to 4001 with interval of one hundred for 32-bit generators. For 63-bit, 64-bit, 127-bit and 128-bit generators, the values of k are from 101 to 2003 with interval of one hundred. The upper limit of k maybe updated as we find larger order MRGs. We need two more inputs from the user: (1) the size of the prime modulus $p = 2^d - c$, where the bit d can be 31, 63, 64, 127, or 128 bit. (2) the type of efficient MRGs as discussed previously which can be DX-k-s with $s = 1, 2, 3, 4$ or DL-k, DS-k, or DT-k. After selecting the bit (d) and type of efficient and portable MRGs, the user can press the "send" button. The program code for the corresponding MRG and its parameter would be shown as in Figure 3 below.

Figure 3 is an example using the values initial seed = 123, order k = 110 (which would be replaced by the closest order k = 101 in the result), bit = 31 and DX-1 type.

The first part of the output as shown in Figure 3 is:

$$X_i = X_{i-1} + BX_{i-101} \tag{28}$$

which displays the generator's (in this case, DX-101-1) recurrence equation. The actual program implementation of this generator can be downloaded which is shown as

Finally, the initialization part of the program

$$\text{DX\_s1\_initial\_seeds}(1048575, 101, 2147400803, 123); \tag{29}$$

which is the code to initiate the start seeds. The following program is automatically produced

```
typedef unsigned long long int64;
typedef struct {
int B, PP, KK;
double PP_inv;
} param_DX_s1;
typedef struct {
int64 *XX;
int II;
} state_DX_s1;
#define DMOD(n, p) ((n) % (p))

param_DX_s1 DX_s1_param ;
state_DX_s1 DX_s1_state ;

void DX_s1_initial_seeds(int B, int k, int m, int seed)
{
DX_s1_param.KK = k ;
DX_s1_param.PP = m ;
DX_s1_param.B = B ;

int i;

DX_s1_state.XX = (int64*)malloc(sizeof(int64)*DX_s1_param.KK);
DX_s1_state.XX[0] = seed;

for(i=1; i < DX_s1_param.KK ; i++)
   DX_s1_state.XX[i] = DMOD( 16807 * DX_s1_state.XX[i-1], DX_s1_param.PP);

DX_s1_param.PP_inv = 1.0/DX_s1_param.PP;
DX_s1_state.II = DX_s1_param.KK-1; /* running index */
}
```

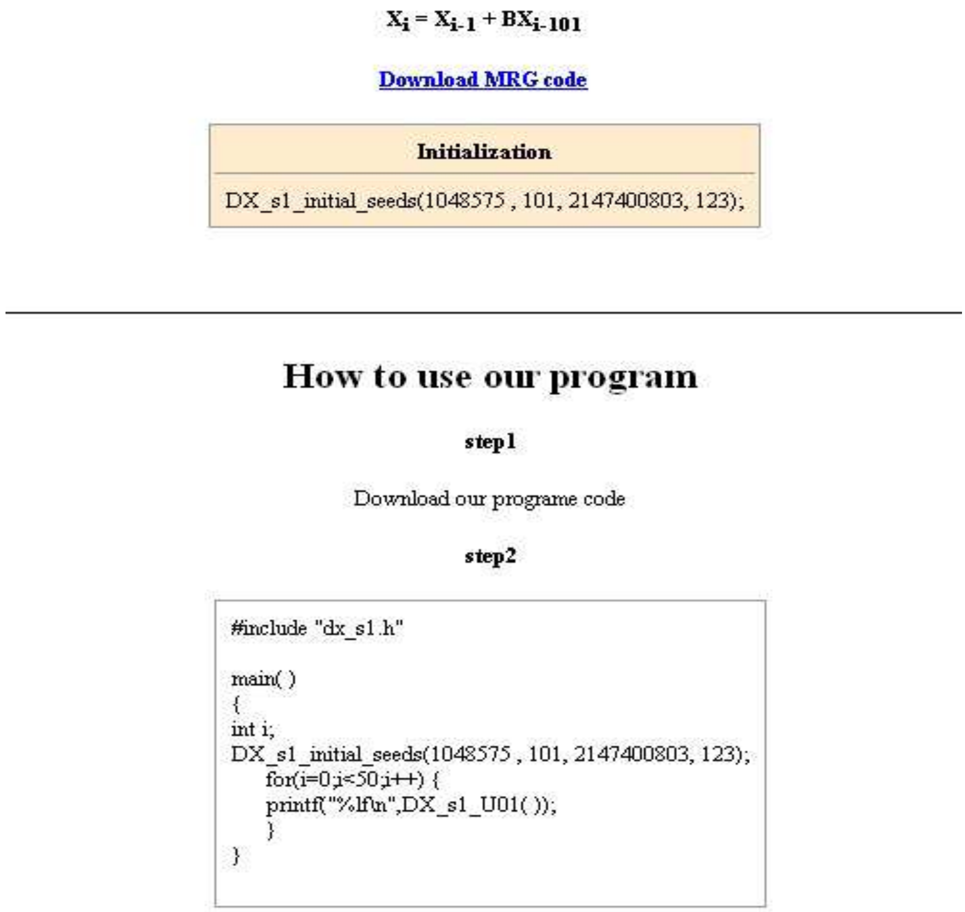

Figure 3: Result for Efficient and Portable MRGs
by our web program

which is a part of the downloaded file for code of the chosen MRG. In (29), the fist parameter 1048575 is the multiplier B in (28); the second parameter 101 is the order k; the third parameter is the modulus p of chosen MRG; the final parameter 123 is the initial seed. The last part of the output gives a simple illustration for using our code. When the option of 63-bit, 64-bit, 127-bit or 128-bit generators is selected, our web program can provide an implementation with codes written for 32-bit CPU with the help of a general multi-precision package called GMP.

7.2 Retrieving programs for parallel MRGs

The second option from the main menu is "Parallel Multiple Recursive Generators" which can provide the program code for generating several MRGs from a single maximum period MRG like DX, DL, DS, or DT generators. Please refer the previous discussion on the steps for a parallel random number generation.

```
double DX_s1_U01( )
{
int II0 = DX_s1_state.II;
DX_s1_state.II = (DX_s1_state.II+1)%DX_s1_param.KK;
DX_s1_state.XX[DX_s1_state.II] = DMOD( DX_s1_param.B*DX_s1_state.XX[DX_s1_state.II]
+ DX_s1_state.XX[II0], DX_s1_param.PP );
return (double)( DX_s1_state.XX[ DX_s1_state.II ] + 0.5 )*DX_s1_param.PP_inv;
}
```

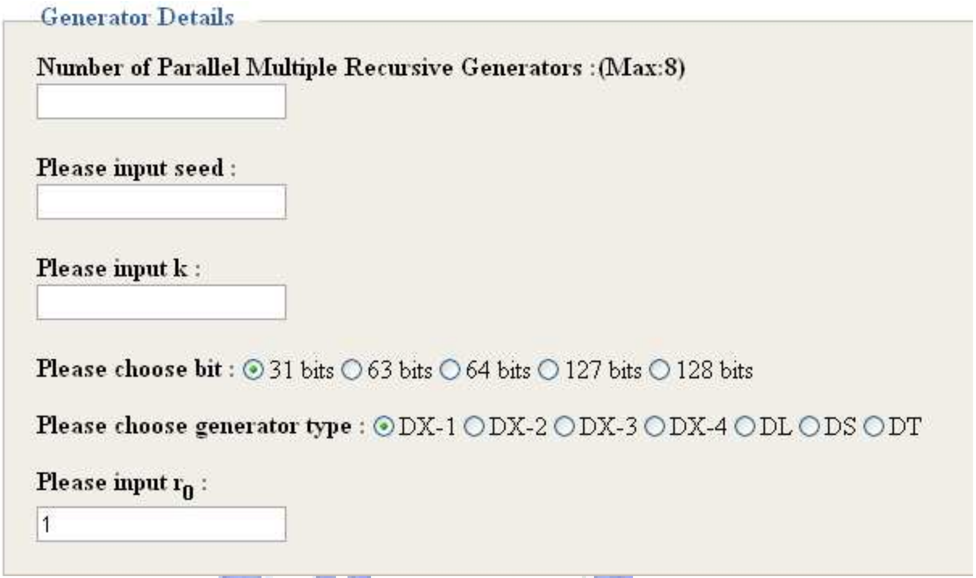

Figure 4: Interface for Parallel MRGs

Figure 4 is the display after the user has chosen the second option from the main menu. The first input box decides the number of parallel MRGs to yield. For some technical reasons, the maximum value that a user can produce at a time is 8. If more than 8 MRGs are needed, the user can get more by entering another value of the r_0. The second input box is the initial sees which is the same as before. Like the first option from the main menu, the next two inputs are for the types of MRGs requested. The final input box is decides r_0 which is the initial r_n in (20). The stating value for the generation of r_i which is 1. As mentioned, if more different generators needed, one can change the value on suitable r_0. For example, after getting 8 parallel MRG's codes, the user can get another 8 parallel MRG's codes by changing the last input box to a new value of r_0 which is automatically set after the program excution. All the user is required to do is to press the "send" button. We give a simple illustration for the output from the parallel MRG code and parameter below.

Figure 5 is an example of the result of pressing the "send" button. Similar to the result of

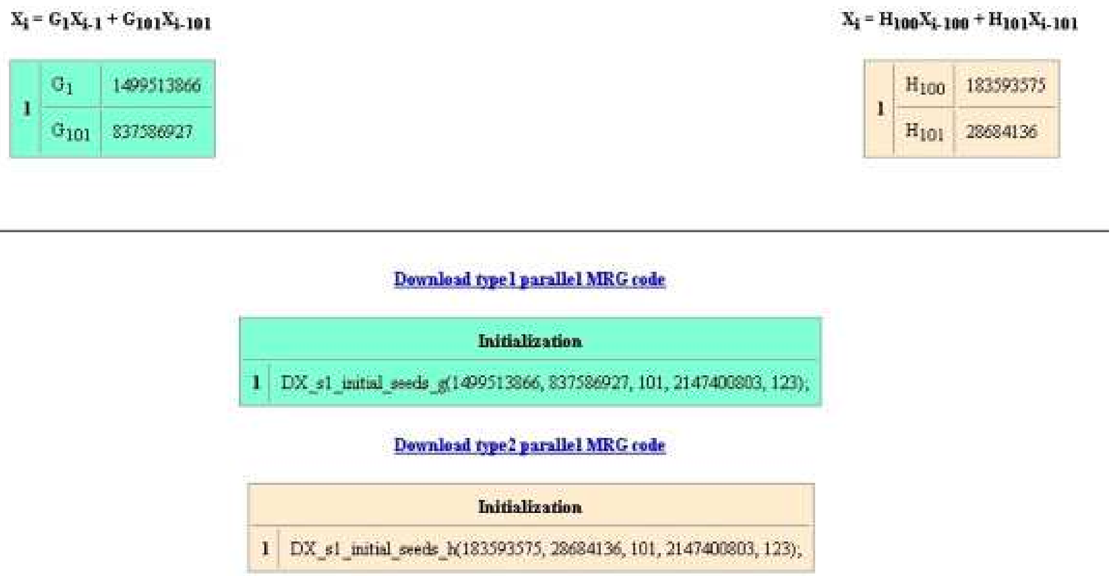

Figure 5: Result for Parallel MRGs

first type, the difference is that we utilize AGM algorithm to generate parallel MRGs, so there would appear two kinds of parallel MRGs (G(x) in (24) and H(x) in (25)).

7.3 Retrieving TestU01 code for the maximum period MRGs

The third option and fourth option are similar to that of the first and second option. The main difference is the program code is specifically written for the use of testing efficient and portable MRGs in TestU01. Specifically, the third option can produce the program for testing some specific MRG requested by TestU01 test suite. Similarly, the forth type will produce codes for parallel MRGs to be tested by TestU01 test suite. Their interfaces are very similar to the first option and the second option from the main menu respectively. As mentioned, the key difference from the first option and the second option is that the TestU01 package codes that are produced to users to evaluate empirical performance of the generators in our web. Before implementing these codes, installation of TestU01 package is needed first. The users can also download the codes from our website and follow our website's directions to implement.

In addition to the web program as described previously, we also develop another system which is easier to use and maintain than the current web program. This package has the advantage of efficiency with same functionality and the same features. Users can download the compiled programs and run them locally on their PCs without running the program online. The package will be available online soon at the same website.

8 Conclusion

We extend the computer search of 32-bit efficient and portable MRGs to the search of MRGs for 64-bit and 128-bit CPUs. Many of these generators have been found for various type of efficient generators and they are listed in various tables. We test their empirical performance by utilizing the TestU01 package and the empirical tests results show that their all pass the TestU01 test suite. We also compare them with other 64-bit random number generators in terms of period length and the property of equi-distribution. We believe that these efficient MRGs should be a great choice for applications running on 64-bit or 128-bit CPUs. For the convenience of users, we designed a web program to produce the program codes and provide parameters for our generators.

References

[1] L. Y. Deng. Generalized mersenne prime number and its application to random number generation. In H. Niederreiter, editor, *Monte Carlo and Quasi-Monte Carlo Methods 2002*, pages 167–180. Springer-Verlag, 2004.

[2] L. Y. Deng. Efficient and portable multiple recursive generators of large order. *ACM Transactions on Modeling and Computer Simulation*, 15(1):1–13, 2005.

[3] L. Y. Deng. Issues on computer search for large order multiple recursive generators. In S. Heinrich, A. Keller, and H. Niederreiter, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2006*, pages 251–261. Springer-Verlag, 2008.

[4] L. Y. Deng and E. O. George. Generation of uniform variates from several nearly uniformly distributed variables. *Communications in Statistics B*, 19, 1990.

[5] L. Y. Deng, H. Li, and J.-J. H. Shiau. Scalable parallel multiple recursive generators of large order. *Parallel Computing*, 35:29–37, 2009.

[6] L. Y. Deng and D. K. J. Lin. Random number generation for the new century. *American Statistician*, pages 145–150, 2000.

[7] L. Y. Deng, D. K. J. Lin, J. Wang, and Y. Yuan. Statistical justification of combination generators. *Statistica Sinica*, 7:993–1003, 1997.

[8] L. Y. Deng, H. S Lu, and T. B Chen. 64-bit and 128-bit dx random number generators. 2009. Preprint.

[9] L. Y. Deng, J. J. Horng Shiau, and G. H. Tsai. Parallel random number generators based on large order multiple recursive generators. In Pierre L'Ecuyer and Art B. Owen, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2008*. Springer-Verlag, 2009. (to appear).

[10] L. Y. Deng and H. Xu. A system of high-dimensional, efficient, long-cycle and portable uniform random number generators. *ACM Transactions on Modeling and Computer Simulation*, 13(4):299–309, 2003.

[11] J. E. Gentle. *Random Number Generation and Monte Carlo Methods, 2nd ed.* Springer-Verlag, New York, 2003.

[12] W. Hormann. A note on the quality of random variates generated by the ratio of uniforms method. *ACM Transactions on Modeling and Computer Simulation*, 4:96–106, 1994.

[13] A. J. Kinderman and J. F. Monahan. Computer generation of random variables using ratio of uniform deviates. *ACM Transactions on Mathematical Software*, 3:257–260, 1997.

[14] P. L'Ecuyer. Efficient and portable combined random number generators. *Communications of the ACM*, 31:742–748, 774, 1988.

[15] P. L'Ecuyer. Bad lattice structures for vectors of non-successive values produced by some linear recurrences. *INFORMS Journal on Computing*, 9:57–60, 1997.

[16] P. L'Ecuyer and F. Blouin. Linear congruential generators of order $k > 1$. In *1988 Winter Simulation Conference Proceedings*, pages 432–439, 1988.

[17] P. L'Ecuyer, F. Blouin, and R. Couture. A search for good multiple recursive linear random number generators. *ACM Transactions on Modeling and Computer Simulation*, 3:87–98, 1993.

[18] D. H. Lehmer. Mathematical methods in large-scale computing units. In *Proceedings of the Second Symposium on Large Scale Digital Computing Machinery*, pages 141–146, Cambridge, MA., 1951. Harvard University Press.

[19] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and Their Applications*. Cambridge University Press, Cambridge, MA., revised edition, 1994.

[20] G. Marsaglia. Random numbers fall mainly in the planes. In *Proceedings of the National Academy of Sciences*, volume 61, pages 25–28, 1968.

[21] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–20, 1998.

[22] M. Matsumoto, I. Wada, A. Kuramoto, and H. Ashihara. Common defects in initialization of pseudorandom number generators. *ACM Transactions on Modeling and Computer Simulation*, 17:15, 2007.

[23] T. Nishimura. Tables of 64-bit mersenne twisters. *ACM Transactions on Modeling and Computer*, 10(348-357), 2000.

[24] W. H. Payne, J. R. Rabung, and T. Bogyo. Coding the lehmer pseudo number generator. *Communications of the ACM*, 12:85–86, 1969.

Table 1: List of k and c for non-Sophine-Germain vs. Sophine-Germain prime, where $p = 2^d - c$

	Sophine-Gernain primes				non-Sophine-Gernain primes			
k	$c(63)$	$c(64)$	$c(127)$	$c(128)$	$c(63)$	$c(64)$	$c(127)$	$c(128)$
101	2941809	103709	8023365	781733	45831	5939	66567	122069
211	969741	2323877	11501829	14363333	7927	73347	67231	162785
307	3400329	9123149	12818949	67573457	11725	114363	281875	22637
401	402105	5109569	10064781	9780293	58959	78207	682617	1953
503	8175705	610553	154659081	25760477	39835	143067	81777	265845
601	3997821	1178813	142397385	29337077	59889	113579	120295	1514669
701	1137009	3863129	31187169	49288097	10465	3497	130395	198653
809	6373005	17589113	81710265	440234213	118017	220647	26941	1691159
907	7416321	2012513	26968581	31065533	169761	26697	314731	793485
1009	6182529	21298889	2451789	170478209	56451	200987	876459	200399
1103	30158505	7366769	253989021	181533689	80059	234945	957505	594789
1201	6186009	8355149	29068281	81181637	461251	186425	360909	412995
1301	3241965	9528257	79595481	283176053	407455	480099	407797	480893
1409	11522061	3454937	21557661	587284637	250815	470015	1673445	1027599
1511	26619045	16445057	185276181	83322269	694161	613443	6720469	678449
1601	20211141	50510633	175492881	68485229	886675	153513	1829847	1716617
1709	5033901	1210769	31346721	30085217	33687	432599	1166731	320613
1801	1742625	13468637	84200469	98538209	100299	261659	1623471	921513
1901	39182001	22944173	155499561	93603137	1124097	1098143	1930005	2374503
2003	50336361	24417377	9580245	142218077	1487491	1037939	1827495	1201557

Table 2: List of B for DX-k-1 generator with $p = 2^{63} - c$

	$p = 2^{63} - c$ is non-Sophine-Germain			$p = 2^{63} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{31}$	c	min B	$B < 2^{31}$
101	45831	26	2147483526	2941809	104	2147483368
211	7927	855	2147483601	969741	126	2147483129
307	11725	280	2147483304	3400329	351	2147483549
401	58959	2109	2147483469	402105	94	2147482138
503	39835	8278	2147483371	8175705	500	2147483268
601	59889	461	2147483110	3997821	245	2147483420
701	10465	2990	2147482879	1137009	1971	2147482313
809	118017	222	2147482539	6373005	2972	2147482662
907	169761	1895	2147483395	7416321	211	2147482851
1009	56451	2636	2147481897	6182529	4995	2147483149
1103	80059	1780	2147480806	30158505	297	2147481846
1201	461251	17651	2147483335	6186009	1811	2147473205
1301	407455	1241	2147482933	3241965	1283	2147482301
1409	250815	3647	2147483348	11522061	4704	2147482492
1511	694161	12455	2147481410	26619045	2842	2147483328
1601	886675	2734	2147483127	20211141	302	2147483444
1709	33687	8868	2147482898	5033901	839	2147482508
1801	100299	6665	2147483482	1742625	3672	2147472396
1901	1124097	4463	2147483491	39182001	12415	2147475250
2003	1487491	9247	2147480835	50336361	11647	2147483078

Table 3: List of B for DX-k-2 generator with $p = 2^{63} - c$

	$p = 2^{63} - c$ is non-Sophine-Germain			$p = 2^{63} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{31}$	c	min B	$B < 2^{31}$
101	45831	197	2147483264	2941809	118	2147483606
211	7927	1909	2147483578	969741	277	2147483390
307	11725	1441	2147483148	3400329	1090	2147483577
401	58959	163	2147483548	402105	402	2147481939
503	39835	4378	2147483576	8175705	266	2147482176
601	59889	1376	2147483194	3997821	2400	2147483197
701	10465	360	2147482223	1137009	224	2147483513
809	118017	2566	2147482767	6373005	2522	2147483487
907	169761	590	2147481107	7416321	274	2147482426
1009	56451	421	2147483457	6182529	524	2147480890
1103	80059	5440	2147481441	30158505	1841	2147483393
1201	461251	3642	2147482727	6186009	1313	2147482568
1301	407455	2828	2147480919	3241965	1044	2147474911
1409	250815	7762	2147483482	11522061	5473	2147482526
1511	694161	7103	2147483251	26619045	3326	2147482443
1601	886675	1715	2147480687	20211141	12479	2147481370
1709	33687	1551	2147483451	5033901	3228	2147481121
1801	100299	4168	2147479165	1742625	4974	2147482748
1901	1124097	5444	2147482978	39182001	3291	2147477245
2003	1487491	8838	2147482021	50336361	221	2147482641

Table 4: List of B for DX-k-3 generator with $p = 2^{63} - c$

	$p = 2^{63} - c$ is non-Sophine-Germain			$p = 2^{63} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{31}$	c	min B	$B < 2^{31}$
101	45831	54	2147483599	2941809	104	2147483358
211	7927	122	2147483264	969741	96	2147483346
307	11725	851	2147483597	3400329	796	2147483009
401	58959	132	2147483641	402105	126	2147483261
503	39835	1588	2147482998	8175705	236	2147479944
601	59889	3660	2147483068	3997821	11	2147483049
701	10465	832	2147483500	1137009	1494	2147481463
809	118017	666	2147482363	6373005	1196	2147480247
907	169761	1306	2147483285	7416321	844	2147483367
1009	56451	61	2147483347	6182529	2319	2147474619
1103	80059	334	2147480840	30158505	1127	2147480008
1201	461251	5275	2147482709	6186009	3975	2147483174
1301	407455	1469	2147476210	3241965	665	2147483192
1409	250815	3304	2147478837	11522061	2868	2147481028
1511	694161	3997	2147477397	26619045	9623	2147471141
1601	886675	6975	2147480346	20211141	487	2147479893
1709	33687	698	2147481726	5033901	4826	2147480530
1801	100299	17	2147481304	1742625	345	2147472252
1901	1124097	4463	2147483343	39182001	8971	2147481116
2003	1487491	653	2147481558	50336361	11011	2147475690

Table 5: List of B for DX-k-4 generator with $p = 2^{63} - c$

	$p = 2^{63} - c$ is non-Sophine-Germain			$p = 2^{63} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{31}$	c	min B	$B < 2^{31}$
101	45831	175	2147483572	2941809	114	2147483434
211	7927	656	2147483526	969741	110	2147483557
307	11725	2395	2147482892	3400329	211	2147483004
401	58959	271	2147483322	402105	490	2147482844
503	39835	5107	2147483424	8175705	449	2147483019
601	59889	118	2147483633	3997821	372	2147482652
701	10465	2167	2147483260	1137009	1453	2147483063
809	118017	186	2147483607	6373005	340	2147482951
907	169761	929	2147482642	7416321	39	2147482515
1009	56451	1825	2147482834	6182529	4977	2147482952
1103	80059	826	2147483607	30158505	136	2147482724
1201	461251	3083	2147479714	6186009	1241	2147482893
1301	407455	898	2147483489	3241965	76	2147482137
1409	250815	712	2147483354	11522061	150	2147481062
1511	694161	2294	2147478183	26619045	2798	2147479114
1601	886675	4423	2147478997	20211141	271	2147477911
1709	33687	1693	2147483636	5033901	134	2147478682
1801	100299	1364	2147483380	1742625	823	2147483334
1901	1124097	3298	2147483132	39182001	11257	2147476729
2003	1487491	3798	2147481287	50336361	6976	2147469339

Table 6: List of B for DL-k generator with $p = 2^{63} - c$

	$p = 2^{63} - c$ is non-Sophine-Germain			$p = 2^{63} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{31}$	c	min B	$B < 2^{31}$
101	45831	218	2147483335	2941809	293	2147483445
211	7927	527	2147483373	969741	408	2147483005
307	11725	471	2147483544	3400329	376	2147483489
401	58959	71	2147482373	402105	104	2147482175
503	39835	1595	2147482926	8175705	385	2147483470
601	59889	4434	2147483583	3997821	110	2147476341
701	10465	348	2147481862	1137009	1085	2147481841
809	118017	637	2147482706	6373005	568	2147482003
907	169761	1714	2147483617	7416321	2370	2147483619
1009	56451	4409	2147483197	6182529	1504	2147482772
1103	80059	1450	2147481499	30158505	1805	2147483318
1201	461251	4372	2147479694	6186009	91	2147483626
1301	407455	89	2147483280	3241965	2015	2147483156
1409	250815	4787	2147482484	11522061	4469	2147483315
1511	694161	1630	2147482762	26619045	2712	2147482070
1601	886675	14451	2147478915	20211141	1107	2147478170
1709	33687	73	2147482947	5033901	7251	2147478980
1801	100299	5065	2147481971	1742625	971	2147477064
1901	1124097	370	2147482662	39182001	466	2147481749
2003	1487491	3245	2147482237	50336361	398	2147479809

Table 7: List of B for DS-k generator with $p = 2^{63} - c$

	$p = 2^{63} - c$ is non-Sophine-Germain			$p = 2^{63} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{31}$	c	min B	$B < 2^{31}$
101	45831	79	2147483026	2941809	552	2147483494
211	7927	116	2147483143	969741	294	2147483646
307	11725	30	2147483502	3400329	179	2147482907
401	58959	2431	2147483577	402105	1815	2147482969
503	39835	3421	2147483450	8175705	436	2147483115
601	59889	2066	2147483398	3997821	535	2147483346
701	10465	2825	2147483263	1137009	873	2147482067
809	118017	8221	2147482652	6373005	203	2147480927
907	169761	365	2147482847	7416321	129	2147483088
1009	56451	308	2147481905	6182529	1492	2147483415
1103	80059	2812	2147478286	30158505	5247	2147481149
1201	461251	299	2147483077	6186009	1270	2147483407
1301	407455	1880	2147482037	3241965	3828	2147483496
1409	250815	626	2147481359	11522061	1559	2147481255
1511	694161	7946	2147483643	26619045	1674	2147483470
1601	886675	4155	2147482064	20211141	1350	2147483171
1709	33687	2818	2147482723	5033901	3805	2147483070
1801	100299	3027	2147483592	1742625	682	2147480126
1901	1124097	4906	2147483550	39182001	7802	2147479190
2003	1487491	1144	2147480032	50336361	2363	2147483415

Table 8: List of B for DT-k generator with $p = 2^{63} - c$

	$p = 2^{63} - c$ is non-Sophine-Germain		$p = 2^{63} - c$ is Sophine-Germain	
k	w	min B	w	min B
101	45831	374	2941809	58
211	7927	615	969741	217
307	11725	153	3400329	1496
401	58959	959	402105	2149
503	39835	246	8175705	1909
601	59889	3296	3997821	3644
701	10465	1128	1137009	641
809	118017	1226	6373005	301
907	169761	2239	7416321	626
1009	56451	878	6182529	3129
1103	80059	658	30158505	7386
1201	461251	4777	6186009	2598
1301	407455	3018	3241965	2298
1409	250815	5446	11522061	1493
1511	694161	4559	26619045	615
1601	886675	2061	20211141	60
1709	33687	14981	5033901	573
1801	100299	1799	1742625	103
1901	1124097	10220	39182001	3858
2003	1487491	2771	50336361	7664

Table 9: List of B for DX-k-1 generator with $p = 2^{64} - c$

	$p = 2^{64} - c$ is non-Sophine-Germain			$p = 2^{64} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{32}$	c	min B	$B < 2^{32}$
101	5939	41	4294967161	103709	240	4294967293
211	73347	57	4294967250	2323877	120	4294967052
307	114363	500	4294966870	9123149	99	4294967295
401	78207	1093	4294966652	5109569	770	4294967137
503	143067	873	4294966842	610553	373	4294966514
601	113579	38	4294967009	1178813	1802	4294966786
701	3497	1940	4294966324	3863129	358	4294965635
809	220647	2369	4294967127	17589113	1329	4294965606
907	26697	3056	4294966730	2012513	1047	4294966905
1009	200987	401	4294966508	21298889	4708	4294960490
1103	234945	4568	4294966133	7366769	73	4294961971
1201	186425	7487	4294966411	8355149	4798	4294966708
1301	480099	11814	4294967259	9528257	11540	4294966815
1409	470015	2316	4294967126	3454937	14801	4294964133
1511	613443	2896	4294967121	16445057	9257	4294966976
1601	153513	3014	4294967169	50510633	4092	4294965732
1709	432599	9865	4294966674	1210769	2979	4294958637
1801	261659	307	4294963908	13468637	5427	4294965796
1901	1098143	2714	4294966614	22944173	906	4294964894
2003	1037939	132	4294966884	24417377	1336	4294959090

Table 10: List of B for DX-k-2 generator with $p = 2^{64} - c$

	$p = 2^{64} - c$ is non-Sophine-Germain			$p = 2^{64} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{32}$	c	min B	$B < 2^{32}$
101	5939	195	4294967273	103709	251	4294966629
211	73347	634	4294967259	2323877	541	4294966680
307	114363	1893	4294967174	9123149	477	4294966991
401	78207	1992	4294967224	5109569	34	4294966905
503	143067	694	4294966989	610553	2802	4294965530
601	113579	4617	4294966581	1178813	845	4294967135
701	3497	634	4294966563	3863129	2089	4294966321
809	220647	449	4294967071	17589113	487	4294964532
907	26697	662	4294967215	2012513	454	4294967254
1009	200987	661	4294966926	21298889	681	4294963149
1103	234945	2742	4294967240	7366769	2221	4294965233
1201	186425	1161	4294963023	8355149	5904	4294966586
1301	480099	16221	4294967113	9528257	55	4294962681
1409	470015	2802	4294965895	3454937	728	4294965185
1511	613443	1052	4294964981	16445057	854	4294966049
1601	153513	3080	4294965560	50510633	401	4294964144
1709	432599	2885	4294966730	1210769	2094	4294966953
1801	261659	1791	4294964628	13468637	5787	4294951553
1901	1098143	10	4294964255	22944173	8047	4294955247
2003	1037939	856	4294964310	24417377	14612	4294965294

Table 11: List of B for DX-k-3 generator with $p = 2^{64} - c$

	$p = 2^{64} - c$ is non-Sophine-Germain			$p = 2^{64} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{32}$	c	min B	$B < 2^{32}$
101	5939	573	4294967205	103709	245	4294967266
211	73347	254	4294967130	2323877	1204	4294966998
307	114363	185	4294967276	9123149	189	4294964840
401	78207	488	4294965956	5109569	657	4294967061
503	143067	1384	4294966836	610553	360	4294967140
601	113579	1136	4294966614	1178813	1498	4294967290
701	3497	526	4294965911	3863129	15	4294964482
809	220647	992	4294966702	17589113	172	4294966247
907	26697	5186	4294966774	2012513	1417	4294969750
1009	200987	171	4294966679	21298889	2822	4294965726
1103	234945	5002	4294966688	7366769	4354	4294965873
1201	186425	7390	4294965837	8355149	112	4294966096
1301	480099	2889	4294967122	9528257	39	4294961896
1409	470015	2412	4294967085	3454937	728	4294965171
1511	613443	10700	4294966167	16445057	3336	4294955652
1601	153513	4211	4294965295	50510633	748	4294962750
1709	432599	765	4294965438	1210769	279	4294964843
1801	261659	5951	4294967201	13468637	40	4294957747
1901	1098143	4854	4294961938	22944173	6850	4294964279
2003	1037939	2138	4294966104	24417377	650	4294966876

Table 12: List of B for DX-k-4 generator with $p = 2^{64} - c$

	$p = 2^{64} - c$ is non-Sophine-Germain			$p = 2^{64} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{32}$	c	min B	$B < 2^{32}$
101	5939	263	4294967235	103709	2	4294966829
211	73347	490	4294967037	2323877	242	4294966783
307	114363	1498	4294966917	9123149	59	4294967229
401	78207	38	4294967006	5109569	2864	4294967162
503	143067	1134	4294966946	610553	956	4294966521
601	113579	1696	4294967038	1178813	244	4294965884
701	3497	1096	4294967196	3863129	1550	4294964225
809	220647	1797	4294966625	17589113	2293	4294967220
907	26697	752	4294967176	2012513	1615	4294966316
1009	200987	897	4294963844	21298889	3966	4294966465
1103	234945	519	4294967109	7366769	2195	4294965920
1201	186425	5326	4294967055	8355149	1465	4294964888
1301	480099	10739	4294967241	9528257	2563	4294962561
1409	470015	1888	4294967290	3454937	3909	4294959534
1511	613443	5787	4294967020	16445057	1528	4294965548
1601	153513	2813	4294966129	50510633	2981	4294964884
1709	432599	616	4294965791	1210769	5103	4294962652
1801	261659	1088	4294965568	13468637	4411	4294966409
1901	1098143	8208	4294962260	22944173	6524	4294957890
2003	1037939	6823	4294966494	24417377	4273	4294966952

Table 13: List of B for DL-k generator with $p = 2^{64} - c$

	$p = 2^{64} - c$ is non-Sophine-Germain			$p = 2^{64} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{32}$	c	min B	$B < 2^{32}$
101	5939	163	4294967274	103709	94	4294966762
211	73347	2591	4294966974	2323877	512	4294966846
307	114363	459	4294967131	9123149	642	4294967009
401	78207	1314	4294967017	5109569	1486	4294966377
503	143067	989	4294967048	610553	953	4294967270
601	113579	122	4294967034	1178813	2660	4294966717
701	3497	652	4294965769	3863129	2189	4294966112
809	220647	94	4294967208	17589113	2499	4294966481
907	26697	544	4294964979	2012513	220	4294966440
1009	200987	2697	4294967032	21298889	188	4294962642
1103	234945	749	4294966992	7366769	645	4294967101
1201	186425	7357	4294965945	8355149	329	4294966479
1301	480099	9243	4294964941	9528257	7702	4294965859
1409	470015	8246	4294967102	3454937	727	4294967103
1511	613443	1376	4294966814	16445057	549	4294967287
1601	153513	1214	4294964557	50510633	6169	4294960745
1709	432599	2924	4294966052	1210769	4682	4294965494
1801	261659	5385	4294966888	13468637	1825	4294960958
1901	1098143	1263	4294966443	22944173	5901	4294961633
2003	1037939	1404	4294967136	24417377	2843	4294962305

Table 14: List of B for DS-k generator with $p = 2^{64} - c$

	$p = 2^{64} - c$ is non-Sophine-Germain			$p = 2^{64} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{32}$	c	min B	$B < 2^{32}$
101	5939	163	4294967294	103709	246	4294967236
211	73347	6	4294967089	2323877	97	4294967195
307	114363	21	4294967097	9123149	448	4294966500
401	78207	1844	4294967271	5109569	71	4294967266
503	143067	4599	4294967195	610553	1096	4294967230
601	113579	162	4294967183	1178813	671	4294962694
701	3497	619	4294966764	3863129	31	4294966914
809	220647	1456	4294967168	17589113	1948	4294967227
907	26697	5322	4294966891	2012513	3422	4294966689
1009	200987	8192	4294967259	21298889	1417	4294967024
1103	234945	4986	4294965247	7366769	7485	4294967165
1201	186425	591	4294967003	8355149	6975	4294966495
1301	480099	7382	4294966789	9528257	346	4294966306
1409	470015	4932	4294967007	3454937	1408	4294965267
1511	613443	2339	4294967256	16445057	1833	4294966620
1601	153513	1632	4294967029	50510633	2743	4294960474
1709	432599	7639	4294966741	1210769	778	4294964359
1801	261659	2970	4294966961	13468637	651	4294963707
1901	1098143	271	4294961240	22944173	2093	4294965677
2003	1037939	928	4294967145	24417377	977	4294963595

Table 15: List of B for DT-k generator with $p = 2^{64} - c$

	$p = 2^{64} - c$ is non-Sophine-Germain		$p = 2^{64} - c$ is Sophine-Germain	
k	w	min B	w	min B
101	5939	675	103709	136
211	73347	891	2323877	515
307	114363	1590	9123149	20
401	78207	2010	5109569	505
503	143067	1720	610553	2037
601	113579	2283	1178813	1248
701	3497	634	3863129	1760
809	220647	319	17589113	2528
907	26697	1045	2012513	2908
1009	200987	838	21298889	1999
1103	234945	892	7366769	2337
1201	186425	937	8355149	2348
1301	480099	227	9528257	6191
1409	470015	3877	3454937	3258
1511	613443	572	16445057	208
1601	153513	645	50510633	4538
1709	432599	4030	1210769	5553
1801	261659	321	13468637	1084
1901	1098143	1535	22944173	12465
2003	1037939	2238	24417377	1055

Table 16: List of B for DX-k-1 generator with $p = 2^{127} - c$, where x = 92233720368547

	$p = 2^{127} - c$ is non-Sophine-Germain			$p = 2^{127} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{63}$	c	min B	$B < 2^{63}$
101	66567	505	x75744	8023365	63	x75754
211	67231	76	x75765	11501829	194	x75540
307	281875	556	x75544	12818949	92	x73423
401	682617	466	x75553	10064781	911	x75653
503	81777	3080	x75595	154659081	1740	x75721
601	120295	3110	x74491	142397385	764	x73801
701	130395	1523	x74620	31187169	601	x74833
809	26941	1615	x75776	81710265	1799	x75068
907	314731	26	x74171	26968581	637	x74754
1009	876459	6014	x74296	2451789	3822	x75062
1103	957505	2335	x75735	253989021	1255	x72352
1201	360909	58	x75272	29068281	130	x70911
1301	407797	10336	x72176	79595481	5503	x75043
1409	1673445	9594	x74717	21557661	839	x75189
1511	6720469	3255	x75447	185276181	3516	x75454
1601	1829847	3733	x72782	175492881	6364	x73854
1709	1166731	5550	x75249	31346721	510	x75037
1801	1623471	3089	x72295	84200469	948	x75127
1901	1930005	720	x71454	155499561	3414	x71355
2003	1827495	8743	x75449	9580245	2678	x75040

Table 17: List of B for DX-k-2 generator with $p = 2^{127} - c$, where x = 92233720368547

	$p = 2^{127} - c$ is non-Sophine-Germain			$p = 2^{127} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{63}$	c	min B	$B < 2^{63}$
101	66567	902	x75659	8023365	26	x75734
211	67231	939	x75622	11501829	59	x75744
307	281875	911	x75549	12818949	788	x75596
401	682617	1580	x75401	10064781	504	x75476
503	81777	3222	x72697	154659081	757	x74786
601	120295	3135	x75635	142397385	978	x71385
701	130395	111	x75510	31187169	921	x74652
809	26941	200	x75011	81710265	454	x72890
907	314731	8483	x75288	26968581	1551	x75689
1009	876459	389	x75077	2451789	439	x75467
1103	957505	1587	x75650	253989021	995	x74786
1201	360909	8698	x75187	29068281	376	x71926
1301	407797	1143	x72387	79595481	153	x75685
1409	1673445	3728	x73656	21557661	2039	x72502
1511	6720469	836	x75554	185276181	1633	x70506
1601	1829847	1274	x75306	175492881	1848	x70662
1709	1166731	1569	x69739	31346721	582	x73774
1801	1623471	6192	x75256	84200469	304	x75430
1901	1930005	688	x75392	155499561	21780	x72224
2003	1827495	3927	x74456	9580245	421	x75203

Table 18: List of B for DX-k-3 generator with $p = 2^{127} - c$, where x = 92233720368547

	$p = 2^{127} - c$ is non-Sophine-Germain			$p = 2^{127} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{63}$	c	min B	$B < 2^{63}$
101	66567	175	x75679	8023365	134	x75500
211	67231	662	x75686	11501829	611	x75774
307	281875	478	x75496	12818949	255	x74819
401	682617	537	x75699	10064781	3054	x74767
503	81777	2047	x75791	154659081	505	x75196
601	120295	799	x75118	142397385	630	x75168
701	130395	1658	x75700	31187169	3060	x74702
809	26941	4416	x75696	81710265	23	x74963
907	314731	4460	x75343	26968581	1463	x74714
1009	876459	3838	x75605	2451789	4577	x74828
1103	957505	4568	x75305	253989021	173	x74774
1201	360909	1016	x74699	29068281	1745	x75307
1301	407797	1020	x74438	79595481	1735	x67518
1409	1673445	3066	x72661	21557661	677	x74760
1511	6720469	2868	x74716	185276181	3911	x75101
1601	1829847	468	x75802	175492881	1090	x74222
1709	1166731	1178	x74972	31346721	769	x70206
1801	1623471	2493	x71827	84200469	10	x75369
1901	1930005	9600	x75719	155499561	3816	x74208
2003	1827495	1092	x75184	9580245	6865	x74663

Table 19: List of B for DX-k-4 generator with $p = 2^{127} - c$, where x = 92233720368547

	$p = 2^{127} - c$ is non-Sophine-Germain			$p = 2^{127} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{63}$	c	min B	$B < 2^{63}$
101	66567	172	x75805	8023365	6	x75180
211	67231	112	x75770	11501829	607	x75101
307	281875	2541	x75294	12818949	151	x75329
401	682617	1076	x74716	10064781	1219	x74845
503	81777	1104	x75782	154659081	393	x75144
601	120295	728	x75780	142397385	1197	x75792
701	130395	2017	x75645	31187169	235	x73564
809	26941	1856	x75764	81710265	1295	x74053
907	314731	1011	x75405	26968581	8867	x74495
1009	876459	2616	x73707	2451789	584	x75460
1103	957505	360	x75541	253989021	1005	x70535
1201	360909	10629	x75086	29068281	257	x74416
1301	407797	5767	x74409	79595481	948	x72427
1409	1673445	5677	x74270	21557661	874	x74987
1511	6720469	1297	x73506	185276181	3245	x74080
1601	1829847	3106	x75175	175492881	1477	x73496
1709	1166731	852	x75523	31346721	8703	x67448
1801	1623471	13876	x75054	84200469	333	x74134
1901	1930005	1475	x70775	155499561	1125	x68375
2003	1827495	11995	x74132	9580245	7002	x73768

Table 20: List of B for DL-k generator with $p = 2^{127} - c$, where x = 92233720368547

	$p = 2^{127} - c$ is non-Sophine-Germain			$p = 2^{127} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{63}$	c	min B	$B < 2^{63}$
101	66567	126	x75753	8023365	74	x75739
211	67231	725	x75481	11501829	274	x75690
307	281875	286	x75567	12818949	174	x75502
401	682617	42	x75672	10064781	155	x75715
503	81777	1811	x75761	154659081	246	x75351
601	120295	1272	x75734	142397385	510	x74908
701	130395	532	x75053	31187169	550	x71962
809	26941	3671	x75681	81710265	1642	x75701
907	314731	4338	x75259	26968581	242	x74828
1009	876459	315	x74209	2451789	1327	x75652
1103	957505	1724	x74770	253989021	7580	x73463
1201	360909	1949	x72176	29068281	3627	x75385
1301	407797	11594	x75370	79595481	104	x72121
1409	1673445	922	x72325	21557661	727	x74593
1511	6720469	4602	x74460	185276181	1652	x74422
1601	1829847	1880	x74354	175492881	3378	x74753
1709	1166731	503	x65077	31346721	58	x74979
1801	1623471	2342	x73672	84200469	2271	x68777
1901	1930005	3333	x75597	155499561	15719	x66530
2003	1827495	217	x75004	9580245	5473	x72709

Table 21: List of B for DS-k generator with $p = 2^{127} - c$, where x = 92233720368547

	$p = 2^{127} - c$ is non-Sophine-Germain			$p = 2^{127} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{63}$	c	min B	$B < 2^{63}$
101	66567	37	x75600	8023365	21	x75762
211	67231	475	x75231	11501829	1292	x75384
307	281875	2403	x75598	12818949	695	x74557
401	682617	4884	x75737	10064781	45	x75084
503	81777	2893	x74993	154659081	1311	x75369
601	120295	1784	x75464	142397385	1333	x75715
701	130395	901	x72289	31187169	1492	x71626
809	26941	3881	x75437	81710265	2362	x71131
907	314731	3231	x74644	26968581	1545	x75136
1009	876459	517	x75100	2451789	13321	x74221
1103	957505	5194	x74864	253989021	1748	x75069
1201	360909	1535	x73584	29068281	659	x72722
1301	407797	3928	x75760	79595481	5362	x71963
1409	1673445	3225	x75648	21557661	107	x74291
1511	6720469	4440	x74509	185276181	6137	x75183
1601	1829847	1773	x74329	175492881	2490	x69286
1709	1166731	1271	x74653	31346721	2255	x68331
1801	1623471	948	x70542	84200469	2132	x72214
1901	1930005	393	x71314	155499561	996	x72441
2003	1827495	1484	x75532	9580245	3381	x71535

Table 22: List of B for DT-k generator with $p = 2^{127} - c$

	$p = 2^{127} - c$ is non-Sophine-Germain		$p = 2^{127} - c$ is Sophine-Germain	
k	w	min B	w	min B
101	66567	202	8023365	80
211	67231	58	11501829	68
307	281875	1333	12818949	268
401	682617	271	10064781	769
503	81777	1515	154659081	826
601	120295	2564	142397385	3394
701	130395	1087	31187169	415
809	26941	860	81710265	3586
907	314731	2490	26968581	1602
1009	876459	3778	2451789	269
1103	957505	2239	253989021	717
1201	360909	3889	29068281	755
1301	407797	2076	79595481	336
1409	1673445	3428	21557661	2532
1511	6720469	506	185276181	5424
1601	1829847	682	175492881	16986
1709	1166731	2897	31346721	777
1801	1623471	309	84200469	332
1901	1930005	165	155499561	97
2003	1827495	5624	9580245	2515

Table 23: List of B for DX-k-1 generator with $p = 2^{128} - c$, where x = 184467440737095

	$p = 2^{128} - c$ is non-Sophine-Germain			$p = 2^{128} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{64}$	c	min B	$B < 2^{64}$
101	122069	721	x51613	781733	264	x51370
211	162785	517	x51560	14363333	306	x50727
307	22637	706	x51033	67573457	402	x51010
401	1953	795	x50802	9780293	118	x51413
503	265845	183	x49811	25760477	837	x50953
601	1514669	746	x51463	29337077	32	x51119
701	198653	172	x51397	49288097	863	x51584
809	1691159	581	x51068	440234213	1417	x50326
907	793485	3396	x50183	31065533	970	x46471
1009	200399	9364	x51484	170478209	1810	x51101
1103	594789	3257	x50988	181533689	379	x49650
1201	412995	9277	x51563	81181637	1142	x49772
1301	480893	410	x51479	283176053	3315	x51436
1409	1027599	2940	x50782	587284637	1054	x51417
1511	678449	4951	x48533	83322269	466	x49588
1601	1716617	3122	x48208	68485229	668	x51550
1709	320613	2463	x51237	30085217	15315	x50136
1801	921513	5713	x48476	98538209	17837	x49388
1901	2374503	3437	x50398	93603137	145	x47271
2003	1201557	5344	x51018	142218077	1121	x51103

Table 24: List of B for DX-k-2 generator with $p = 2^{128} - c$, where x = 184467440737095

	$p = 2^{128} - c$ is non-Sophine-Germain			$p = 2^{128} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{64}$	c	min B	$B < 2^{64}$
101	122069	260	x51529	781733	331	x51579
211	162785	470	x51428	14363333	1131	x50662
307	22637	93	x51423	67573457	1103	x51560
401	1953	1637	x51452	9780293	558	x50844
503	265845	237	x51517	25760477	2511	x51566
601	1514669	1624	x51399	29337077	295	x51166
701	198653	524	x50755	49288097	3040	x48278
809	1691159	766	x50194	440234213	1185	x51459
907	793485	882	x48501	31065533	157	x48411
1009	200399	4075	x50443	170478209	4576	x51425
1103	594789	1051	x49160	181533689	895	x50537
1201	412995	5732	x49287	81181637	757	x45824
1301	480893	1995	x50115	283176053	4759	x49542
1409	1027599	7080	x51595	587284637	8154	x51201
1511	678449	2960	x51446	83322269	2729	x50441
1601	1716617	2158	x50844	68485229	2944	x50158
1709	320613	2553	x49805	30085217	2009	x50287
1801	921513	6118	x46874	98538209	2839	x51145
1901	2374503	293	x50302	93603137	2068	x47244
2003	1201557	2638	x50480	142218077	4745	x44195

Table 25: List of B for DX-k-3 generator with $p = 2^{128} - c$, where x = 184467440737095

	$p = 2^{128} - c$ is non-Sophine-Germain			$p = 2^{128} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{64}$	c	min B	$B < 2^{64}$
101	122069	440	x51504	781733	430	x51485
211	162785	1432	x51534	14363333	869	x51593
307	22637	114	x51173	67573457	255	x51314
401	1953	492	x51322	9780293	321	x50503
503	265845	2805	x51604	25760477	218	x50913
601	1514669	2275	x50950	29337077	873	x51595
701	198653	145	x50875	49288097	302	x50028
809	1691159	228	x51351	440234213	257	x51538
907	793485	5231	x50779	31065533	701	x51610
1009	200399	173	x49806	170478209	503	x48893
1103	594789	504	x51455	181533689	1960	x48697
1201	412995	5421	x50310	81181637	765	x49236
1301	480893	4846	x51499	283176053	5810	x50139
1409	1027599	1112	x51059	587284637	3460	x49479
1511	678449	1716	x50890	83322269	1120	x50388
1601	1716617	1818	x50142	68485229	6877	x50762
1709	320613	5871	x47311	30085217	3777	x51261
1801	921513	574	x50991	98538209	2279	x50988
1901	2374503	8262	x51399	93603137	170	x50653
2003	1201557	546	x50527	142218077	4671	x47887

Table 26: List of B for DX-k-4 generator with $p = 2^{128} - c$, where x = 184467440737095

	$p = 2^{128} - c$ is non-Sophine-Germain			$p = 2^{128} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{64}$	c	min B	$B < 2^{64}$
101	122069	644	x51345	781733	367	x51432
211	162785	232	x51339	14363333	113	x50957
307	22637	1281	x51439	67573457	387	x51127
401	1953	1481	x51549	9780293	1500	x51137
503	265845	963	x51125	25760477	719	x51548
601	1514669	3847	x51553	29337077	1189	x50695
701	198653	1217	x51065	49288097	859	x49708
809	1691159	3888	x51353	440234213	944	x50634
907	793485	4284	x51594	31065533	38	x47425
1009	200399	244	x51326	170478209	1360	x50881
1103	594789	10480	x48275	181533689	1999	x50453
1201	412995	8867	x49238	81181637	1761	x46375
1301	480893	3087	x49875	283176053	42	x50009
1409	1027599	4899	x50084	587284637	2324	x49996
1511	678449	1263	x50968	83322269	1062	x51364
1601	1716617	10650	x48857	68485229	2474	x38714
1709	320613	4875	x50110	30085217	1602	x49159
1801	921513	6890	x50041	98538209	2510	x50471
1901	2374503	11351	x49766	93603137	139	x47923
2003	1201557	572	x48610	142218077	15600	x47155

Table 27: List of B for DL-k generator with $p = 2^{128} - c$, where x = 184467440737095

	$p = 2^{128} - c$ is non-Sophine-Germain			$p = 2^{128} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{64}$	c	min B	$B < 2^{64}$
101	122069	62	x51415	781733	47	x51246
211	162785	323	x51590	14363333	228	x50074
307	22637	924	x51004	67573457	207	x51458
401	1953	1543	x50682	9780293	51	x50804
503	265845	707	x51168	25760477	1795	x50922
601	1514669	394	x51522	29337077	1337	x51072
701	198653	479	x51189	49288097	954	x48689
809	1691159	3291	x51254	440234213	766	x51420
907	793485	5010	x50944	31065533	294	x48552
1009	200399	3914	x51429	170478209	1909	x49287
1103	594789	9386	x50161	181533689	2380	x46579
1201	412995	2732	x49715	81181637	1123	x50985
1301	480893	1097	x51562	283176053	3171	x51374
1409	1027599	6167	x48124	587284637	1027	x49079
1511	678449	7009	x50997	83322269	2943	x46768
1601	1716617	2530	x51193	68485229	1116	x40215
1709	320613	4637	x49494	30085217	12027	x48844
1801	921513	1750	x51386	98538209	677	x49380
1901	2374503	10358	x50520	93603137	2551	x50247
2003	1201557	865	x50403	142218077	14164	x48540

Table 28: List of B for DS-k generator with $p = 2^{128} - c$, where x = 184467440737095

	$p = 2^{128} - c$ is non-Sophine-Germain			$p = 2^{128} - c$ is Sophine-Germain		
k	c	min B	$B < 2^{64}$	c	min B	$B < 2^{64}$
101	122069	28	x51603	781733	19	x51585
211	162785	462	x51249	14363333	412	x51061
307	22637	101	x51574	67573457	664	x51411
401	1953	1478	x50953	9780293	579	x50342
503	265845	843	x51393	25760477	1000	x51494
601	1514669	280	x51484	29337077	190	x51578
701	198653	2613	x51392	49288097	235	x49986
809	1691159	1499	x50075	440234213	124	x50723
907	793485	2088	x51495	31065533	80	x48415
1009	200399	2796	x50721	170478209	65	x51196
1103	594789	1198	x49667	181533689	3954	x46992
1201	412995	1567	x49286	81181637	2195	x46065
1301	480893	9001	x46862	283176053	533	x50334
1409	1027599	6091	x51441	587284637	8585	x46311
1511	678449	2220	x51221	83322269	1620	x46785
1601	1716617	4583	x51239	68485229	494	x50751
1709	320613	4423	x50797	30085217	2355	x48633
1801	921513	10152	x48619	98538209	969	x50622
1901	2374503	17759	x48729	93603137	5092	x50804
2003	1201557	2846	x51572	142218077	1696	x45568

Table 29: List of B for DT-k generator with $p = 2^{128} - c$

	$p = 2^{128} - c$ is non-Sophine-Germain		$p = 2^{128} - c$ is Sophine-Germain	
k	w	min B	w	min B
101	122069	450	781733	267
211	162785	725	14363333	246
307	22637	727	67573457	335
401	1953	3789	9780293	596
503	265845	1966	25760477	341
601	1514669	206	29337077	3079
701	198653	789	49288097	341
809	1691159	1695	440234213	1512
907	793485	1447	31065533	1141
1009	200399	6582	170478209	218
1103	594789	113	181533689	2668
1201	412995	3369	81181637	2817
1301	480893	960	283176053	2679
1409	1027599	586	587284637	448
1511	678449	2786	83322269	232
1601	1716617	4961	68485229	7207
1709	320613	937	30085217	4543
1801	921513	1630	98538209	1612
1901	2374503	6641	93603137	3435
2003	1201557	3293	142218077	3698

Table 30: Results of Crush test module on DL(63), DL(64), DL(127) and DL(128) generators

p-value	$> 1 - 10^{-15}$	$> 1 - 10^{-5}$	$> 1 - 10^{-4}$	$> 1 - 10^{-3}$	$< 10^{-3}$	$< 10^{-4}$	$< 10^{-5}$
DL(63), (5760 p-values)							
counts	0	0	0	5	6	1	0
proportion	0	0	0	0.000868	0.001042	0.000174	0
DL(64), (5760 p-values)							
counts	0	0	0	6	4	0	0
proportion	0	0	0	0.001042	0.000694	0	0
DL(127), (5760 p-values)							
counts	0	0	1	8	9	0	0
proportion	0	0	0.000174	0.001389	0.001563	0	0
DL(128), (5760 p-values)							
counts	0	0	1	8	10	2	0
proportion	0	0	0.000174	0.001389	0.001736	0.000347	0
64-bit DL: DL(63)+DL(64), (5760 $\times$ 2 p-values)							
counts	0	0	0	11	10	1	0
proportion	0	0	0	0.000955	0.000868	0.000087	0
128-bit DL: DL(127)+DL(128), (5760 $\times$ 2 p-values)							
counts	0	0	2	16	19	2	0
proportion	0	0	0.000174	0.001389	0.001649	0.000174	0
64-bit and 128-bit DL, (5760 $\times$ 4 p-values)							
counts	0	0	2	27	29	3	0
proportion	0	0	0.000087	0.001172	0.001259	0.000130	0

Table 31: Results of Crush test module on DS(63), DS(64), DS(127) and DS(128) generators

p-value	$> 1 - 10^{-15}$	$> 1 - 10^{-5}$	$> 1 - 10^{-4}$	$> 1 - 10^{-3}$	$< 10^{-3}$	$< 10^{-4}$	$< 10^{-5}$
DS(63), (5760 p-values)							
counts	0	0	0	8	3	0	0
proportion	0	0	0	0.001389	0.000521	0	0
DS(64), (5760 p-values)							
counts	0	0	0	2	7	1	0
proportion	0	0	0	0.000347	0.001215	0.000174	0
DS(127), (5760 p-values)							
counts	0	0	0	6	9	1	0
proportion	0	0	0	0.001042	0.001563	0.000174	0
DS(128), (5760 p-values)							
counts	0	0	1	4	8	1	0
proportion	0	0	0.000174	0.000694	0.001389	0.000174	0
64-bit DS: DS(63)+DS(64), (5760 $\times$ 2 p-values)							
counts	0	0	0	10	10	1	0
proportion	0	0	0	0.000868	0.000868	0.000087	0
128-bit DS: DS(127)+DS(128), (5760 $\times$ 2 p-values)							
counts	0	0	1	10	17	2	0
proportion	0	0	0.000087	0.000868	0.001476	0.000174	0
64-bit and 128-bit DS, (5760 $\times$ 4 p-values)							
counts	0	0	1	20	27	3	0
proportion	0	0	0.000043	0.000868	0.001172	0.000130	0

Table 32: Results of Crush test module on DT(63), DT(64), DT(127) and DT(128) generators

p-value	$> 1 - 10^{-15}$	$> 1 - 10^{-5}$	$> 1 - 10^{-4}$	$> 1 - 10^{-3}$	$< 10^{-3}$	$< 10^{-4}$	$< 10^{-5}$
DT(63), (5760 p-values)							
counts	0	0	1	4	4	0	0
proportion	0	0	0.000174	0.000694	0.000694	0	0
DT(64), (5760 p-values)							
counts	0	0	0	1	7	2	0
proportion	0	0	0	0.000174	0.001215	0.000347	0
DT(127), (5760 p-values)							
counts	0	0	0	8	6	0	0
proportion	0	0	0	0.001389	0.001042	0	0
DT(128), (5760 p-values)							
counts	0	1	1	5	5	1	0
proportion	0	0.000174	0.000174	0.000868	0.000868	0.000174	0
64-bit DT: DT(63)+DT(64), (5760 × 2 p-values)							
counts	0	0	1	5	11	2	0
proportion	0	0	0.000087	0.000434	0.000955	0.000174	0
128-bit DT: DT(127)+DT(128), (5760 × 2 p-values)							
counts	0	1	1	13	11	1	0
proportion	0	0.000087	0.000087	0.001128	0.000955	0.000087	0
64-bit and 128-bit DT, (5760 × 4 p-values)							
counts	0	1	2	18	22	3	0
proportion	0	0.000043	0.000087	0.000781	0.000955	0.000130	0