J. Cryptology (1989) 1: 177–184

Journal of Cryptology
© 1989 International Association for Cryptologic Research

# Inferring Sequences Produced by a Linear Congruential Generator Missing Low-Order Bits[1]

Joan Boyar
Computer Science Department, University of Chicago, Chicago, IL 60637, U.S.A.

**Abstract.** An efficient algorithm is given for inferring sequences produced by linear congruential pseudorandom number generators when some of the low-order bits of the numbers produced are unavailable. These generators have the form $X_n = aX_{n-1} + b \pmod{m}$. We assume that the constants $a$, $b$, and $m$ are unknown, and that $t = O(\log\log m)$ of the low-order bits are not used.

**Key words.** Cryptography, Pseudorandom number generators, Linear congruential method.

## 1. Introduction

A pseudorandom number generator is considered cryptographically secure if, even when a cryptanalyst has obtained long segments of the generator's output, he or she is unable to compute any other segment within feasible time and space complexity bounds. The linear congruential pseudorandom number generators, those of the form $X_{i+1} = aX_i + b \pmod{m}$, are very fast and widely used in Monte Carlo simulations and probabilistic algorithms. Boyar has shown that these generators are cryptographically insecure even if the constants $a$, $b$, and $m$ are unknown [1], [7].

The low-order bits of numbers produced using the linear congruential method tend to appear much less random than the high-order bits [4]. Knowing this, a cryptographer using a linear congruential pseudorandom number generator would probably only use the high-order bits. Knuth [5] has discussed the problem of predicting these sequences produced by the linear congruential method. He assumes that the modulus $m$ is known and is a power of two, but assumes that only the high-order bits of the numbers generated are actually used. Frieze *et al.* [3] have a much faster algorithm than Knuth's for predicting sequences produced by the linear congruential method even if half of the low-order bits are unavailable, but they assume that the multiplier, $a$, and the modulus, $m$, are known. Frieze *et al.* [2] have generalized this work, but they also assume that the multiplier and modulus are known. In addition, in both [3] and [2], the algorithms fail on a small set of exceptional multipliers. Stern [8] has proved similar results assuming only that the

[1] Date received: April 11, 1988. Date revised: March 28, 1989. This work was supported by an Educational Opportunity Fellowship and by DARPA Grant No. N00039-82-C-0235.

modulus $m$ is known. Stern also gives an algorithm which does not require that $m$ be known, but that result relies on two unproven assumptions.

In this paper we assume that a fixed linear congruential pseudorandom number generator, $X_{i+1} = aX_i + b \pmod{m}$, was used to produce the integers $U_i = \lfloor X_i/2^t \rfloor$, but the nonnegative constants $a$, $b$, and $m$, with $m > \max(1, X_0)$, are unknown. The low-order $t$ bits of the $X_i$'s are never known. The problem is to predict, from the high-order bits $\{U_i | 1 \leq i \leq j\}$ of some of the $X_i$'s the value of $U_{j+1}$. We assume that whenever an incorrect prediction $\hat{U}_{j+1}$ occurs, the correct value $U_{j+1}$ is revealed before $U_{j+2}$ is predicted. We show that, even if some small number $t = O(\log \log m)$ of the low-order bits are unused, the sequences produced are still cryptographically insecure. We present a polynomial-time algorithm that correctly infers the sequence with at most a polynomial in $\log_2 m$ errors. The algorithm presented here will not fail on any multiplier or modulus.

## 2. The Prediction Algorithm

To show that linear congruential pseudorandom number generators are cryptographically insecure, even if the low-order bits of the numbers produced are never seen, we describe a prediction algorithm. We assume that $t$ is a known positive integer, that $t \leq c \log_2 \log_2 m$ for some constant $c$, and that $m > 2^{3t+1}$. The sequence of numbers $\langle X_j \rangle$ is obtained by setting $X_{j+1} = (aX_j + b) \pmod{m}$ for $j \geq 0$. The only data available to the cryptanalyst is the sequence $\langle U_j \rangle$ which is obtained by setting $U_j = \lfloor X_j/2^t \rfloor$. The sequence of numbers $\langle Y_k \rangle$ is defined by $Y_k = X_k - X_{k-1}$ for $k \geq 1$. The $X_i$'s and $Y_i$'s are never available to us, but some of the $U_i$'s are available. We know, however, that $X_i = 2^t \cdot U_i + x_i$, where $0 \leq x_i \leq 2^t - 1$. If $t = 1$, then only one bit is missing and $x_i \in \{0, 1\}$. When $t$ is small enough, $2^t$ is also small, so we could try all possible values for $X_i$ for two or three different values of $i$. If, however, the algorithm is to be polynomial, we cannot do this for too many different values of $i$ simultaneously; not even for $\log_2 m$ different values.

The algorithm presented here closely parallels the algorithm for inferring sequences produced by the linear congruential method, which appears in [7]. (Generalizations of that algorithm can be found in [1] and [6].) As in that algorithm, we use the fact that $Y_k \equiv aY_{k-1} \pmod{m}$ for $k \geq 2$, we predict the $Y_i$'s, and we use these predictions to predict the $X_i$'s and $U_i$'s. The major difference between this algorithm and the algorithm in [7] is that this algorithm occasionally guesses the low-order bits of some $X_i$'s. The guesses for $X_i$, $Y_i$, and $U_i$ are written $\hat{X}_i$, $\hat{Y}_i$, and $\hat{U}_i$, respectively. If it later turns out that the guesses currently in effect cannot all be correct, the algorithm backtracks and makes at least one new guess.

A naive approach would be to follow the algorithm in [7] exactly, guessing the low-order bits each time an error is made. Since that algorithm can make $O(\log m)$ errors, this appears to lead to $m^{c \log \log m}$ guesses and this is superpolynomial. The key to keeping the running-time polynomial is Lemma 4 which shows the following: if the guesses for $X_0$, $X_1$, $X_2$, and a particular $X_k$ are correct, then, after this $X_k$ is predicted, there will be at most one $X_i$ for which we will find two possible $\hat{X}_i$'s consistent with all our previous guesses and with the high-order bits of that $X_i$, and we will never find three or more such possibilities. In all other cases, there will only be one consistent possibility for $X_i$. Thus, there is no problem in keeping the running time between successive predictions polynomial.

The algorithm makes guesses for $a$, $b$, and $m$, written $\hat{a}$, $\hat{b}$, and $\hat{m}$. In addition, it keeps track of a value $\hat{X}_{\max}$ which is used to detect errors in the current guesses for the $X_i$'s. At any given time in the execution of the algorithm, it has a set of guesses $\hat{X}_0, \hat{X}_1, \ldots, \hat{X}_{k-1}$ for an initial segment of the sequence of $\hat{X}_i$'s. $\hat{X}_{\max}$ is the maximum of these $k$ guesses. Since all of the values produced by the original linear congruence are less than the true modulus $m$, if we ever have a guess for the modulus which is less than $\hat{X}_{\max}$, then we have made an error somewhere. The only possibility for this sort of error is in our guesses for the low-order bits of the $X_i$'s, so we backtrack and make other guesses.

The initialization consists of making guesses $\hat{X}_0$ and $\hat{X}_1$ for $X_0$ and $X_1$, and letting $\hat{Y}_1$ be $\hat{X}_1 - \hat{X}_0$. There are two major cases to consider: either the sequence is the constant sequence, or it is not. If $\hat{Y}_1$ is zero at this point, the high-order bits of $X_0$ and $X_1$ are equal, and we guess that the low-order bits are also equal and that our sequence is the constant sequence. In this case, we set $\hat{a} = 1$, $\hat{b} = 0$, and $\hat{m} = \hat{X}_0 + 1$, and we predict that all the elements of the sequence are equal to $X_0$.

If $\hat{Y}_1$ is nonzero, then we guess that the sequence is not the constant sequence (if $U_0 \neq U_1$, then we are certain that it is not the constant sequence). Continuing our initialization, we make a guess $\hat{X}_2$ for $X_2$, set $\hat{Y}_2 = \hat{X}_2 - \hat{X}_1$, and set $\hat{X}_{\max}$ to the maximum of $\hat{X}_0$, $\hat{X}_1$, and $\hat{X}_2$ (the largest value seen so far). Again we consider two subcases. The first case occurs when $\hat{Y}_1$ divides $\hat{Y}_2$; in this case we can compute the multiplier $a$ immediately, without first finding $m$. We set $\hat{a} = \hat{Y}_2/\hat{Y}_1$ and $\hat{b} = \hat{X}_1 - \hat{a}\hat{X}_0$, and we make predictions for the $X_i$'s and the $U_i$'s, using the multiplier $\hat{a}$ and the additive constant $\hat{b}$, assuming that the true modulus $m$ is arbitrarily large, i.e., we predict that $\hat{X}_{i+1} = \hat{a}\hat{X}_i + \hat{b}$. Unfortunately, we are unable to check if our guesses for the $X_i$'s are correct; we can only check if they are consistent with the $U_i$'s. In fact, however, as the following lemma shows, if $X_0$, $X_1$, and $X_2$ were guessed correctly, whenever the $\hat{U}_i$'s are correct, the $\hat{X}_i$'s are also.

**Lemma 1.** *If $X_0 = \hat{X}_0$, $X_1 = \hat{X}_1$, and $X_2 = \hat{X}_2$, and $Y_1$ divides $Y_2$, then, if $\hat{X}_{j+1} = \hat{a}\hat{X}_j + \hat{b}$ and $U_j = \hat{U}_j$ for $1 \leq j \leq i$, $U_{i+1} = \hat{U}_{i+1}$ if and only if $X_{i+1} = \hat{X}_{i+1}$.*

**Proof.** Certainly if $X_{i+1} = \hat{X}_{i+1}$, then $U_{i+1} = \hat{U}_{i+1}$. The other direction is proved by induction on $i$. By assumption, $X_2 = \hat{X}_2$. Suppose that $X_j = \hat{X}_j$ for some $j \geq 2$. Then, by Theorem 1 of [7], $X_{j+1} \equiv \hat{a}X_j + \hat{b} \pmod{m}$, so $X_{j+1} \equiv \hat{X}_{j+1} \pmod{m}$. But $\lfloor X_{j+1}/2^t \rfloor = \lfloor \hat{X}_{j+1}/2^t \rfloor$, so $|\hat{X}_{j+1} - X_{j+1}| < 2^t < m$. Thus $X_{j+1} = \hat{X}_{j+1}$. □

Thus, if correct guesses have been made for $X_0$, $X_1$, and $X_2$, and $Y_1 | Y_2$, then the algorithm correctly detects the first time $X_{i+1} \neq \hat{a}X_i + \hat{b}$. Although it is possible that our sequence will have $X_{i+1} = \hat{a}X_i + \hat{b}$ for all $i$, it is likely that, eventually, this process will either give us an answer larger than the modulus or less than zero and we will make an error in predicting some $U_{i+1}$. At this point we assume we are given the correct value for $U_{i+1}$ and we make a guess for the low-order bits of $X_{i+1}$. Now we compute a guess for the modulus, $\hat{m} = |\hat{X}_{i+1} - (\hat{a}\hat{X}_i + \hat{b})|$, which will be a positive integer multiple of the correct modulus if our guesses for $X_0$, $X_1$, $X_2$, and $X_{i+1}$ are correct. If this guess $\hat{m}$ for the modulus is less than the largest $\hat{X}_i$ computed, an error has occurred, so the algorithm backtracks. If not, we proceed to the final phase of the algorithm and continue making predictions. This final phase is discussed later.

Now let us look at the more difficult second case: $\hat{Y}_1 \neq 0$ and $\hat{Y}_1$ does not divide $\hat{Y}_2$. Following the algorithm in [7], we compute $g = \gcd(\hat{Y}_1, \hat{Y}_2)$, $C_1 = \hat{Y}_1/g$, and $C_2 = \hat{Y}_2/g$. The algorithm continues, predicting that $Y_{i+1} = (C_2/C_1)Y_i$ until it is proven wrong, possibly by getting a nonintegral prediction. Of course, it can only tell if $U_{i+1} = \hat{U}_{i+1}$. We now show that if correct guesses were made for $X_0$, $X_1$, and $X_2$, then $U_{i+1} = \hat{U}_{i+1}$ if and only if $X_{i+1} = \hat{X}_{i+1}$.

**Lemma 2.** *If $X_0 = \hat{X}_0, X_1 = \hat{X}_1$, and $X_2 = \hat{X}_2$, then, if $\hat{Y}_{j+1} = (C_2/C_1)\hat{Y}_j$ and $U_j = \hat{U}_j$ for $1 \le j \le i$, $U_{i+1} = \hat{U}_{i+1}$ if and only if $X_{i+1} = \hat{X}_{i+1}$.*

**Proof.** Certainly if $X_{i+1} = \hat{X}_{i+1}$, then $U_{i+1} = \hat{U}_{i+1}$. Again the other direction is proved by induction on $i$. Suppose $X_i = \hat{X}_i$ for all $i \le j$. It is easy to see that $|C_2 Y_j - C_1 Y_{j+1}|$ is an integer multiple of $m$ [7, proof of Theorem 4]. But

$$\begin{aligned} |C_2 Y_j - C_1 Y_{j+1}| &= |C_2 Y_j - C_1(X_{j+1} - X_j)| \\ &= |C_2 Y_j - C_1(\hat{X}_{j+1} - X_j) - C_1(X_{j+1} - \hat{X}_{j+1})| \\ &= |C_1(X_{j+1} - \hat{X}_{j+1})|. \end{aligned}$$

Since $\hat{Y}_{j+1} = (C_2/C_1)^j \hat{Y}_1$, $C_1^j$ divides $\hat{Y}_1 = Y_1$, and $|Y_1| < m$, the hypothesis $U_{i+1} = \hat{U}_{i+1}$ gives $|C_1(X_{j+1} - \hat{X}_{j+1})| < (m^{1/(j-1)})2^t \le \sqrt{m} \cdot 2^t < m$. Thus $|C_2 Y_j - C_1 Y_{j+1}|$ must equal zero, so we have $X_{j+1} = \hat{X}_{j+1}$. □

The above lemma tells us that if correct guesses have been made for $X_0$, $X_1$, and $X_2$, the algorithm correctly detects the first time that $Y_{i+1} \neq (C_2/C_1)Y_i$. At this point the algorithm makes a guess for $X_{i+1}$ and computes $\hat{Y}_{i+1}$. Then the algorithm computes $\hat{m} = |C_2 \hat{Y}_i - C_1 \hat{Y}_{i+1}|$, which is a positive multiple of $m$. In order to eliminate any excess factors in $\hat{m}$ which may be preventing our solving the congruence $Y_2 \equiv \hat{a} Y_1 \pmod{\hat{m}}$, we execute the following loop:

```
repeat
    $m' \leftarrow \gcd(C_1, \hat{m}/\gcd(\hat{m}, g))$
    $\hat{m} \leftarrow \hat{m}/m'$
until $m' = 1$
```

It is possible that $\hat{m}$ is now less than the largest $\hat{X}_j$ computed so far. If so, our guesses for $X_0$, $X_1$, $X_2$, and $X_{i+1}$ were incorrect and we need to backtrack and try new values for $\hat{X}_0$, $\hat{X}_1$, $\hat{X}_2$, and/or $\hat{X}_{i+1}$. If, however, our guesses for these $X_j$'s were correct, $\hat{m}$ is a multiple of the actual modulus $m$, so we solve for $\hat{a}$ and $\hat{b}$. First we compute $C_1^{-1}$, the multiplicative inverse of $C_1$ $(\text{mod}(\hat{m}/\gcd(\hat{m}, g)))$. Then, $\hat{a} = C_1^{-1}C_2$ $\pmod{\hat{m}}$ and $\hat{b} = \hat{X}_1 - \hat{a}\hat{X}_0 \pmod{\hat{m}}$.

Thus, in all cases, except for that of the constant sequence, we are able to compute $\hat{a}$, $\hat{b}$, and a multiple $\hat{m}$ of the modulus efficiently, using only polynomial time between each successive prediction made up to that point.

In the final phase of the algorithm, we predict the remainder of the sequence, updating the guess for the modulus as necessary. In this phase we begin predicting the $X_i$'s by computing $\hat{Y}_{i+1} \leftarrow \hat{a}\hat{Y}_i \pmod{\hat{m}}$ and $\hat{X}_{i+1} \leftarrow \hat{X}_i + \hat{a}\hat{Y}_i \pmod{\hat{m}}$. Again, an error is only detected if $U_{i+1} \neq \hat{U}_{i+1}$. Again, however, assuming that the previous guesses (there have been at most four) were all correct, $U_{i+1} = \hat{U}_{i+1}$ if and only if $X_{i+1} = \hat{X}_{i+1}$.

**Lemma 3.** *During the final phase, if $X_i = \hat{X}_i$ for all $i \leq j$, then $U_{j+1} = \hat{U}_{j+1}$ if and only if $X_{j+1} = \hat{X}_{j+1}$.*

**Proof.** Similar to that of Lemma 1. □

Thus, when an error occurs in the predictions, it is detected. After it is detected, we are given the correct value for $U_{i+1}$. We then try to find an $\hat{X}_{i+1}$ consistent with $U_{i+1}$ and consistent with all previous guesses for the $\hat{X}_i$'s, and we compute a new modulus $\hat{m} = \gcd(\hat{m}, \hat{X}_{j+1} - \hat{X}_j - \hat{a}Y_j)$. Such a guess $\hat{X}_{i+1}$ is consistent if the updated $\hat{m}$ is still greater than $\hat{X}_{\max}$. Now we show that if such an error occurs, there are at most two consistent possibilities for $\hat{X}_{i+1}$, given $U_{i+1}$.

**Lemma 4.** *Within the final phase, if $X_i = \hat{X}_i$ for all $i \leq j$ and $U_{j+1} \neq \hat{U}_{j+1}$, there are at most two possibilities for $X_{j+1}$ which are consistent with $U_{j+1}$ and the previous $U_i$'s. If there are two such possibilities, then $\hat{m}$ can be updated at most $t$ times, and there will never be two possibilities for any later $X_{i+1}$.*

**Proof.** Assume for contradiction that $X_{j+1}^{(1)}$, $X_{j+1}^{(2)}$, and $X_{j+1}^{(3)}$ are distinct valid possibilities. Then $m_k = \gcd(\hat{m}, X_{j+1}^{(k)} - X_j - \hat{a}Y_j) \geq \hat{X}_{\max}$ for $k \in \{1, 2, 3\}$. Since the least common multiple of $m_1$, $m_2$, and $m_3$ divides $\hat{m}$,

$$\hat{m} \geq \frac{m_1 m_2 m_3}{\gcd(m_1, m_2)\cdot \gcd((m_1 m_2/\gcd(m_1, m_2)), m_3)}$$
$$\geq \frac{m_1 m_2 m_3}{\gcd(m_1, m_2)\cdot \gcd(m_1, m_3)\cdot \gcd(m_2, m_3)}.$$

But $|X_{j+1}^{(1)} - X_{j+1}^{(2)}| < 2^t$, so $\gcd(m_1, m_2) < 2^t$. Similarly, $\gcd(m_1, m_3) < 2^t$, and $\gcd(m_2, m_3) < 2^t$. Since one of $m_1$, $m_2$, and $m_3$ is a multiple of $m$, one of them is at least as large as $m$, and the other two are at least as large as $\hat{X}_{\max}$ or they would not be possibilities. Hence, $\hat{m} > m_1 m_2 m_3/2^{3t} \geq m\hat{X}_{\max}^2/2^{3t} > 2\hat{X}_{\max}^2 > \hat{m}$, and we have a contradiction. Therefore there are at most two valid possibilities for $X_{j+1}$.

Assume $X_{j+1}^{(1)}$ and $X_{j+1}^{(2)}$ are the valid possibilities, and let $m_1$ and $m_2$ be defined as above. Then, since the least common multiple of $m_1$, and $m_2$ divides $\hat{m}$, we have

$$\hat{m} \geq \frac{m_1 m_2}{\gcd(m_1, m_2)}.$$

Again $\gcd(m_1, m_2) < 2^t$. The initial value for $\hat{m}$ is $|\hat{X}_{i+1} - (\hat{a}\hat{X}_i + \hat{b})| = |\hat{Y}_{i+1} - \hat{a}\hat{Y}_i|$ when $\hat{Y}_1$ divides $\hat{Y}_2$. In this case, $\hat{a} = \hat{Y}_2/\hat{Y}_1$. Since $\hat{Y}_i \leq \hat{X}_{\max}$ for all $i \leq j$, we have $\hat{m} \leq 2\hat{X}_{\max}^2$. In the case $\hat{Y}_1$ does not divide $\hat{Y}_2$, the initial value for $\hat{m}$ is $|C_2\hat{Y}_i - C_1\hat{Y}_{i+1}|$, so again $\hat{m} \leq 2\hat{X}_{\max}^2$. Thus, in all cases, the product $m_1 m_2 < 2^{t+1}\hat{X}_{\max}^2$. Since both $m_1$ and $m_2$ are greater than $\hat{X}_{\max}$, both $m_1$ and $m_2$ are less than $2^{t+1}\hat{X}_{\max}$. Since $2^{t+1}\hat{X}_{\max} < 2^{t+1}m$, neither $m_1$ nor $m_2$ can be updated more than $t$ times. In addition, whenever further updates are required, there will only be one possible $X$ value. This is true because now $2^{t+1}m > \hat{m}$. If we had two possible values $X'$ and $X''$, with $m'$ and $m''$ for updated moduli, then

$$2^{t+1}m > \hat{m} \geq \frac{m'm''}{\gcd(m', m'')} > \frac{m^2}{2^t},$$

which is impossible. □

When an error is detected, i.e., $U_{k+1} \neq \hat{U}_{k+1}$, there are at most two possibilities for $X_{k+1}$. The algorithm looks for a first possibility, trying all possibilities for the low-order bits of $X_{k+1}$, and, if none exists, it backtracks. After finding a first candidate for $X_{k+1}$, the algorithm looks for a second. If no second possibility exists, the algorithm simply updates $\hat{m}$ and continues predicting $X_i$'s. Suppose a second candidate exists. Then the algorithm saves the current state and tries out the first possibility. With this possibility, it continues making predictions as before. Now, however, when errors occur, there should only be one consistent possibility. This is continued unless at some point there are no consistent possibilities. When this occurs, we restore our saved state and try out the second possibility. If the second possibility also fails, we made an incorrect guess for $\hat{X}_0$, $\hat{X}_1$, $\hat{X}_2$, or the $\hat{X}_{i+1}$ which was used to compute our initial guess for $\hat{m}$, so the algorithm backtracks.

Notice that the index $i$ of the element of the sequence $X_i$, which causes us to backtrack, depends on the guesses made; different guesses for the low-order bits of $X_0$, $X_1$, $X_2$, and of the $X_k$ which allows us to compute an initial multiple of the modulus, could lead to such different sequences that we would backtrack at different points. Other than the chance of two consistent possibilities for some $X_{k+1}$ in the final phase, there are only four different points to which we might backtrack, i.e., four different values which we must simultaneously guess correctly. Thus there are a maximum of $2(2^t)^4 < 2(\log_2 m)^{4c}$ guesses made.

The only thing that could keep the algorithm from running in polynomial time between successive predictions is if more than a polynomial number of $U_i$'s have been predicted and the algorithm backtracks to the beginning and has to predict all of them again before predicting a $U_i$ which has not yet been predicted. If, after a backtrack, the algorithm restarts its predictions exactly where it left off, then this problem has been avoided. It is unnecessary to check if the new guesses are consistent with all of the $\hat{U}_i$'s already seen. We may need to compute new values for $\hat{a}$, $\hat{b}$, and $\hat{m}$, depending on how far the algorithm backtracked (all of these will be recomputed if $\hat{X}_0$, $\hat{X}_1$, or $\hat{X}_2$ are given new values). Fortunately it is possible to compute them quickly in all cases. When $\hat{Y}_2 \not\equiv 0$ and $\hat{Y}_1 \neq \pm\hat{Y}_2$, then $\hat{a}$, $\hat{b}$, and $\hat{m}$ can be determined from $(\hat{X}_i | 0 \leq i \leq \lceil \log_2 m \rceil + 2\}$ [1, Theorem 7], [7, Theorem 4]. If $\hat{Y}_2 = 0$ or $\hat{Y}_1 = -\hat{Y}_2$, then no mistakes are ever made with $\hat{m} = \infty$. When $\hat{Y}_1 = \hat{Y}_2$, then we are just adding in $\hat{b}$. Once the $\hat{X}_j$ value is larger than $m$ or less than zero, the $\hat{U}_i$'s will remain incorrect. Thus we can do a binary search to locate the least $j$

such that $\hat{U}_j \neq U_j$. After we have an $\hat{a}$, $\hat{b}$, and $\hat{m}$ consistent with the guesses made for the low-order bits of the initial $\hat{X}_i$'s, we use them to predict the remainder of the sequence, starting with the first $X_i$ which has never been predicted. This can be done because we can quickly compute $X_{k+j}$ (mod $\hat{m}$) for any $j < m$, as $X_{k+j} = a^j X_k + b \sum_{i=0}^{j-1} a^i \pmod{\hat{m}} = (a^j X_k + (a^{j-1}/(a-1))b) \pmod{\hat{m}}$. Then we can predict that the next value is $\hat{a}X_{k+j+1} + \hat{b} \pmod{\hat{m}}$ and continue from there. Note that with a fixed set of guesses, every time the algorithm makes an incorrect prediction, $\hat{m}$ is decreased by some nontrivial factor. Since the original value for $\hat{m}$ is bounded from above by $2\hat{X}^2_{\max}$, no more than $1 + \log_2 m$ can occur even though we never check if $\hat{a}$, $\hat{b}$, and $\hat{m}$ are consistent with some of the $X_i$'s which have already been predicted. (An alternative method for handling the problem of too many repetitive predictions is to maintain a list of all consistent guesses and to run the algorithm in parallel for all of these possibilities. Since there are only a polynomial number of consistent guesses, the running time between successive predictions will still be polynomial.)

After determining that the time required to predict each value is polynomial in $m$, we can ask how many errors are made. Consider one complete set of values guessed at the four points the algorithm makes guesses for some low-order bits. If, with this set, the algorithm finds two possibilities for some $X_{i+1}$ in the final phase, there can be at most $t$ errors made after that point with each possibility. Thus, far fewer than $\log_2 m$ errors can be made. Since the algorithm described in [7] makes at most $2 + \log_2 m$ errors (other than the necessary errors for $X_0$, $X_1$, and $X_2$, this algorithm makes at most $2 + \log_2 m$ errors for each complete set of choices and thus makes less than $(2 + \log_2 m)(\log_2 m)^{4c} + 3$ errors, which is polynomial. The previous lemmas and the above discussion give us the following:

**Theorem 5.** *Assume that the sequence of numbers $\langle X_j \rangle$ is obtained by setting $X_{j+1} = (aX_j + b) \pmod{m}$, for $j \geq 0$, but $m$, $a$, and $b$ are unknown and only $U_i = \lfloor X_j/2^t \rfloor$ is observed. Assume that $t \leq c \log_2 \log_2 m$ for some constant $c$, and that $m > 2^{3t+1}$. Further assume that when a mistake is made in predicting the high-order $\lceil \log_2 m \rceil - t$ bits, that the correct high-order bits are made known. Then the algorithm presented here makes at most $(2 + \log_2 m)(\log_2 m)^{4c} + 3$ errors. Furthermore, the time needed to compute each prediction, $\hat{U}_i$, is polynomial in $\log_2 m$.*

The results in this paper indicate that using linear congruential recurrences in cryptographic applications can be very dangerous, even if the modulus and coefficients are kept secret.

**Acknowledgments**

I would like to thank my thesis advisor, Manuel Blum, who suggested the problem discussed in this paper. He was very supportive, encouraging, and helpful throughout the research. Other people I would like to thank include George Bergman, Faith Fich, Howard Karloff, Richard Karp, Jeff Lagarias, Bart Plumstead, David Shmoys, Alice Wong, and the anonymous referees, all of whom made many helpful comments.

## References

[1] Boyar, J., Inferring sequences produced by pseudo-random number generators, *J. Assoc. Comput. Mach.*, Vol. 36, No. 1, January 1989, pp. 129–141.

[2] Frieze, A. M., Hastad, J., Kannan, R., Lagarias, J. C., and Shamir, A., Reconstructing truncated integer variables satisfying linear congruences, *SIAM J. Comput.*, Vol. 17, No. 2, April 1988, pp. 262–280.

[3] Frieze, A. M., Kannan, R., and Lagarias, J. C., Linear congruential generators do not produce random sequences, *Proc. 25th IEEE Symp. on Foundations of Computer Science*, 1984, pp. 480–484.

[4] Knuth, D. E., *Seminumerical Algorithms, The Art of Computer Programming*, Volume 2, Addison-Wesley, Reading, MA, 1969.

[5] Knuth, D. E., Deciphering a linear congruential encryption, *IEEE Trans. Inform. Theory*, Vol. 31, 1985, pp. 49–52.

[6] Lagarias, J. C., and Reeds, J. A., Unique extrapolation of polynomial recurrences, *SIAM J. Comput.*, Vol. 17, N›. 2, April 1988, pp. 342–362.

[7] Plumstead, J. B., Inferring a sequence generated by a linear congruence, *Proc. 23rd IEEE Symp. on Foundations of Computer Science*, 1982, pp. 153–159.

[8] Stern, J., Secret linear congruential generators are not cryptographically secure, *Proc. 28th IEEE Symp. on Foundations of Computer Science*, 1987, pp. 421–426.