# EXTENDING RC4 TO CONSTRUCT SECURE RANDOM NUMBER GENERATORS

Lih-Yuan Deng[*]
Dale Bowman[*]
Ching-Chi Yang

Department of Mathematical Sciences
University of Memphis
373 Dunn Hall, Memphis, TN 38152, USA
{lihdeng, ddbowman, cyang3}@memphis.edu

Henry Horng-Shing Lu [†]

Institute of Statistics
National Yang Ming Chiao Tung University
1001 University Road, Hsinchu 30010, TAIWAN
hslu@stat.nctu.edu.tw

## ABSTRACT

We consider a general framework for constructing non-linear generators by adding a (32-bit or larger) pseudo-random number generator (PRNG) as a baseline generator to the basic RC4 design, in which an index-selection scheme similar to RC4 is used. We refer to the proposed design as the eRC (enhanced/extended RC4) design. We discuss several advantages of adding a good baseline generator to the RC4 design, including new updating schemes for the auxiliary table. We consider some popular PRNGs with the nice properties of high-dimensional equi-distribution, efficiency, long period, and portability as the baseline generator. We demonstrate that eRC generators are very efficient via extensive empirical testing on some eRC generators. We also show that eRC is flexible enough to choose minimal design parameters for eRC generators and yet the resulting eRC generators still pass stringent empirical tests, which makes them suitable for both software and hardware implementations.

**Keywords:** DX generator, eRC, eSTREAM ciphers, MRG, RC4

## 1 INTRODUCTION

Stream ciphers produce a sequence of ciphertext (in bits, bytes, words or larger sizes) sequentially by adding a variate from a key stream (of the same size) to a sequence of plaintext. The key stream sequence generated by a synchronous stream cipher depends only on the key, not on the plaintext or ciphertext. Therefore, it is clear that the security of a stream cipher completely depends on how “good" the generated key stream sequence is. If one could generate a sequence of truly-random variates for the key stream, then the resulting sequence of ciphertext would behave like a random sequence of variates. Consequently, it would be hard for attackers to decrypt from the ciphertext without knowing the values of the generated key stream. It is clear that no generators based on deterministic algorithms can claim to generate a truly-random sequence of uniformly distributed random variates. Therefore, it is essential to design “good” pseudo-random number generators (PRNGs) to generate the key stream for a stream cipher. There are several desirable properties for a “good” random number generator: (i) long period length so that the key stream sequence would not repeat

[*]This research was sponsored by the National Science Foundation under the award The Learner Data Institute (award #1934745). The opinions, findings, and results are solely the authors and do not reflect those of the funding agencies.

[†]This research of HHS Lu was partially supported by the Ministry of Science and Technology, Taiwan, under grant number 107-2118-M-009-006-MY3.

quickly, (ii) forward and backward unpredictability so that attackers could not predict the future variates from observing a number of past generated variates, (iii) great support of statistical/distributional properties for excellent empirical performances, (iv) great efficiency—fast generating speed and less memory space/gate count (mainly for hardware), and (v) simplicity, extendibility, and flexibility of the design.

Due to totally different objectives and concerns, two very different types of pseudo-random number generators (PRNGs) have been developed in the literature and used in practice for applications in computer simulation and computer security, respectively. For many computer simulation applications, it is common to use PRNGs to produce a sequence of variates that is very hard to distinguish from a sequence of (truly) random numbers. However, most of the proposed generators are linear generators and the linearity makes the generated sequences easily predictable from just a few past values. In contrast, the PRNGs developed for computer security applications put the major emphasis on the security issue.

For security applications, many pseudo-random bit generators (PRBGs), also known as deterministic random bit generators (DRBGs), have been developed. Barker and Kelsey (2012) published NIST's recommendation and specification on approved DRBG algorithms, including those based on hash functions, block cipher algorithms (AES and Triple DES), and a number theory problem that is expressed in elliptic curve technology. The popular RC4 stream cipher designed by Rivest is a byte-oriented stream cipher because it outputs only 8 bits (one byte) at each iteration. Another popular generator is the shrinking generator proposed by Coppersmith, Krawczyk, and Mansour (1994), which was shown to be vulnerable to multiple attacks including the correlation attack (Golić 2001). Nonetheless, the shrinking generator is an interesting example of using one LFSR (Linear Feedback Shift Register) acting as a (random) selector to choose or skip the output bit produced by another LFSR. Many other generators consider the strategy of applying some kind of non-linear filtering processes on multiple LFSRs; see, for example, Menezes, Van Oorschot, and Vanstone (1996) and Robshaw and Billet (2008).

An MRG (Multiple Recursive Generator) of order $k$ computes its next variate by a linear combination of the past $k$ variates, over a finite field of a large prime modulus $m$. MRG includes LFSR as a special case with modulus 2. MRGs have strong statistical justifications and excellent empirical performances. Another type of PRNG having great theoretical properties is MT19937, proposed by Matsumoto and Nishimura (1998), which is currently the most popular 32-bit PRNG and it has a period length of $2^{19937} - 1 \approx 10^{6001}$ with the dimensions of equi-distribution up to 623.

Neither MRGs nor LFSRs (including MT19937) are secure generators, because one can compute their next value from a sequence of observed past values by solving a system of linear recurrence equations. See, for example, Stinson (2006) for cryptanalysis of the LFSR and Lidl and Niederreiter (1994) for that of the MRG. An MRG of order $k$ requires up to $k$ past known variates (in correct order) to determine its next variate. Therefore, to make it harder for attackers to predict the next variate, Deng, Shiau, Lu, and Bowman (2018) proposed to scramble or hide (or both) the generated values by shuffling or mixing with other variates.

In this paper, we propose a general scheme, called eRC, to enhance RC4 by adding a baseline generator with great theoretical and empirical properties to the RC4 design. The proposed generator can benefit from both ends: (i) it can easily extend the RC4's 8-bit architecture to 32-bit (or larger) without sacrificing the simplicity of RC4; (ii) since the proposed generator uses the general table-shuffling method with random index-selection, so it is at least as secure as the popular RC4 (if not more); (iii) this RC4-like design can also help to break the linearity of a PRNG that already enjoys nice statistical properties; (iv) the addition of random selection of output transformation functions increases the difficulty of mounting attacks; and (v) the RC4-like design can still maintain or further improve the empirical performance of the baseline generator.

The rest of the paper is organized as follows. In Section 2, we describe and discuss some popular stream ciphers—RC4 and its popular variants. In Section 3, we first propose a general framework for the eRC design. We then consider a more specific eRC implementation, in which a similar scheme to that of RC4 is used for index selections. In Section 4, we discuss the design specifications recommended for various components used. In Section 5, we briefly review some basic properties of maximum-period MRGs and some specific efficient large-order MRGs. In Section 6, we give a formal discussion on the security properties of the proposed eRC generators. In Section 7, we consider the empirical performance of some eRC generators using an LCG as the baseline generator. By comparing their actual running times, we show that these eRC generators can be more efficient than RC4. In addition, we show the great empirical performance of these eRC generators even with very small order $k$ and a small table size. This would enable us to consider small eRC-based hardware ciphers that are highly competitive with existing eSTREAM hardware ciphers, which are mainly LFSR-based. See, Robshaw and Billet (2008). Finally, we provide a brief summary and concluding remarks in Section 8.

### 1.1 Notation

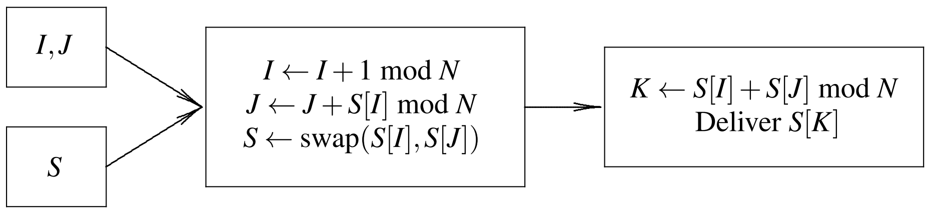
For two integers $x$ and $y$, we define $x\&y$ to be the number corresponding to the bit-wise “logical and” of their binary representations and $x \oplus y$ to be the bit-wise “exclusive or", and denote $(x+y) \bmod 2^{32}$ by $x \boxplus y$. Similarly, $x \gg d$ represents the logical right shift operation and $x \ll d$ the logical left shift operation of $d$ bits in the binary representation of $x$. For an integer $m$, we denote $\mathbb{Z}_m = \{0,1,\ldots,m-1\}$.

## 2 RC4 AND RELATED METHODS

The idea of using auxiliary tables has been quite successfully used in the area of computer security. Two popular stream ciphers, RC4 and HC-128, both have a similar feature of producing output with some auxiliary table(s) that is (are) updated at each iteration.

### 2.1 RC4

RC4 is a well-known byte-oriented software-designed stream cipher. During the initialization stage, the $S$-table is a shuffled table (based on the user-input key values) of $S[i] = i$, $i = 0,\ldots,255 (= 2^8 - 1)$. The generation stage of RC4 can be shown as in the following diagram:


$I, J$
$S$
$I \leftarrow I + 1 \bmod N$
$J \leftarrow J + S[I] \bmod N$
$S \leftarrow \text{swap}(S[I], S[J])$
$K \leftarrow S[I] + S[J] \bmod N$
Deliver $S[K]$

For RC4, it is common to set $N = 2^8$.

The security strength of RC4 relies on the unpredictability of the hidden “internal states” of $I$, $J$, and the $S$-table. The values for these “internal states” are “moving targets” because of the continual updates on $I$, $J$, and “random” permutation of the $S$-table (via the swapping of $S[I]$ and $S[J]$) at each iteration. Several problems with RC4 have been identified, including correlations between consecutive bytes (Fluhrer and

McGrew 2000), classes of weak keys (Roos 1995, Wagner 1995), and bias in the second byte of RC4 (Mantin and Shamir 2001). Mironov (2002) provides descriptions of attacks on RC4.

### 2.2 Previous (failed) attempts to extend RC4

Paul and Maitra (2011) (Chapter 9) gave an extensive review and analysis of RC4 and its variants. In particular, they recommended RC4$^+$, proposed by Maitra and Paul (2008). According to Paul and Maitra (2011), KSA$^+$ (Key Schedule Algorithm) of RC4$^+$ can circumvent the weakness of standard RC4 KSA and PRGA$^+$ (PRNG Algorithm) can better protect the information about the S-table by masking the output bytes. On the negative side, its running time is about 2.94 times (KSA$^+$) and 1.70 times (PRGA$^+$) of RC4 counterparts. In addition, RC4$^+$ is still a byte-oriented stream cipher.

We will only mention some other RC4 variants that are related to our proposed eRC. To expand the 8-bit architecture of RC4 to 32-bit or 64-bit, Nawaz, Gupta, and Gong (2005) and Gong, Gupta, Hell, and Nawaz (2005) proposed RC(8, 32) and RC(8, 64), respectively. Their main idea is to fill the initial S-table with 32-bit (or 64-bit) numbers during their KSA with a very slight change in the PRGA. However, Wu (2005) showed that there is a simple distinguished attack on RC(8, 32) using only 100 key stream words. While the motivation of extending RC4 to a 32-bit stream cipher is useful, we believe that lack of continual injection of “random source” into the S-table is a major cause for the failure of their approach.

## 3 eRC: A GENERAL SCHEME TO ENHANCE/EXTEND RC4

RC4, its variants, and HC-128 share one common “weakness”: they lack a “good” and “random” mechanism to drive the table-index selection in the algorithm. For RC4 and some of its variants, user keys are used to “scramble” the initial S-table, which may yield a “poor” permutation (randomization) of the initial S-table, especially when a weak KSA is used. Likewise, HC-128 requires a long iteration process to ensure that the initial tables are properly “randomized”. The S-table of RC4 is always a “shuffle” of $0,1,\ldots,255(=2^8-1)$. In contrast, the P/Q tables of HC-128 have less “justification” for their uniformity. Once the initial tables (S, P/Q) are fixed, the “randomness” of the output is produced through a sequence of complex transformations and combinations, without a continual “injection” of good external generators.

### 3.1 Adding a good external PRNG as baseline generator

As mentioned earlier, to enhance RC4, we propose adding an efficient PRNG in the eRC design. Consider a general PRNG that generates the sequence $\{X_i, i \geq 0\}$ by

$$X_i = f(X_{i-1},\ldots,X_{i-k}) \bmod m, \quad i \geq 0, \tag{1}$$

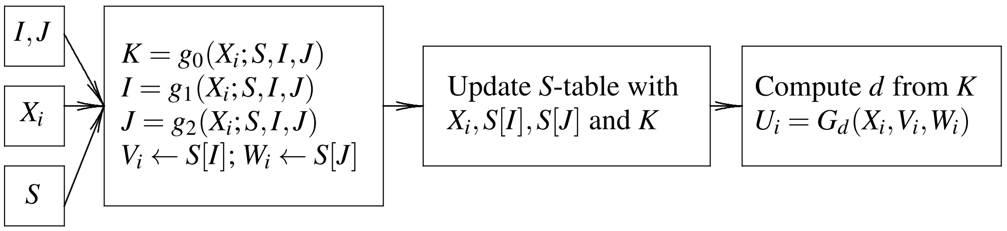
with a “suitably chosen” integer-valued function, $f$, of the most recent $k$ integers in the past and $k$ initial seeds, $X_{-k}, X_{-(k-1)}, \ldots, X_{-1}$. We shall consider a specific choice of the baseline generator later.

Next, we propose a general framework that can be viewed as an “*enhancement*” for RC4. The key idea is to add an external generator that continuously feeds/updates the S-table to ensure better uniformity and randomness.

### 3.2 General framework to extend RC4

Like RC4, let $S$ be a table of size $2^C$; but unlike RC4, each entry of the $S$-table holds a value generated by the baseline generator, which can be of any size, say, 32-bit, 64-bit, or even larger.

Consider the general framework of eRC as shown in the following diagram (at the *i*-th iteration):


$I,J$
$X_i$
$S$
$K = g_0(X_i;S,I,J)$
$I = g_1(X_i;S,I,J)$
$J = g_2(X_i;S,I,J)$
$V_i \leftarrow S[I]; W_i \leftarrow S[J]$
Update $S$-table with $X_i, S[I], S[J]$ and $K$
Compute $d$ from $K$
$U_i = G_d(X_i, V_i, W_i)$

We briefly describe each of the blocks in the diagram:

1. To begin the generation step, we need to initialize the internal states of RNG-X (the baseline generator), the contents of the $S$-table, and the initial values of indices $I$ and $J$. All of these need to be filled using the private user-keys and initialization values (IVs).
2. Compute the new values of $I, J$, and $K$ based on the current values of $I, J$, the $S$-table, and $X_i$. The purpose of $I$, $J$, and $K$ in eRC are similar to that of RC4:
   (a) $K = g_0(X_i;S,I,J)$ will produce one random number using a combination of the current variate ($X_i$) and previous $S[I]$ and $S[J]$ at the beginning of the iteration. Note that $K$ is computed from feedbacks (i.e., the $S[I]$ and $S[J]$ from the previous iteration).
   (b) $I = g_1(X_i;S,I,J)$ and $J = g_2(X_i;S,I,J)$ are suitably chosen "$C$-bit generators" producing the selection indices used for choosing entries from the $S$-table.
3. Different bits (say, $a(K)$ and $b(K)$) of $K$ will be used to (a) determine the location to store $X_i$ in the table $S[\cdot]$ (using $a(K)$) and/or (b) randomly select one mixing function from a list of possible $2^D$ functions (using $b(K)$):
   (a) Use $a(K)$ to update the $S$-table with some "random shuffling" of the $S$-table and "random assignment" of $X_i$ into the $S$-table. In RC4, the $S$-table (of size $2^C$) is "shuffled" by swapping $S[I]$ and $S[J]$. In our design, in addition to "shuffling", we need a scheme to randomly update $S[I]$ or $S[J]$.
   (b) Use $b(K)$ to perform a random selection from a list of $2^D$ output transformations, say, $G_d(X_i, V_i, W_i), d = 0, 1, \ldots, 2^D - 1$. This new feature of random selection of the output function can greatly increase the difficulty of mounting attacks.
   (c) It is easy to show that if random variable $K$ is uniformly distributed over $0, 1, \ldots, 2^w - 1$, where $w$ is a positive integer, then the two random binary variables $a(K)$ and $b(K)$ are uniformly distributed; and they are statistically independent of each other. Therefore, we can make "independent" decisions based on different parts of $K$.

For each of the stages listed above, many choices for these procedures can be considered and we will provide our recommendations later. The main requirement is that the chosen procedures would enhance the desired properties of "randomness", efficiency, and unpredictability.

Next, we recommend some specific choices for the new eRC design. We then finalize a specific design with efficiency and security considerations.

### 3.3 eRC-E and eRC-S: two specific eRC ciphers

There are many choices that can be used for table index selection to produce $I$, $J$, and $K$ for the general eRC design. With simplicity and efficiency in mind, we consider specific choices that have strong resemblance to the popular RC4 cipher.

### 3.3.1 Two proposed procedures to update *S*-table in eRC design

We proposed two procedures for eRC design with (1) efficiency or (2) complexity consideration.

For the design with the first procedure to update the *S*-table, we denote it by **eRC(RNG-X; E,** $2^C$**,** $2^D$), where "E" stands for efficiency. It can be described in the diagram below:

$$\begin{tikzcd}
\boxed{I,J} \arrow[dr] & & & \\
\boxed{X_i} \arrow[r] & \boxed{\begin{array}{l} K=(S[I]+S[J])\gg A \\ I=I+1 \bmod 2^C \\ J=J+S[I] \bmod 2^C \\ V_i \leftarrow S[I];\, W_i \leftarrow S[J] \end{array}} \arrow[r] & \boxed{\begin{array}{l} S[I] \leftarrow S[J] \\ S[J] \leftarrow X_i \end{array}} \arrow[r] & \boxed{\begin{array}{l} \text{Compute } d=b(K) \\ U_i=G_d(X_i,V_i,W_i) \end{array}} \\
\boxed{S} \arrow[ur] & & &
\end{tikzcd}$$

For the design with the second procedure to update the *S*-table, we use **eRC(RNG-X; S,** $2^C$**,** $2^D$**)** to denote it, where "S" stands for symmetry (or security). It can be described in the diagram below:

$$\begin{tikzcd}
\boxed{I,J} \arrow[dr] & & & \\
\boxed{X_i} \arrow[r] & \boxed{\begin{array}{l} K=(S[I]+S[J])\gg A \\ I=I+1 \bmod 2^C \\ J=J+S[I] \bmod 2^C \\ V_i \leftarrow S[I];\, W_i \leftarrow S[J] \end{array}} \arrow[r] & \boxed{\begin{array}{l} \text{if } a(K)=0 \text{ then} \\ \quad S[J] \leftarrow S[I] \\ \quad S[I] \leftarrow X_i \\ \text{else} \\ \quad S[I] \leftarrow S[J] \\ \quad S[J] \leftarrow X_i \end{array}} \arrow[r] & \boxed{\begin{array}{l} \text{Compute } d=b(K) \\ U_i=G_d(X_i,V_i,W_i) \end{array}} \\
\boxed{S} \arrow[ur] & & &
\end{tikzcd}$$

### 3.4 Choice of output functions

We define output transformations via combinations of several bit rotation functions as defined below:

1. The bit rotation function for a 32-bit variate $x$ is defined as

$$f_r(x) = (x \gg r) \oplus (x \ll (32-r)),$$

   where $r$ is the size of bit rotation. The bit rotation function is similar to the ones used in Wu (2004).
2. For the list of $2^D$ output transformation functions, we consider

$$G_d(x_i, v_i, w_i) = f_{s_{d1}}(x_i) \oplus f_{s_{d2}}(v_i) \oplus f_{s_{d3}}(w_i),$$

   with $0 < s_{dj} < 31$ for $j = 1,2,3$ chosen with different values for the bit rotation sizes, $d = 0,1,\ldots,2^D-1$. For $D = 2$, we can choose from the four possible output functions below:
   (a) $G_0(X_i,V_i,W_i) = f_{s_{01}}(X_i) \boxplus f_{s_{02}}(V_i) \boxplus f_{s_{03}}(W_i)$.
   (b) $G_1(X_i,V_i,W_i) = f_{s_{11}}(Y_i) \boxplus f_{s_{12}}(V_i) \boxplus f_{s_{13}}(W_i)$.
   (c) $G_2(X_i,V_i,W_i) = f_{s_{21}}(X_i) \boxplus f_{s_{22}}(Y_i) \boxplus f_{s_{23}}(V_i)$.
   (d) $G_3(X_i,V_i,W_i) = f_{s_{31}}(X_i) \boxplus f_{s_{32}}(Y_i) \boxplus f_{s_{33}}(W_i)$.
   Here we assume that different shifts $1 \le s_{dj} \le 31$ ($d = 0,1,\ldots,2^D-1$, $j = 1,2,3$) are used; therefore there are plenty of possible different configurations.

The list size, $2^D$, does not have to be large and we set $D \le 2$. When $D = 0$, with a single output function, it is the most efficient but may lack a degree of complexity.

## 4 DESIGN SPECIFICATIONS OF eRC

### 4.1 Choices of indexing functions

As mentioned earlier, there are numerous ways to choose the index values for $I$, $J$, and $K$. In this paper, we will only consider two configurations that are similar to the ones used by RC4—eRC(RNG-X; E, $2^C$, $2^D$) and eRC(RNG-X; S, $2^C$, $2^D$)—as described earlier.

### 4.2 Choices of table size $2^C$ and number of functions $2^D$

For software implementation, there is no practical limitation on the size of the $S$-table, $2^C$. Larger table sizes tend to yield better performances on both of the statistical/distributional property and security property. For hardware implementation, however, the table size $2^C$ can be a major determining factor in the hardware cost. To compete with the existing leading hardware ciphers (e.g., eSTREAM finalists Trivium and Grain), we need to consider a much smaller $C$, say, $C = 1$ or $C = 2$, which corresponds to the table size of 2 or 4. With a chosen good PRNG (to be discussed later), we will show that the proposed eRC generator can still pass stringent empirical tests even with table size of 2 or 4.

Choice of $D$ may depend on the conflicting requirements between generating efficiency and security. Using only a single output function would enhance the generating efficiency in software implementation as well as reduce the hardware cost in hardware implementation. On the other hand, using multiple branching outputs would greatly increase the difficulty of the cryptanalysis by attackers. Since both $D = 0$ and $D = 1$ can be implemented efficiently in software, we will consider $D \leq 1$ in this paper.

### 4.3 Key/IV procedure resistant to chosen IV attacks

For cryptographic applications, Stallings (2010) gave three requirements on the PRNG—randomness (passing stringent statistical tests), unpredictability (forward and backward), and seed requirements (large key space for possible seed (random) assignment). With a proper choice of the baseline generator, our proposed design allows a large enough space for (secret) seed assignment. For a PRNG to be suitable for computer simulation applications, it often requires great empirical performances with any chosen starting seeds. On the other hand, a random secret seed assignment is important for a PRNG to be useful in cryptographic applications. Thus one should follow the rigorous NIST guideline as specified in Barker and Kelsey (2012) for proper seed initialization.

Most stream ciphers load the values of the user-key and IV into the internal states of the baseline generators and then use a number of "burn-in" iterations. See the discussion in Englund, Johansson, and Turan (2007) for Grain-128 and Trivium. Typically, a large burn-in period is needed to ensure that there is no discernible relationship between the output variates and the user-key or IV.

To complete the description of our specific eRC, we need to specify the choice of the baseline generator. In principle, the eRC scheme can work with any type of PRNGs. After a brief general discussion, we recommend a specific class of PRNGs to serve as the baseline generator in the eRC scheme.

## 5 PSEUDO-RANDOM NUMBER GENERATORS

When the function $f$ in (1) is a linear function, it defines a popular MRG of order $k$ as

$$X_i = (\alpha_1 X_{i-1} + \ldots + \alpha_k X_{i-k}) \bmod m, \quad i \geq 0, \tag{2}$$

where $\alpha_1, \ldots, \alpha_k$ are not-all-zero integers in $\mathbb{Z}_m$ and $\alpha_k \neq 0$. We denote it by MRG($k$;$m$). With proper choices of $\alpha_i$'s, the maximum period, $m^k - 1$, can be achieved; see Knuth (1998). When $k = 1$, the MRG in (2) is reduced to an LCG with the maximum period length, $m - 1$.

### 5.1 Properties of maximum-period MRGs and efficient DX-$k$ generators

The maximum period of an MRG as in (2) is $m^k - 1$. In addition to its long period, a maximum-period MRG enjoys the nice property of equi-distribution up to $k$ dimensions. Lidl and Niederreiter (1994), Theorem 7.43) showed that every $t$-tuple ($1 \leq t \leq k$) of integers between 0 and $m-1$ appears exactly the same number of times (i.e., $m^{k-t}$) over its entire period, $m^k - 1$, with the exception of the all-zero tuple that appears one time less (i.e., $m^{k-t} - 1$). The equi-distribution property is a desirable feature for a good random number generator.

Deng and Lin (2000) proposed a fast MRG (FMRG) using this principle that is almost as efficient as a classic LCG. Deng and Xu (2003) introduced the DX-$k$-$s$ generators, a special class of MRGs in (2) with $s$ nonzero coefficients. In particular, DX-$k$-1 ($\alpha_1 = 1, \alpha_k = B$) and DX-$k$-2 ($\alpha_1 = \alpha_k = B$) are defined as

$$X_i = X_{i-1} + BX_{i-k} \bmod m, \quad i \geq 0, \tag{3}$$

and

$$X_i = B(X_{i-1} + X_{i-k}) \bmod m, \quad i \geq 0, \tag{4}$$

respectively.

### 5.2 Recommended baseline generator for eRC

As mentioned earlier, there is a great flexibility in choosing the baseline generator. For example, because of the popularity and generating efficiency, we can choose the popular MT19937 as the baseline generator for the software implementation. However, it is easy to switch to another PRNG like a DX-$k$ generator with large order $k$, if higher order of the equi-distribution property or a much longer period is desired.

## 6 COMPARISON AND SECURITY ANALYSIS

To break eRC, in addition to these values, attackers need to know the contents of the $k$ "internal states" of the baseline generator as well. The recovery process is considered difficult because (i) the observable output is a transformation masking (hiding) the secret "internal states"; (ii) the parameter space for the $k$ hidden internal states can be huge (up to 20897 words), when a large-order MRG (or MT19937) is used as the baseline generator; (iii) all hidden states are "moving targets" due to continual updating; and (iv) if a large-order maximum-period MRG is used as the baseline generator, the equi-distribution property can assure equal likelihood for any possible (huge) current "internal states"; hence, naive brute-force attacks cannot succeed with today's technology.

### 6.1 Period length of eRC

Like RC4, the exact period length of the eRC(RNG-X; $g$, $2^C$, $2^D$) generator is unknown. The major determining factor is the period length of the baseline generator (RNG-X), the $S$-table size, $2^C$ and the number of output transformations, $2^D$. Conservatively speaking, the period length of the eRC generator is at least as large as the period length of the baseline generator used. Since the eRC generator uses "feedback" ($S[I]$ and $S[J]$) as part of its next input, the period length can be much larger with a larger $S$-table. See, for example, Bays and Durham (1976).

### 6.2 Algebraic attack against eRC

Most applications of algebraic attacks on LFSR-based stream ciphers are effective because of the simple linear generating equations and binary output of the ciphers. Wong, Carter, and Dawson (2010) considered the problem of extending algebraic analysis to non-LFSR-based stream ciphers such as RC4. According to the study in Wong, Carter, and Dawson (2010), RC4 is somewhat resistant to powerful algebraic attacks. We believe that eRC generators are also less susceptible to algebraic attacks because (i) they are MRG-based, not LFSR-based; (ii) their output is word-based (32 bits or higher, not just 8 bits like RC4) at each iteration; (iii) they are much more complicated than RC4; and (iv) they use a "self" (with feedbacks) shuffling scheme and a nonlinear output transformation.

### 6.3 Enhance security with multiple output transformations

The choice of the list size $2^D$ is flexible, depending on the security and efficiency requirements. A large number of choices for the output function in general can improve the complexity of the proposed cipher and hence increase the difficulty of algebraic attacks. When $D = 0$, it is reduced to a simpler design with a fixed non-linear transformation function. When $D \geq 1$, there are $2^D$ possible choices of the output functions that can be used at each iteration.

Adding multiple branching output functions is one of the major enhancements. It can greatly increase the complexity of the design and (most likely) enhance the security without losing too much computational efficiency. The feature of random selection from multiple output functions should allow the proposed generators to be more resistant to the powerful algebraic attack as proposed by Courtois (2002) and Courtois and Meier (2003).

## 7 EMPIRICAL EVALUATION OF eRC GENERATORS

To compare the actual computing efficiency with RC4, we select several PRNGs to serve as the baseline generators for eRC, including five LCGs and five for each small-order DX-$k$-1 ($k = 2, 3, 4, 5$) generators. For efficiency consideration, we use the multiplier $B = 2^r \pm 2^w$ and the modulus $m = 2^{31} - 1$.

### 7.1 Timing comparison with other fast stream ciphers

Matsumoto, Saito, Nishimura, and Hagita (2007) and Matsumoto, Saito, Nishimura, and Hagita (2008) proposed CryptMT version 3 (CryptMT3) to implement the original design on some modern CPUs (e.g., Intel Core 2 Duo) with SIMD (single-instruction-multiple-data) operations on 128-bit registers. They reported that CryptMT3 is about 1.8 times faster than the original CryptMT and can be considered as one of the fastest stream ciphers.

To show the great efficiency of the proposed eRC design, we compare various eRC generators with CryptMT and RC4. In addition to MT19937, we also choose some previously described small order DX-$k$-1 generators as the baseline generator for eRC. According to our study, the table size $2^C$ and the order of the RNG-X are not major factors for generating efficiency. We compute the average running time (over all generators considered) for generating $10^9$ variates (words, not bytes). The average computing time required are: 3.15s for eRC(DX; E, $2^C$, 1), 3.66s for eRC(MT19937; E, $2^C$, 1), 8.92s for CryptMT, and 5.82s for RC4 on computers in the university computing clusters. The actual running time is highly hardware (and compiler) dependent and it also depends on the generation speed of the baseline generator used. We can see that eRC(DX; E, $2^C$, 1) and eRC(MT19937; E, $2^C$, 1) are about 2.84 times and 2.44 times faster than CryptMT, respectively. Therefore both eRCs are faster than fast CryptMT3. Moreover, our eRC generators do not require the use of 128-bit registers with SIMD operations.

### 7.2 Empirical test on small order eRC generators

We have tested extensively eRC generators with a very-small-order DX-$k$ generator and with a small table size, $2^C$, can still pass stringent empirical tests like TestU01 developed by L'Ecuyer and Simard (2007). Due to space limitation, we skip the results here. Our empirical results show that we can indeed also turn a very bad PRNG (for example, LCG) into a good one through our eRC scheme.

## 8 CONCLUDING REMARKS

In this paper, we find a general class of eRC generators using certain "good" PRNGs, such as small-order DX generators, that are as efficient as the popular LFSRs. Such eRC generators can be useful for a new hardware-based cipher with the following characteristics: (i) it uses an $S$-table like RC4 but with a much smaller size, say, of size 2; (ii) it uses a simple and efficient PRNG (not LFSR) as the baseline random number generator to continually feed and update the shuffle table ($S[\cdot]$); (iii) the baseline generator used has a small number of internal states and will produce a sequence of successive pseudo random numbers with a known large period; (iv) compared to the software implementation, it is more efficient for the hardware implementation on (a) a more complex (symmetric and random) updating scheme for the $S$-table such as eRC-S design and (b) the multiple branching step for randomly selecting an output function from $2^D$ output functions with $D \geq 1$.

Because of the flexibility in the new eRC design, we believe that we can achieve the goal of finding "universal generators" that are suitable for both software and hardware implementations and also for both computer simulation and computer security applications. In addition, the "universal generators" can be implemented easily and efficiently either by software programming or in hardware design. Design flexibility enables users to choose the smallest, fastest cryptosystems that provide an acceptable security level.

## REFERENCES

Barker, E., and J. Kelsey. 2012, jun. "Recommendation for random number generation using deterministic random bit generators (revised)". *NIST Special publication* vol. 800 (March), pp. 90.

Bays, C., and S. D. Durham. 1976, March. "Improving a Poor Random Number Generator". *ACM Transactions on Mathematical Software (TOMS)* vol. 2 (1), pp. 59–64.

Coppersmith, D., H. Krawczyk, and Y. Mansour. 1994. "The shrinking generator". *Lecture Notes in Computer Science* vol. 773 LNCS, pp. 22–39.

Courtois, N. T. 2002. "Higher Order Correlation Attacks, XL Algorithm and Cryptanalysis of Toyocrypt". *ICISC 2002* vol. LNCS 2587, pp. 182–199.

Courtois, N. T., and W. Meier. 2003. "Algebraic attacks on stream ciphers with linear feedback". *Advances in Cryptology – EUROCRYPT* vol. LNCS 2003, pp. 345—-359.

Deng, L.-Y., and D. K. Lin. 2000. "Random number generation for the new century". *The American Statistician* vol. 54 (2), pp. 145–150.

Deng, L.-Y., J.-J. H. Shiau, H. H.-S. Lu, and D. Bowman. 2018, July. "Secure and Fast Encryption (SAFE) with Classical Random Number Generators". *ACM Trans. Math. Softw.* vol. 44 (4), pp. 17.

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

Englund, H., T. Johansson, and M. S. Turan. 2007. "A framework for chosen IV statistical analysis of stream ciphers". In *International Conference on Cryptology in India*, pp. 268–281. Springer.

Fluhrer, S., and D. McGrew. 2000. "Statistical analysis of the alleged RC4 keystream generator. In proceedings Fast Software Encryption 2000". *Lecture Notes in Computer Science* vol. 1978, pp. 19–30.

Golić, J. D. 2001. "Correlation analysis of the shrinking generator". In *Annual International Cryptology Conference*, pp. 440–457. Springer.

Gong, G., K. C. Gupta, M. Hell, and Y. Nawaz. 2005. "Towards a general RC4-like Keystream Generator". In *CISC 2005, Lecture Notes in Computer Science*, Volume 3822, pp. 162–174. Springer.

Knuth, D. E. 1998. *The art of computer programming, vol 2: seminumerical algorithms*. 3 ed. Addison-Wesley.

L'Ecuyer, P., and R. Simard. 2007. "TestU01: A C library for empirical testing of random number generators". *ACM Transactions on Mathematical Software (TOMS)* vol. 33 (4), pp. 22.

Lidl, R., and H. Niederreiter. 1994. *Introduction to finite fields and their applications*. Cambridge University Press.

Maitra, S., and G. Paul. 2008. "Analysis of RC4 and Proposal of Additional Layers for Better Security Margin". *INDOCRYPT 2008, Lecture Notes in Computer Science* vol. 5365, pp. 27–39.

Mantin, I., and A. Shamir. 2001. "A practical attack on broadcast RC4". In *International workshop on fast software encryption*, pp. 152–164. Springer.

Matsumoto, M., and T. Nishimura. 1998. "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator". *ACM Transactions on Modeling and Computer Simulation (TOMACS)* vol. 8 (1), pp. 3–30.

Matsumoto, M., M. Saito, T. Nishimura, and M. Hagita. 2007. "A fast stream cipher with huge state space and quasigroup filter for software, selected areas in cryptography". *Lecture Notes in Computer Science (LNCS)* vol. 4876, pp. 246–263.

Matsumoto, M., M. Saito, T. Nishimura, and M. Hagita. 2008. "CryptMT3 stream cipher, New Stream Cipher Designs". *Lecture Notes in Computer Science (LNCS)* vol. 4986, pp. 7–19.

Menezes, A. J., P. C. Van Oorschot, and S. A. Vanstone. 1996. *Handbook of applied cryptography*. CRC press.

Mironov, I. 2002. "(Not so) random shuffles of RC4". In *Annual International Cryptology Conference*, pp. 304–319. Springer.

Nawaz, Y., K. C. Gupta, and G. Gong. 2005. "A 32-bit RC4-like Keystream Generator.". *IACR Cryptol. ePrint Arch.* vol. 2005, pp. 175.

Paul, G., and S. Maitra. 2011. *RC4 stream cipher and its variants*. CRC Press.

Robshaw, M., and O. Billet. 2008. *New Stream Cipher Designs – The eSTREAM Finalists*, Volume LNCS 4986. Springer-Verlag.

Roos, A. 1995. "Class of weak keys in the RC4 stream cipher".

Stallings, W. 2010. *Cryptography and Network Security: Principles and Practice*. 5 ed. Prentice Hall.

Stinson, D. 2006. *Cryptography: theory and practice*. 3 ed. Chapman and Hall/CRC Press.

Wagner, D 1995. "My RC4 weak keys". Post in sci. crypt.

Wong, K., G. Carter, and E. Dawson. 2010. "An analysis of the RC4 family of stream ciphers against algebraic attacks". In *Information Security 2010*, edited by C. Boyd and W. Susilo, pp. 73–80. Australia, Australian Computer Society.

Wu, H. 2004. "A new stream cipher HC-256". In *Proceedings of FSE 2004, Lecture Notes in Computer Science*, edited by B. Roy and W. Meier, Volume 3017, pp. 226–244. Springer.

Wu, H. 2005. "Cryptanalysis of a 32-bit RC4-like Stream Cipher.". *IACR Cryptol. ePrint Arch.* vol. 2005, pp. 219.

## AUTHOR BIOGRAPHIES

**LIH-YUAN DENG** received the B.S. and M.S. degree in Mathematics from National Taiwan University, Taiwan in 1975 and 1977, respectively. He also received M.S. and Ph.D. degrees in Computer Science and Statistics from University of Wisconsin-Madison, USA in 1982 and 1984, respectively. He is currently Professor in the Department of Mathematical Sciences, University of Memphis, Memphis, Tennessee, USA. His active research work is mainly in the area of "design of random number generators" for computer simulation and computer security applications. His email address is lihdeng@memphis.edu.

**DALE BOWMAN** received a B.S. in computer science in 1984 from Middle Tennessee State University and a PhD in Statistics from the University of Memphis in 1993. She is currently an Associate Professor in the Department of Mathematical Sciences at the University of Memphis , Memphis TN, USA. Her research work is currently in exchangeability and random number generators. Her email address is ddbowman@memphis.edu.

**CHING-CHI YANG** received a Ph.D. in Statistics from the Pennsylvania State University in 2019. He is currently an Assistant Professor in the Department of Mathematical Sciences at the University of Memphis, Memphis, TN, USA. His primary interests focus on statistical learning and random number generators. His email address is cyang3@memphis.edu.

**HENRY HORNG-SHING LU** received his Ph.D. and M.S. degrees in Statistics from Cornell University, USA, in 1994 and 1990, respectively, and his B.S. degree in electric engineering from National Taiwan University, Taiwan, in 1986. He is a Professor in the Institute of Statistics, National Chiao Tung University that is merged to National Yang Ming Chiao Tung University, Taiwan. He has served as the Vice President of Academic Affairs in National Chiao Tung University. He is an elected member of International Statistical Institute (ISI) and Principal Fellow of the Higher Education Academy (PFHEA). His research interests include statistics, image science, bioinformatics, data science, and machine learning. His email address is hslu@stat.nctu.edu.tw