Usage of LCG/CLCG numbers for electronic gambling applications

Anders Knutsson

Simovits Consulting, Wenner-Gren Center,
Sveavägen 166, 113 46 Stockholm, Sweden
anders.knutsson@simovits.com

Abstract. Several attacks and weaknesses against the Linear Congruential Generator (LCG) have been shown in literature, but not descriptions how these attacks successfully can be used in practice against gaming applications that use LCG values. This paper presents results from implementations of different attack methods against an internet casino roulette game using an LCG. It is shown that the casino cannot prevent attacks by decreasing the interval of reseeding the generator within reasonable limits. The attack methods discussed are the attack by Frieze et al, a brute force attack and an attack called the chosen gap method, the latter introduced in this paper.

1. Introduction

Pseudo random numbers are often used instead of real random values when speed to create numbers is important as is the case for electronic gambling applications. One popular pseudo random generator is the Linear Congruential Generator (LCG). This generator is suggested to be used when implementing casino games by the Swedish lottery authority [7].

The LCG has been shown to be unsafe by several publications. Boyar [1] and Krawczyk [4] showed that a generator value can be predicted given a polynomial bounded number of earlier values without the knowledge of the LCG constants a, b and m. Frieze, Hastad, Kannan, Lagarias and Shamir [2] showed that truncated values generated by an LCG, with known a and m, may in many cases be restored when given the most significant bits, the least significant bits or any other window of bits. The probability of a successful reconstruction increases with more knowledge, as more bits of each value or more values. Stern [8] showed that the knowledge of a and m could be omitted, as long as more knowledge in the form of bits of values and number of values can be had.

This paper will show the results of an implementation of the work of Frieze et al. in a casino environment and show that these attacks cannot be prevented by decreasing the interval of reseeding, as long as the casino demands that many more bits in the form of pseudo randomness shall be produced than bits of real randomness is given to the generator. Comparisons are made to a brute force attack and a method called the chosen gap attack.

2. Prerequisites

The linear congruence generator, LCG produces a sequence X according to the mathematical function (1). Let X_0 be the seed, i.e. the first value in the sequence and a real random integer value. The LCG has three positive integers a, b and m.

$$X_i = (aX_{i-1} + b) \bmod m \tag{1}$$

The sequence X has values in the interval [0, m-1] and the period will at a maximum be m. The LCG passes the usual statistical tests when initialized and used properly. How the integers a, b and m should be chosen is investigated by Knuth [3].

Roulette is used as an example of a gambling application in this paper. A participant in the game chooses one or several numbers out of 37 or 38 (American roulette). A winning gives a profit of 35 times the wager. Since the chance of winning is 1 of 37 and the reward is just 36 to 1, the player will on average lose 2.7% of the wagered money in each game.

Roulette values is created by taking the sequence X with values in the interval [0, m-1] and produce the sequence Y with values between 0 and 36 by:

$$Y_i \equiv X_i \bmod 37 \tag{2}$$

This corresponds to using the least significant bits of the pseudo random value. Another alternative is to use the most significant bits:

$$Y_i \equiv \left\lfloor \frac{X_i}{37} \right\rfloor \tag{3}$$

In this paper the method (2) is used for the examples.

If m is not divisible by 37 the highest values are discarded to not make the distribution distorted. The value Y_i is output by the roulette and publicly known.

It is assumed that a, b and m are known or correctly guessed by the attacker. To guess a, b and m is not too difficult, since there exists lists of known good constants and implementations are likely to use one of these configurations.

Values for other games could be retrieved in much the same way. A slot machine with 20 different symbols on each wheel could use (2) but with mod 20 instead of mod 37.

Two LCGs can be combined to create a combined linear congruence generator, CLCG. With good constants in the underlying LCGs the generator has a period that is the product of the period of each LCG. The first CLCG presented was the Wichmann and Hill generator [9]. The equivalence between this generator and an LCG was shown by Zeisel [10].

Later L'Ecuyer [6] presented another CLCG definition. This generator is not equivalent to, but can be approximated by, an LCG.

The Frieze et. al. algorithm can be used to predict CLCGs of the Wichmann-Hill type, while the added noise in the L'Ecuyer CLCG prevents us from using the algorithm, when the known bits of the values are the least significant bits. However if the known bits are the most significant bits, the L'Ecuyer CLCG can be exposed, since the noise is in the least significant bits and the L'Ecuyer CLCG would give the

same truncated sequence as a corresponding LCG. This is also true if a window of bits is used, and no part of the window includes any of the "noisy" bits.

Also other forms of congruential generators can be predicted by the Frieze et al. attack, such as a multivariate linear congruence. While polynomial congruential generators cannot be predicted, since the attack method demands linear congruencies. A brute force attack may still be used depending on the size of the period.

3. Results of implementations

This section presents results about how many seen values in a roulette game that will suffice to do a correct prediction by the method by Frieze et al [2] and a brute force attack. What we are interested in is to compare the number of bits that have been presented by the roulette when this happens to the number of bits needed to seed the generator and calculate a quotient of these values in the following manner:

$$Quotient = \frac{bits\,observed}{bits\ in\ seed} \tag{4}$$

If this quotient is no more than 1 no more bits could be extracted from the generator before a reseeding is needed, than the number of bits of real randomness given to the generator as seed. The interval between reseedings is then so small that it would be better to use the real randomness directly to create roulette values.

3.1. Frieze et al. attack method

The Frieze et al. method requires the knowledge of a and m, while b may stay unknown. The reason why b may be unknown is that the sequence of differences between consecutive values can be predicted as well as the sequence of values. The sequence is defined as:

$$X_i' \equiv X_{i+1} - X_i \tag{5}$$

Since the actual values of X are not known but Y, only a part of X' will be known. This part is here denoted Y'. The sequence Y' is created by subtracting consecutive values in the sequence Y with each other. Using values in the Y' sequence to reconstruct the values in the X' sequence according to the attack method, a prediction for the value Y_{i+1} can be done:

$$Y_{i+1} \equiv Y_i + X_i' \bmod 37 \ \text{ or } \ Y_{i+1} \equiv Y_i + X_i' - m \bmod 37 \tag{6}$$

The left alternative is true if

$$X_i + X_i' < m \tag{7}$$

Otherwise the right alternative is true. Since it is always true that

$$X_i + X_i^{'} < 2m \tag{8}$$

there are only two options. We cannot know which of these options is true, since we cannot calculate X_i, lacking the knowledge of b. Therefore we bet on both numbers.

Equation (9) below shows the Ripley LCG generator, recommended as a good generator for casino implementations [7].

$$X_i = (1589013525 X_{i-1} + 1) \bmod 2^{32} \tag{9}$$

The Ripley generator is initialized with the seed 760611 and the method (2) is used to create roulette values. The twelve first output values are 15, 1, 32, 32, 25, 1, 25, 19, 29, 27, 24 and 24. The attacker tries to make predictions after having seen six values and the attempts are shown in table 1. It is unlikely to find a unique reconstruction before this since it takes at least six values to be likely to have a unique following value. This can be shown by investigating how many values between 0 till 36 that can be represented by 32 bits:

$$\left(\frac{1}{37}\right)^n 2^{32} = 1 \Rightarrow n = \log_{37} 2^{32} \Rightarrow n \approx 6.14 \tag{10}$$

The formula above also shows how many roulette values that could be created by the seed of 32 bits that the Ripley generator uses.

Table 1. Prediction of roulette values created from a Ripley LCG and transformed using (2).

Predicting	Prediction	Real Value	Result
7	16 or 23	25	Failed
8	22 or 29	19	Failed
9	22 or 29	29	Succeeded
10	20 or 27	27	Succeeded
11	24 or 31	24	Succeeded
12	24 or 31	24	Succeeded

Figure 1 shows how often the predictions succeed. The 6th consecutive value is predicted after seeing the five earlier, the 7th after seeing the six earlier values and so on. This is done three times with 1000 sequences in each test batch. The 3000 seeds are chosen pseudo randomly. If the sequence were truly random we ought to win around 2 of 37 ≈ 5.4% of the games.

The test shows that there is a great chance to get a correct prediction for the 8th value, demanding a reseeding after seven values to not give an attacker a statistical advantage.

Knowing that a real random value of 32 bits was used to seed the sequence and that seven values correspond to 36.5 bits, the quotient of how many times more bits that can be used of the generator's output than the seed, is 1.14, i.e. only 1.14 times more information can be received from the Ripley generator than was put in.

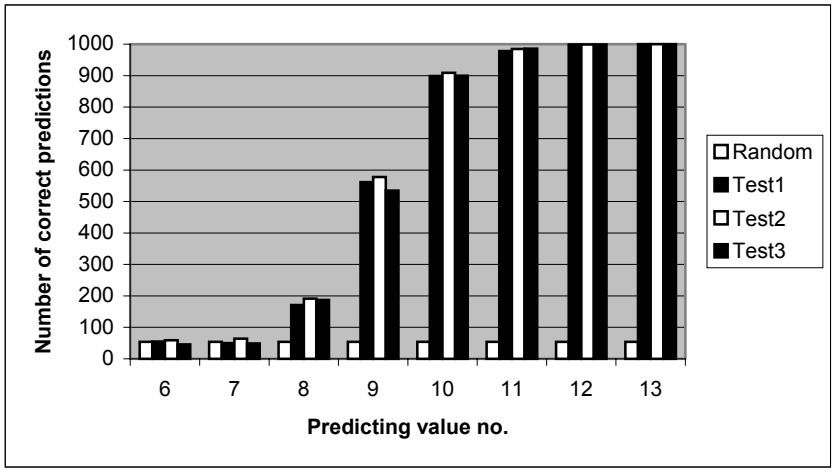

Fig. 1. How many of 1000 sequences that were rightly predicted for the Ripley LCG (9), and the expected number of winnings if the sequences were truly random.

The value *m* is quite small for the Ripley generators and a brute force attack could be used just as well as the Frieze et al. method. The next test uses an LCG with a much larger *m*. This LCG is taken from [5]:

$$X_i = 1968402271571654650 X_{i-1} \bmod 4611685301167870637 \quad \textbf{(11)}$$

By the same reasoning as in equation (10) 12 values are probably needed to get a unique reconstruction.

$$\left(\frac{1}{37}\right)^n 4611685301167870637 = 1 \Rightarrow n \approx 11.9 \quad \textbf{(12)}$$

The generator needs 62 bits of real randomness to be configured with each value in the sequence having the same chance of being the seed. The results presented in figure 2 show that more than 14 values should not be output. This corresponds to 72.9 bits and a quotient of 1.18. It can thereby be seen that also for this larger generator no or not much higher percentage of bits can be output than was given to it as seed.

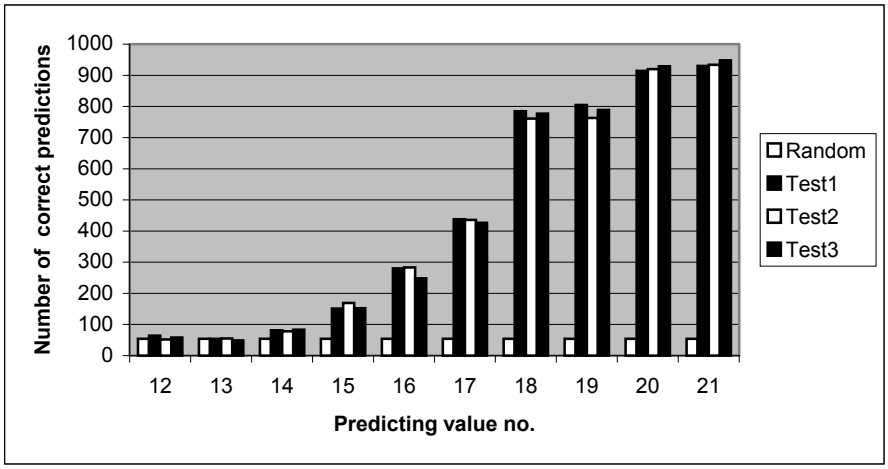

Fig. 2. How many of 1000 sequences were rightly predicted for the generator of (11), and the expected number of winnings if the sequences were truly random.

3.2. Brute force approach

With the knowledge of Y_i as defined by (2), many values for X_i cannot be true. Only $m/37$ different values for X_i are possible. A brute force approach is to test for every possible X_i if the following value X_{i+1} mod 37 is equal to Y_{i+1}. If that is true we continue to test for X_{i+2} and so on. If it is not, we investigate the next possible value for X_i and se if we have better luck. This takes rather long time even for such a small m as defined by the Ripley generator. The time can be bettered with a constant by doing some observations. Take two values X_i and X_{i+1} that match Y_i and Y_{i+1} after a modular reduction with 37. We will call the next matching values $nextmatch(X_i)$ and $nextmatch(X_{i+1})$. We can make a calculation on how much larger $nextmatch(X_i)$ must be than X_i. Let

$$nextmatch(X_i) = X_i + 37k \tag{13}$$

, then

$$nextmatch(X_{i+1}) = aX_i + 37ka + b \equiv X_{i+1} + 37ka \bmod m \tag{14}$$

We now calculate the smallest k that fulfils the following congruence:

$$X_{i+1} \bmod 37 = (X_{i+1} + 37ka \bmod m) \bmod 37 \tag{15}$$

The next value to be tested is $37k$ larger than the last one. This decreases the investigation time, but since the decrease is by a constant the method will take to much time to be of any use if the period is much larger than the m of the Ripley generator.

From equation (10) we are inclined to suspect that 6 values ought to be enough to have a great chance for a unique suggestion for the next value output by the Ripley generator. But it is not necessary for the attacker to be absolutely sure what the next value is. If we collect statistics on how large the probability is for every value from 0 to 36 to be next, we know that it will be profitable in the long run to play on any value that has larger probability than 1/36 to be next. Therefore statistical advantage is had even when predicting the 6th value, giving a quotient below 1, meaning fewer bits can safely be used of the generators' values than were used in the seed. It would therefore be better to use the real randomness from the seed to decide the roulette values.

4. Chosen Gap Method

For the chosen gap method, based on some known weaknesses of the LCG, the attacker must be able to choose a gap in the sequence when he is not playing. This gap must not be as large as a period; then it would be trivial to do the prediction. For this section it is convenient to rewrite the LCG in the following way:

$$X_i = a^i X_0 + \sum_{j=0}^{i-1} a^j b \bmod m \tag{16}$$

The method is shown using a small LCG:

$$X_i = \left(106 X_{i-1} + 1283\right) \bmod 6075 \tag{17}$$

The period is 6075, a full-period. But if b is changed to be 0, while keeping the same a and m, the period is just 405. This means that the following is true.

$$a^{405t} \equiv 1 \bmod 6075 \quad \text{for any } t \tag{18}$$

Two values at a distance of $405t$ are related in the following way:

$$X_i = X_{i-405t} + \sum_{j=0}^{405t-1} a^j b \bmod m \tag{19}$$

With a, b and m known the difference between the two values can be calculated. When working with the truncated sequence Y as defined by (2), the difference to a value $405t$ later is calculated to be

$$\sum_{j=0}^{405t-1} a^j b \bmod m \bmod 37 \quad \text{or} \quad \left(\sum_{j=0}^{405t-1} a^j b \bmod m \right) + m \bmod 37 \tag{20}$$

When the value

$$\sum_{j=0}^{405t-1} a^j b \bmod m \tag{21}$$

is close to 0 or m one of the possible values has a greater possibility to be the correct one. But we can always cover both choices by placing a double bet.

The method can be used also for CLCG. The problem for the attacker with the CLCG compared to the LCG is that the whole previous values are not enough to be able to predict the next value, since two sequences cover each other.

Let a CLCG be made of 2 generators producing sequences V and W respectively. The output X from the CLCG is defined according with (22) with m_x as the modulus of sequence X:

$$X_i = (V_i - W_i) \bmod m_x \tag{22}$$

A usual configuration of the constants is those presented in [5] or [6]. For these generators it is true that two values half the period apart added together give m. This follows from Fermat's theorem.

$$a^{m-1} \bmod m \equiv 1 \Rightarrow a^{\frac{m-1}{2}} \bullet a^{\frac{m-1}{2}} \bmod m \equiv 1 \tag{23}$$

A value half a period away from X_i can be calculated to be:

$$X_{i+\frac{m-1}{2}} = a^{\frac{m-1}{2}} X_i \bmod m \equiv X_i(m-1) \bmod m \equiv -X_i \bmod m = m - X_i \tag{24}$$

If we have two values that are half the period of the W sequence apart, we add these two values with m_w, the constant m of generator W, to get rid of the influence of the W sequence. The result is denoted T.

$$T = X_i + X_{i+\frac{m_w-1}{2}} + m_w \bmod m_x \equiv V_i + V_{i+\frac{m_w-1}{2}} \bmod m_x \tag{25}$$

As

$$V_{i+\frac{m_w-1}{2}} \equiv a_v^{\frac{m_w-1}{2}} V_i \bmod m_v \tag{26}$$

, V_i can be calculated to be one of three alternatives depending on how many modular reductions that have occurred in (25), which is no more than two.

$$V_i \equiv \frac{(T + tm_x \bmod m_v)}{a_v^{\frac{m_w-1}{2}} + 1} \bmod m_v, 0 \le t \le 2 \tag{27}$$

Therefore the whole sequence V can be calculated and by the same method W can also be calculated. However, in the gambling application we do not get the sequence X but instead Y as defined by (2). The same procedure is used, but as we only get a part of each value more values are needed to be able to make the reconstruction. T is calculated as:

$$T = X_i + X_{i+\frac{m_w-1}{2}} + m_w \bmod m_x \bmod 37 \tag{28}$$

As before modular reductions will play a role and the real difference between two values may be tm_x mod 37, for $0 \leq t \leq 2$. Getting several differences a total search could be deployed to go through the complete sequence V taken mod 37 and see if these differences could be found somewhere. Having a number of values at a chosen gap for the CLCG will therefore enable us to use a total search on the underlying LCG to find future values. The number of values needed on both sides of the half period, n, should satisfy the following:

$$\left(\frac{37}{3}\right)^n > m_x \tag{29}$$

The same principle can be used for CLCGs consisting of more generators. From this aspect we find that it is better to have fewer larger generators than many small generators for the CLCG even if they end up to around the same period for the CLCG, since a small generator has a shorter half period. We must conclude that this method is of minor importance at least for the CLCGs presented in [5] and [6], since half a period is a billion values and the attacker would have to wait a long time for this to happen and the application can surely reseed before this will happen.

5. Problems for an attacker

While the casino cannot stop the attacks by reseeding more often, there are several problems for the attacker that must be taken into consideration as the following list shows:

- Values are missing in the observed sequence. The Frieze et. al method demands that the relations between the values are known. They do not have to be in consecutive order but their relative positions must be known, i.e. if there are any gaps in the sequence. Gaps can occur for several reasons as for instance that a value is dismissed after a certain time interval when the game is not used. If the gaps are not known the attacker has to guess. This is possible if the gaps are either small or a short interval that the gaps are within is known.
- Values are modified before they are used. The gambling application may make modification of values as a way of making it harder for an attacker to predict. A simple substitution between the values is a possibility as well as a double truncation of the values.
- Unknown a and m. It has been assumed that the attacker has knowledge of a and m. If the attacker uses the method by Sterner [9] the knowledge of a and m can also be omitted, with the cost that more bits are needed for the prediction. This method has not been implemented and how well it performs and how many more bits that are needed can not be given.

6. Conclusions

In this paper it has been shown through tests how dangerous LCG/CLCG is to use in an electronic gambling application. Not much more bits can be received from the generator than was given as input as a seed before a reseeding is necessary to keep the attacker from getting a statistically advantage. It can also be concluded that for small generators such as the classical Ripley generator a brute force method could be used giving statistically advantage even earlier.

The chosen gap method based on weaknesses of the LCG, showed another way of predicting values. It did however not improve the results compared to the other methods.

This paper was motivated by the recommendation in [7] to use the Ripley generator for gaming implementations. Nothing is said about how many if any real casinos use this kind of generator.

7. References

[1] J. Boyar, "Inferring Sequences Produced by Pseudo-Random Number Generators," Journal of ACM, Vol. 36, p 129-141, 1989

[2] A.M Frieze, J. Håstad, R. Kannan, J. C. Lagarias and A. Shamir, "Reconstructing Truncated Integer Variables Satisfying Linear Congruences", Siam J. of Computing, Vol 17, p 262-280, 1988

[3] D.E. Knuth, "The Art of Computer Programming, Volume 2: Seminumerical Algorithms" Addison-Wesley Publishing, 1980

[4] H. Krawczyk, "How to Predict Congruential Generators," CRYPTO '89, Springers Verlag, 1990

[5] P. L'Ecuyer, S. Tezuka, "Structural Properties for two classes of combined random number generators", 1991.

[6] P. L'Ecuyer, "Efficient and portable combined number generators", Comm. ACM 31, p 742-749, 774, 1989

[7] Lotteriinspektionen (Swedish Lottery Authority), "Lotteriinspektionens villkor för lotter och slumptalsgeneratorer (The Swedish Lottery Authorities conditions for lotteries and random number generators)", 2002

[8] J. Stern,"Secret Linear Congruential Generators Are Not Cryptographically secure," Proceeding of the 28th Symposium of FOCS 1987

[9] B.A. Wichmann and I.D. Hill, "An efficient and portable pseudo-random number generator," Applied Statistics 31, 188-190, 1984

[10] H. Zeisel, "A remark on algorithm AS183", Applied Statistics35, 1986