SECURITY EVALUATION OF EMAIL ENCRYPTION USING RANDOM NOISE GENERATED BY LCG

Chung-Chih Li, Hema Sagar R. Kandati, Bo Sun
Dept. of Computer Science, Lamar University, Beaumont, Texas, USA
409-880-8748, 409-454-3496, 409-880-8781
licc@hal.lamar.edu, Kandati@gmail.com, bsun@cs.lamar.edu

ABSTRACT

Theoretically, using any Linear Congruence Generator (LCG) to generate pseudo-random numbers for cryptographic purposes is problematic because of its predictableness. On the other hand, due to its simplicity and efficiency, we think that the LCG should not be completely ignored. Since the random numbers generated by the LCG are predictable, it is clear that we cannot use them directly. However, we shall not introduce too much complication in the implementation which will compromise the reasons, simplicity and efficiency, of choosing the LCG. Thus, we propose an easy encryption method using an LCG for email encryption. To see how practical in predicting random numbers produced by an LCG, we implement Plumstead's inference algorithm [2] and run it on some numbers generated by the easiest congruence: $X_{n+1} = aX_n + b \bmod m$. Based on the result, we confirm the theoretical fault of the LCG, that is, simply increasing the size of the modulus does not significantly increase the difficulty of breaking the sequence. Our remedy is to break a whole random number into pieces and use them separately (with interference from another source, in our case, English text). We use 16-bytes random numbers and embed each byte of the random number as noise in one text character. In such a way, we can avoid revealing enough numbers for the attacker to predict.

Keywords: linear congruential generator, email encryption, lightweight encryptions

1. INTRODUCTION

The Linear Congruence Generator (LCG) has been used in generating pseudo-random numbers for a large variety of applications [11]. Although the pseudo-random numbers produced by an LCG bear many favorable properties to many (if not all) applications [6], they are theoretically not suggested for cryptographic purposes because they are predictable (even if the parameters of the linear congruential equations are unknown) [8]. This weakness however is not formidable in some practical applications when a lightweight encryption will suffice but efficiency is a major concern. Ordinary email exchange seems to be such an example where some lightweight encryption may be enough to provide basic protection as the envelope to regular postal mails. We therefore propose an easy method for email encryption with an LCG involved.

To assess the security strength of using the LCG, we implement Plumstead's (Boyar's) powerful inference algorithm against congruential equations in its easiest form. Our experimental results show that the algorithm can correctly predict the entire sequence with a few known numbers in the sequence. Thus, if we directly embed the numbers in our email text, some straightforward classical cryptanalysis can recover enough numbers for inference. To avoid this problem without further complicating our congruential equation, we choose big moduli (16 bytes or bigger) to produce big pseudo-random numbers and embed a number in 16 characters of the text. Since a correct guess on a string of 16 characters reveals just one number in the sequence, no prediction can be possibly made.

2. LINEAR CONGRUENTIAL GENERATORS

The Linear Congruential Generator (LCG) generates a sequence of pseudo-random numbers according to some recurrence congruence. The easiest LCG uses the following equation:

$$X_{n+1} = aX_n + b \bmod m. \tag{1}$$

In (1), a is called the multiplier, b the increment, and m the modulus. The numbers will be generated in order as a sequence: X_0, X_1, X_2, etc. In the sequence, X_0 is given, which is called seed. We say that a, b, m, and X_0 are the parameters of an LCG. The quality of an LCG depends on the selection of its parameters, which has been widely discussed for practical reasons. For example, the condition of obtaining the maximum cycle (m-1) of un-repeated numbers was given by M. Greenberger in 1961 for the special case, $m = 2^n$. In 1962, Hull and Dobell gave the general condition (detailed discussion and proofs can be found in [7]). Park and Miller [10] suggested a "minimal standard" for implementation, which is not as easy as it may seem. A more recent and complete survey was done by Entacher [3]. However, these researchers emphasized on engineering applications, mostly for simulations, and put aside the issue on predictableness. In fact, most commercial LCGs disclose their implementations with fixed parameters because secrecy is not an issue for statistical use [12]. In other words, their designs are not meant for cryptographic applications.

3. EMAIL ENCRYPTION WITH LCG

Although the market for email encryption has always been promising ever since the Internet came to its age, its research and products are unreasonably few. In part this was because software companies and individuals had been discouraged to develop an email software with encryption ability due to the export control laws of the United States. Also, most public key ciphers were patented. For example, PGP and its creator, Phil Zimmermann, had experienced all these non-technique troubles [4]. Until recently, the export controls are relaxed and patents of many public-key ciphers have gradually expired. A new wave of developing fast, secure, inexpensive, and legal email encryption software is certainly expected.

3.1. Encryption Method

In this section we introduce our email encryption method.

Hybrid Cipher: We follow the structure of a hybrid cipher used in PGP. In PGP, there is an Advanced Encryption Standard (AES) cipher for actually encrypting the email text. Also, there is an RSA cipher for encrypting the symmetric keys of the AES. In our cipher, we keep the standard RSA for the symmetric key exchange and use our LCG noise cipher for encrypting the email message.

The two parties, the sender and the receiver, shall establish keys for the RSA cipher upon their first communication. We omit the detailed descriptions about how a hybrid cipher and RSA work, which are rather standard (see [4, 13] for details). We use 512 bits (64 bytes) for the RSA public keys in our cipher, which is big enough for us to encrypt the 4 parameters of our LCG (16 bytes for each). In real implementation, we suggest using at least 2048 bits for the RSA public keys to guarantee security under today's computer power.

Key Scheduling: Since our cipher does not resist plain-text attack, the sender has to randomly select new LCG parameters for each email to send off. Repeatedly using the same LCG parameters will significantly undermine the security of our system. A fully isolated random number generator should be used for the selection of the parameters. For example, the interval of mouse clicks is considered as a good source for seeding a generator, namely selecting values for a, b, m, and X_0.

The moduli are required to be a 16-byte prime. We use the Miller-Rabin test to select prime numbers with error rate less than $1/2^{|m|}$, where |m| is the length of m (see [13], page 179). According to the Prime Number Theorem, the density of primes in 16-byte integers is about $1/(\ln 2^{128}) = 0.0127$. Thus, we can successfully pick up a prime within about 100 random tests, which is affordable. After m has been decided, we randomly assign numbers less than m to a, b, and X_0 without other restrictions except for removing some trivial values such as 0 or 2^k. There is no concern about the length of the cycle in the sequence generated, since a 128-bit prime as the modulus is very likely to generate unrepeated numbers within the length of a regular email, especially we will use one random number for every 16 characters of the email.

Encrypting Text: We assume the plaintext to be encrypted is prepared in extended ASCII code (8-bits) without compression. Compression can increase the difficulty of cryptanalysis (this is what PGP does) but we do not consider this in our test cipher. Once the parameters of our LCG are decided, we generate X_1, X_2, X_3, ... one by one, until all characters in the message have been encrypted. Starting from X_1, we embed each byte of the random number in a character of the email message. For example, let (2) shown in the following be the message to be encrypted:

```
Telecommunication in the state of the art
```
(2)

The values of the first three characters are T = 84, e =101, l = 108. Since there are 41 characters, we need three 16-byte random numbers. The embedding operation is simply the addition modulo 256. A more complicate operation doesn't seem necessary. For example, let

$$X_1 = 1O5AFB11FCBBOO112233445566778899_h$$

The first three bytes are $10_h = 16$, $5A_h = 90$, and $FB_h = 251$. The values of the first three cipher-text characters of (2) are:

$$\begin{aligned} 84 + 16 \quad &\text{mod} \quad 256 = 100 \\ 101 + 90 \quad &\text{mod} \quad 256 = 191 \\ 108 + 251 \; &\text{mod} \quad 256 = 103 \end{aligned}$$

Moreover, a message will be padded by space to make the length a multiple of 16.

For decryption, the receiver used the RSA to decrypt the parameters of the LCG. Then, run the LCG to reproduce the same sequence of the random numbers used by the sender. The decryption method is the reverses operation of addition modulo 256, which is straightforward.

4. CRYPTANALYSIS

It is well known that the LCG is predictable. This was first argued by Knuth in [8], then by Boyar [2, 1]. Until Krawczyk [9] who gave an inference algorithm for predicting any LCG in the most general form, the problem was theoretically settled. Moreover, in his survey paper [12], Ritter's again warns any attempt to use the LCG for cryptographic purposes unless the sequence can be isolated from another generator. In our cipher, we will use plain-text to serve as such a generator for isolation. Although English certainly can't be seen as a source of random numbers, we think the entropy of English has been good enough for the purpose. More precisely, the strength of our encryption method depends on the answers to the following two questions:

(1) How many random numbers in the sequence need to be known for extrapolating the entire sequence?

(2) How many random numbers used for encryption can be retrieved from the cipher-text by cryptanalysis?

First, we need experimental results for question (1) so we can propose an appropriate remedy for this theoretical weakness. For (2), we shall answer the question in Section 4.2.

4.1 Predicting the Sequence of an LCG

In this section, we briefly describe and analyze Plumstead's (Boyar's) algorithm for inferring an unknown LCG and discussion our experimental results.

Plumstead's Algorithm [2]: The algorithm assumes that the LCG is fixed to (1) with a, b, m, and X_0 unknown and no specific properties of them are further assumed. The algorithm will find a congruence, $X_{n+1} = a'X_n + b' \bmod m$, possibly with different multiplier and increment but generating the same sequence as with the unknown a, b, m, and X_0. The inference consists of two stages: (I.) Finding a' and b'; (II.) Predicting X_{i+1} and modifying the modulus m, if necessary. Let $\langle X_i \rangle$ denote sequence $X_0, X_1, X_2, \ldots\ldots$, and $\langle Y_i \rangle$ denote the sequence (X_1-X_0), (X_2-X_1), (X_3-X_2), ….., i.e. $Y_i=(X_{i+1}-X_i)$.

Stage I:

1. Find the least t such that $d|Y_{t+1}$, where $d = \gcd(Y_0, Y_1, \ldots, Y_t)$;
2. For each i with $0 \le i \le t$, find u_i such that $(\sum_{i=0}^{t} u_i Y_i) = d$;
3. Set $a' = (\sum_{i=0}^{t} u_i Y_{i+1})/d$, and $b' = (X_1 - a'X_0)$.

This stage will give $X_{i+1} = a'X_i + b' \bmod m$ for all $i \ge 0$.

Stage II: We use Gold's learning paradigm [5], i.e., when a prediction X_i is made, the actual value will be available to the inference algorithm. Initially, set $i = 0$ and $m = \infty$ and assume X_0 and X_i are available (we can reuse the numbers used in the previous stage). Repeat the following steps forever:

1. Set $i = i+1$ and predict $X_{i+1} = a'X_i + b' \bmod m$.
2. If X_{i+1} is incorrect, set $m = \gcd(m, a'Y_{i-1} - Y_i)$.

It is easy to prove that $\langle X_i \rangle$ indeed can be inferred in the limit, but the argument for an upper bound of the number of incorrect predictions is highly nontrivial (see [2] for details).

Analytic and Empirical Results of Plumstead's Algorithm: It is clear that every step in both stages is polynomial-time computable in terms of the size of m. In [2], Plumstead proves that t in Stage I is bound by the length of m. Thus, the algorithm is optimal with sample complexity $O(\log_2 m)$ in the worst case.

In practice, we found that the number of samples needed in average is far fewer than in the worst case. We test moduli m, from 1 byte in size and double the size up to 32 bytes. The parameters are selected according to the guideline of our symmetric key scheduling. For moduli m bigger that two bytes, we use the Miller-Rabin test to select prime numbers with error rate less than $1/2^{|m|}$. For each size of m, we test 1000 different sets of parameters.

m (bytes)	μ	δ	Min	Max
1	5.29541	1.06442	4	13
2	5.52248	1.04243	4	13
4	5.60579	1.15785	4	17
8	5.58585	1.11389	4	16
16	5.80249	1.76449	5	31
32	6.1059	3.14907	5	57

Table 1: The Results of Plumstead's Algorithm

Table 1 shows the results of our experiments, where μ is the average of the numbers needed to infer the sequence and δ is the standard deviation. Also, the table shows the best (min) and worst (max) cases. In our experiments we take the liberty to notify our inference program to stop when it reaches the correct point. Note that, we do not overpower our program with this ability because, in our email encryption case, this can be done by checking, for example, the index of coincidence of the decrypted cipher to decide if the random sequence is correctly inferred.

4.2. Retrieving Random Numbers from Cipher-Text

Based on the results above, it is clear that the size of m does not significantly prolong the inference process. Intuitively, this is because the size of m does not affect the number of the internal states of the LCG, which is very small [6]. Thus, if we simply add a whole random number into a plain-text character, a classical cryptanalysis can easily retrieve 5 to 6 numbers from the cipher-text, which are enough to infer the entire sequence via Plumstead's algorithm in most cases (even the modulus is as big as 32 bytes). A common attack is to search common words in typical English and simply try every position of the cipher-text to see if we can recover a meaningful message. For example, "the" can be expected in almost every English email. Moreover, since we keep all delimiters of the plain-text, a 3-letter word actually contributes 5 characters. Consider the following diagram where the segment $C_1C_2C_3C_4C_5$ in the cipher-text is obtained from " the " (note the two space before and after the word).

....	C_1	C_2	C_3	C_4	C_5	....
		t	h	e		
	X_1	X_2	X_3	X_4	X_5	

After a few trials, we will land " the " at a right position. Then, each X_i can be obtained by removing corresponding characters in " the " from the cipher character, C_i's. Note that the plain-text in (2) has two positions for " the ". In general, if we can guess a word of n letters in the message and land it at the right position, we can obtain n+2 pseudo-random numbers.

As the results shown in Table 1, it is very likely that five numbers are enough for inferring the sequence. Even if in some cases the random sequence cannot be broken with five known numbers, a serious attacker may search a dictionary for a longer word and try to find a match in the email message. For example, the word "Telecommunication" in (2) will reveal a sequence of 19 random numbers (include the ending space). As we mentioned earlier, the checking can be done automatically by examining the index of coincidence of the decrypted text obtained by removing random numbers generated by a hypothetical LCG. If the LCG is not the correct one, the index of coincidence will indicate that the decrypted text is close to a random text and the LCG should be rejected. With today's computer power, the attacker can search an entire dictionary and do the necessary computation within a few hours.

On the other hand; our remedy to this problem is to distribute a 16-byte random number into 16 text characters as shown in Section 3.1. In such a way, even if the attacker successfully guesses 48 characters, (this is very unlikely, since this requires two consecutive long words to be located in the cipher-text), giving away three pseudo-random numbers generated by an LCG can't hurt the secrecy of the LCG.

5. CONCLUSION

Since LCG is such an efficient and widely known pseudo-random numbers generator, we don't want to completely remove it from our cryptographic design just because of its theoretical weakness, especially when we can find an easy way to patch this fault. In general, this fault can be patched by blocking away the numbers from directly exposing to the attacker. In particular, email encryption is a good example to achieve this. That is, the pseudo-random numbers and the text itself can hide each other.

To conclude this investigation, we recommend that: if simplicity and efficiency are our major concerns and light-weight encryptions can meet our security requirement, LCG's should be considered, provided we should find an efficient way to hide the pseudo-random numbers generated by LCG's.

REFERENCES

[1] Boyar J., Inferring Sequences Produced by Pseudo-Random Number Generators. Journal of the ACM, 36(1):129-141, 1989.

[2] Boyar J. P., Inferring a Sequence Generated by a Linear Congruence. Proceedings of the 23rd Annual IEEE Symposium on the Foundations of Computer Science, pages 153-159, 1982.

[3] Entacher K., A Collection of Selected Pseudorandom Number Generators with Linear Structures. Technical report, 97-1, ACPC-Austrian Center for Parallel Computation, University of Vienna, Austria, 1997. http://crypto.mat.sbg.ac.at/results/karl/server/.

[4] Garfinkel S., PGP: Pretty Good Privacy. O'Reilly & Associates, mc, 1995.

[5] Gold E. M., Language Identification in the Limit. Information and Control, 10:447-474,1967.

[6] Heinrich J., Detecting a Bad Random Number Generator. Technical report, University of Pennsylvania, 2004. http://www-cdf.fnal.gov/publications/cdf685obadrand.pdf.

[7] Knuth D. E., The Art of Computer Programming, volume 2: Semi-numerical Algorithms. Addison-Wesley, 1969.

[8] Knuth D. E., Deciphering a Linear Congruential Encryption. IEEE Transaction on Information Theory, IT-31(1):49-52, January 1985.

[9] Krawczyk. H., How to Predict Congruential Generators. Journal of Algorithms, 13(4):527-545, 1992.

[10] Park S. K., and Miller K. W., Random Number Generators: Good ones are hard to find. Communication of the ACM, 31(10):1192-1201, 1988.

[11] Press W. H., Teukolsky S. A., Vetterling W. T., and Flannery B. P., Numerical Recipes in C, The Art of Scientific Computing. Cambridge University Press, New York, 2nd edition, 1992. Chapter 7 Random Number:274-328.

[12] Ritter T., The Efficient Generation of Cryptographic Confusion Sequences. Cryptologia, 15(2):81-139, 1991.

[13] Stinson D., Cryptography: Theory and Practice. 2nd Edition. Chapman & Hall Publications, 2002.