Proc. 1982 FOCS

# Inferring a Sequence Generated by a Linear Congruence

Joan B. Plumstead

Computer Science Division, University of California - Berkeley

## Abstract

Suppose it is known that $\{X_0, X_1, \cdots, X_n\}$ is produced by a pseudo-random number generator of the form $X_{i+1} = aX_i + b \bmod m$, but $a$, $b$, and $m$ are unknown. Can one efficiently predict the remainder of the sequence with knowledge of only a few elements from that sequence? This question is answered in the affirmative and an algorithm is given.

## Introduction

A pseudo-random number generator which is as cryptographically secure as the Rivest-Shamir-Adleman encryption scheme is presented in [Shamir]. This method for generating pseudo-random numbers is quite slow, though, and it is not known whether any statistical biases might be present in the sequences it generates. Blum and Micali give a pseudo-random bit generator, with arbitrarily small bias, which is cryptographically strong, assuming the problem of index finding is intractable. But their method is also slow. This brings up the question of whether any of the pseudo-random number generators commonly in use are also cryptographically secure. In particular, the linear congruential method is very popular and fast. Obviously, this method is not cryptographically secure if the modulus, $m$, is known. In that case, one could solve for $x$ in the congruence $(X_1 - X_0)x \equiv (X_2 - X_1) \bmod m$. Then $X_{i+1} = x(X_i) + (X_1 - x(X_0)) \bmod m$ will generate the original sequence. In [K1980], Knuth has discussed this problem, assuming $m$ is known and is a power of two, but assuming that only the high order bits of the numbers generated are actually used. In this paper, we assume the $m$ is unknown and arbitrary, but that the low order bits are also used. It is shown that this method is also cryptographically insecure. A similar result is given in [Reeds], but that result, unlike the one presented here, relies on the assumption that factoring is easy.

Throughout this paper, we assume that a fixed linear congruential random number generator, $X_{i+1} = aX_i + b \bmod m$, is given, but the constants $a$, $b$, and $m$ are unknown. The problem is to predict, from some of the $X_i$, the remainder of the sequence. In the first section, two algorithms which use only an initial segment of the sequence to find an $\hat{a}$ and a $\hat{b}$, such that $X_{i+1} = \hat{a}X_i + \hat{b} \bmod m$ for all $i \geq 0$, are presented and compared. In the worst case, neither algorithm requires an initial segment of length greater than $2 + \lceil \log_2 m \rceil$, but neither algorithm finds $m$.

In the second section, a modified version of one of these algorithms is used to predict $m$ and the unknown portion of the sequence. Here the problem becomes one of inference. In some cases, it may take a long time to find $m$, even though many correct predictions for the $X_i$ can be made before $m$ can be uniquely determined. The algorithm presented in this second section looks at $X_0$, $X_1$, and $X_2$, and then begins predicting the $X_i$. In a few cases, $X_0$, $X_1$, and $X_2$ will be sufficient to determine an $\hat{a}$ and a $\hat{b}$ such that $X_{i+1} = \hat{a}X_i + \hat{b} \bmod m$ for all $i \geq 0$. Otherwise, suitable $\hat{a}$ and $\hat{b}$ are determined when the first incorrect prediction is made and the correct value for that $X_i$ is made known. In all cases, when the first error occurs in predicting some $X_i$, we assume the correct value is made known. Then an initial guess is made for $m$. When further errors are made in predicting the $X_i$ and the correct values are made known, $m$ is updated so that it is consistent with all previous data. Analysis of the algorithm shows that no more than $2 + \log_2 m$ updates need ever be made.

* This research was supported by a fellowship from the University of California.

In the third section, we see that, if some of the $X_i$ with $i > j$ are known, they can also be useful in predicting $X_j$.

## 1. Finding a multiplier and increment

To show that linear congruential pseudo-random number generators are cryptographically insecure, we will assume that nonnegative integers $X_0$, $a$, $b$, and $m$ are fixed, but unknown, and that $m > max(1, X_0, a, b)$. The sequence of numbers $<X_j>$ is obtained by setting $X_{j+1} = (aX_j + b) \bmod m$, for $j \geq 0$.

In this section, two algorithms are presented for finding an $\hat{a}$ and a $\hat{b}$ such that $X_j = (\hat{a}X_j + \hat{b}) \bmod m$, for all $j \geq 0$, given a relatively short initial subsequence of $<X_j>$. With either algorithm, although it is possible that the $\hat{a}$ and $\hat{b}$ found will not satisfy $\hat{a} \equiv a \bmod m$ or $\hat{b} \equiv b \bmod m$, they can be used in predicting the remainder of the sequence $<X_j>$, and this is all we are interested in. Neither algorithm requires knowledge of $m$.

The sequence of numbers $<Y_k>$ is obtained by setting $Y_k = X_k - X_{k-1}$ for $k \geq 1$. By subtracting $X_{k+1} \equiv (aX_k + b) \bmod m$ and $X_k \equiv (aX_{k-1} + b) \bmod m$, one gets that $X_{k+1} - X_k \equiv (a(X_k - X_{k-1})) \bmod m$, and thus $Y_{k+1} \equiv aY_k \bmod m$ for $k \geq 1$. The following theorem reduces the problem of computing $a$, $b$, and $m$ to that of computing an $\hat{a}$ and an $\hat{m}$ such that $Y_{k+1} \equiv \hat{a}Y_k \bmod \hat{m}$ for $k \geq 1$.

**Theorem 1**. Suppose $Y_{k+1} \equiv \hat{a}Y_k \bmod \hat{m}$, for $1 \leq k \leq s$. Then setting $\hat{b} = X_1 - \hat{a}X_0$ gives $\hat{a}X_j + \hat{b} \equiv X_{j+1} \bmod \hat{m}$ for $0 \leq j \leq s$.
**Proof:** Suppose $Y_{k+1} \equiv \hat{a}Y_k \bmod \hat{m}$, for $1 \leq k \leq s$. Let $0 \leq j \leq s$. Then, $\hat{a}X_j + \hat{b} - X_{j+1} = \hat{a}X_j + (X_1 - \hat{a}X_0) - X_{j+1} = \hat{a}(X_j - X_0) - (X_{j+1} - X_1) = \hat{a}\sum_{i=1}^{j} Y_i - \sum_{i=1}^{j} Y_{i+1} = \sum_{i=1}^{j} (\hat{a}Y_i - Y_{i+1}) \equiv 0 \bmod \hat{m}$. Therefore, for $0 \leq j \leq s$, we have that $\hat{a}X_j + \hat{b} \equiv X_{j+1} \bmod \hat{m}$. ∎

The following algorithm finds an $\hat{a}$ and a $\hat{b}$ from $\{X_i \mid 0 \leq i \leq t\}$, where $t = O(\log_2 m)$. For purposes of this algorithm, define $gcd(Y) = Y$.

**Procedure** Find-ab1:
**if** $Y_1 = 0$ **then** $\hat{a} \leftarrow 1$
**else begin**
  Find the least $t \geq 1$, and the corresponding $d$, such that $gcd(Y_1, Y_2, \ldots, Y_t) = d$ and $d \mid Y_{t+1}$;
  Find $u_i$ for $1 \leq i \leq t$ such that $\sum_{i=1}^{t} u_i Y_i = d$;
  $\hat{a} \leftarrow (1/d) \sum_{i=1}^{t} u_i Y_{i+1}$;
**end**
$\hat{b} \leftarrow X_1 - \hat{a}X_0$;
**end** Find-ab1.

Figure 1. Algorithm 1 for finding the multiplier and increment.

This algorithm is both correct and fast.

**Theorem 2**. The algorithm Find-ab1 computes an $\hat{a}$ and a $\hat{b}$ such that $X_{k+1} \equiv \hat{a}X_k + \hat{b} \bmod m$ for all $k \geq 0$. In addition, the algorithm only requires knowledge of $\{X_i \mid 0 \leq i \leq t+1\}$ where $t \leq \lceil \log_2 m \rceil$, and can be executed in time polynomial in $\log_2 m$.
**Proof:** We will show that $Y_{k+1} \equiv \hat{a}Y_k \bmod m$ for all $k \geq 1$. By Theorem 1, this is sufficient to prove the correctness of algorithm Find-ab1. Define $g = gcd(m, d)$. Notice that $\sum_{i=1}^{t} au_i Y_i \equiv \sum_{i=1}^{t} u_i Y_{i+1} \bmod m$; so $da \equiv d\hat{a} \bmod m$ and $a \equiv \hat{a} \bmod m/g$. Since $g$ divides $Y_1$ and $m$, $g$ divides $gcd(Y_j, m)$ for all $j \geq 1$. Hence $a \equiv \hat{a} \bmod (m / gcd(Y_j, m))$ for all $j \geq 1$. It is well known [NZ] that if $x_0$ is a solution of $rx \equiv s \bmod m$, then, for any integer $k$, $x_0 + km/gcd(r, m)$ is also a solution. Since $a$ is a solution of $xY_j \equiv Y_{j+1} \bmod m$, $\hat{a}$ is also a solution, and $\hat{a}Y_j \equiv Y_{j+1} \bmod m$ for all $j \geq 1$.

To see that knowledge of $\{X_i \mid 0 \leq i \leq t+1\}$, where $t \leq \lceil \log_2 m \rceil$, is sufficient, note that from these $X_i$, $\{Y_i \mid 1 \leq i \leq t+1\}$ can be calculated. In calculating $t$, if $gcd(Y_1, Y_2, \ldots, Y_j) = g$ and $g$ does not divide $Y_{j+1}$, then $gcd(Y_1, Y_2, \ldots, Y_{j+1}) \leq g/2$. The condition $gcd(Y_1, Y_2, \ldots, Y_t) = d$ and $d \mid Y_{t+1}$ will eventually be satisfied, because if $d = 1$, then $d \mid Y_{t+1}$. Thus, since $|Y_1| < m$, $t \leq \lceil \log_2 m \rceil$. The gcd, multiplication, division, and addition operations are all polynomial, and they are only executed a polynomial number of times. Thus the algorithm is polyno-

mial. ∎

A slightly faster, but more complicated algorithm for finding $\hat{a}$ and $\hat{b}$ from $\{X_i \mid 0 \le i \le \hat{t}\}$, where $\hat{t} = O(\log_2 m)$, is given in Figure 2.

**Procedure** Find-ab2:
**if** $Y_1 = 0$ **then** $\hat{a} \leftarrow 1$
**else if** $Y_1 \mid Y_2$ **then** $\hat{a} \leftarrow Y_2 / Y_1$
**else begin**
$g \leftarrow gcd(Y_1, Y_2)$
$C_1 \leftarrow Y_1 / g$
$C_2 \leftarrow Y_2 / g$
$i \leftarrow 1$

{Look for least $i$ such that $C_2 Y_i \neq C_1 Y_{i+1}$}

**while** $(C_2 Y_i = C_1 Y_{i+1})$ **do** $i \leftarrow i + 1$

{Compute an $\hat{m}$ such that $m \mid \hat{m}$ and $gcd(C_1, \hat{m} / gcd(\hat{m}, g)) = 1$}

$\hat{m} \leftarrow |C_2 Y_i - C_1 Y_{i+1}|$
**repeat**
$m' \leftarrow gcd(C_1, \hat{m} / gcd(\hat{m}, g))$
$\hat{m} \leftarrow \hat{m} / m'$
**until** $m' = 1$

{Compute $\hat{a}$. $C_1^{-1}$ is the multiplicative inverse of $C_1$ $mod$ $(\hat{m} / gcd(\hat{m}, g))$}

$\hat{a} \leftarrow C_1^{-1} C_2 \bmod \hat{m}$
**end**
$\hat{b} \leftarrow X_1 - \hat{a} X_0 \bmod \hat{m}$
**end** Find-ab2.

Figure 2. Algorithm 2 for finding the multiplier and increment.

Let $g$ be the greatest common divisor of $Y_1$ and $Y_2$, and let $\hat{g}$ be the greatest common divisor of $g$ and $m$. Then for all $k \ge 1$, $\hat{g}$ divides $Y_k$. Define $C_1 = Y_1 / g$ and $C_2 = Y_2 / g$. For convenience define $v = (aY_1 - Y_2)/m$ and $d = g / \hat{g}$. The proof of correctness for algorithm Find-ab2 uses the following technical lemma:

**Lemma 3**. If $m$ divides an integer $q$, then $m$ divides $q / gcd(C_1, q / gcd(q, g))$.
**Proof:** This proof uses the following fact about greatest common divisors: for any $a$, $b$, and $c$, $gcd(ab, c) = gcd(a, c) \cdot gcd(b, c / gcd(a, c))$. Since $gcd(Y_1, Y_2) = g$, and $gcd(Y_1, m)$ divides $Y_2$, we have that $gcd(Y_1, m)$ divides $g$ and $gcd\left(\frac{Y_1}{gcd(m,g)}, \frac{m}{gcd(m,g)}\right) = 1$. This implies that $gcd(C_1, m / gcd(m, g)) = 1$. Now $q = wm$ for some integer $w$, so

$$\begin{aligned} & q / gcd(C_1, q / gcd(q, g)) \\ &= wm / gcd(C_1, wm / gcd(wm, g)) \\ &= wm / gcd(C_1, wm / (gcd(m, g) \cdot gcd(w, d))) \\ &= wm / gcd(C_1, w / gcd(w, d)) \\ &= w'm \text{ where } w' \text{ divides } w. \end{aligned}$$

Therefore, $m$ divides $q / gcd(C_1, q / gcd(q, g))$. ∎

**Theorem 4**. The algorithm in Figure 2 computes an $\hat{a}$ and a $\hat{b}$ such that $X_{k+1} \equiv \hat{a} X_k + \hat{b} \bmod m$ for all $k \ge 0$. In addition, the algorithm only requires knowledge of $\{X_i \mid 0 \le i \le \hat{t}+1\}$, where $\hat{t} \le \lceil \log_2 m \rceil$, and can be executed in time polynomial in $\log_2 m$.
**Proof:** Suppose that $C_2 Y_i = C_1 Y_{i+1}$ for all $i < t$, but $C_2 Y_t \neq C_1 Y_{t+1}$. Since $aY_t \equiv Y_{t+1} \bmod m$, $(Y_1 / g) a Y_t \equiv C_1 Y_{t+1} \bmod m$. This implies that $\left(\frac{Y_2 + vm}{g}\right) Y_t \equiv C_1 Y_{t+1} \bmod m$, and $C_2 Y_t \equiv C_1 Y_{t+1} - \left(\frac{v Y_t}{d\hat{g}}\right) m \bmod m$. But $d$ divides $v$ and $\hat{g}$ divides $Y_t$, so $C_2 Y_t \equiv C_1 Y_{t+1} \bmod m$. Thus, before the *repeat* loop is entered, $m$ divides $\hat{m}$ and $\hat{m} > 0$. By Lemma 3, if $m$ divides $\hat{m}$, then $m$ divides $(\hat{m} / gcd(C_1, \hat{m} / gcd(\hat{m}, g)))$; so execution of the *repeat* loop does not alter the fact that $m$ divides $\hat{m}$.

Consider the $\hat{a}$ which is computed. We will show that $\hat{a} Y_k \equiv Y_{k+1} \bmod m$ for all $k \ge 1$. Certainly $\hat{a} Y_1 \equiv Y_2 \bmod \hat{m}$, so $\hat{a} Y_1 \equiv Y_2 \bmod m$. If $\hat{a} Y_j \equiv Y_{j+1} \bmod m$, then, $\hat{a} a Y_j \equiv a Y_{j+1} \bmod m$, so $\hat{a} Y_{j+1} \equiv Y_{j+2} \bmod m$. Thus we have that $\hat{a} Y_k \equiv Y_{k+1} \bmod m$ for all $k \ge 1$. By Theorem 1, $X_{k+1} \equiv \hat{a} X_k + \hat{b} \bmod m$ for all $k \ge 0$.

If $Y_1 = 0$ or $Y_1$ divides $Y_2$, this algorithm is essentially the same as that in Figure 1. Therefore assume $Y_1 \neq 0$ and $Y_1$ does not divide $Y_2$. To see that knowledge of $\{X_i \mid 0 \le i \le \hat{t}+1\}$, where $\hat{t} \le \lceil \log_2 m \rceil$ is sufficient, note that from these $X_i$, $\{Y_i \mid 1 \le i \le \hat{t}+1\}$ can be calculated. Certainly, $C_2 Y_1 = C_1 Y_2$. Suppose the condition on the *while* loop holds for $i \le j$. Then, for $i \le j$, $Y_{i+1} = \left(\frac{C_2}{C_1}\right) Y_i$ and $Y_{i+1} = \left(\frac{C_2}{C_1}\right)^i Y_1$. Thus $(C_1)^j$ divides $|Y_1| < m$. Since $|C_1| \ge 2$, $j \le \lfloor \log_2(m-1) \rfloor$, so $\hat{t} \le \lceil \log_2 m \rceil$. The gcd, in-

verse, multiplication, division, and addition operations are all polynomial, and they are only executed a polynomial number of times. Thus the algorithm is polynomial. ▪

The worst case analysis for both of these algorithms for finding $\hat{a}$ and $\hat{b}$ gives that neither algorithm requires that more than $\lceil \log_2 m \rceil + 2$ of the $X_i$ be known. In fact, this analysis is fairly tight. When $X_0 = 0$, $a = 2^{n-1}$, $b = 2^n$, and $m = 2^n + 1$, both algorithms require knowledge of $n+3$ of the $X_i$. Thus, in the worst case, neither algorithm is better. In some cases, though, such as when $X_0 = 0$, $a = 3$, $b = 36$, and $m = 49$, the second algorithm finds $\hat{a}$ using fewer $X_i$. (This example also shows that the *repeat* loop in the second algorithm cannot be completely eliminated.) It is fairly easy to see that the second algorithm never requires knowledge of more of the $X_i$ than the first does. The following theorem shows that algorithm Find-ab2 is optimal in the sense that it never requires knowledge of more of the $X_i$ than are necessary to uniquely determine $\hat{a}$ $mod\ (m/gcd(Y_i,m))$. By this criterion, both algorithms are optimal in the cases where three or fewer of the $X_i$ are required.

**Theorem 5**. Suppose algorithm Find-ab2 uses only $\{X_0, X_1, \cdots, X_s\}$, where $s \geq 0$. Then, there exists a $z \not\equiv a\ mod\ (m/gcd(Y_1,m))$ and an $m'$ such that $Y_{i+1} \equiv zY_i\ mod\ m'$ for $1 \leq i \leq s-2$.

**Proof**: For $s < 3$, this is vacuously true, so we may assume that $Y_1 \neq 0$, and $Y_1$ does not divide $Y_2$. Recall that $C_1^{-1}$ is the multiplicative inverse of $C_1\ mod\ (\hat{m}/gcd(\hat{m},g))$. Suppose $C_1^{-1}C_1 - 1 = km/\hat{g}$.

If $s = 3$, then it is sufficient to find a $z \not\equiv a\ mod\ (m/\hat{g})$ and an $m'$ such that $Y_2 \equiv zY_1\ mod\ m'$. (Note that $\hat{g} = gcd(g,m) = gcd(Y_1,m)$). Let $z = a+1$ and $m' = |zY_1 - Y_2|$. Since $Y_1$ does not divide $Y_2$, $|aY_1 - Y_2| < |zY_1 - Y_2|$, so $m' > m > max(Y_1, Y_2)$.

If $s > 3$, then let $m' = m + C_1\hat{g}$ and $z = (C_1^{-1}+k)C_2\ mod\ m'$. Since, for $1 \leq i \leq s-2$, $C_2Y_i - C_1Y_{i+1} = 0$, we have

$$\begin{aligned} 0 &= (C_1^{-1}+k)C_2Y_i - (C_1^{-1}+k)C_1Y_{i+1} \\ &= (C_1^{-1}+k)C_2Y_i - (1 + kC_1 + km/\hat{g})Y_{i+1} \\ &= (C_1^{-1}+k)C_2Y_i - Y_{i+1} - (kY_{i+1}/\hat{g})(m + C_1\hat{g}) \\ &\equiv zY_i - Y_{i+1}\ mod\ m'. \end{aligned}$$

One could negate both $C_1$ and $C_2$ in algorithm Find-ab2 without affecting anything else, so it is easy to arrange for $C_1$ to be positive. Then $m' > m$, so there cannot exist a $Y_i > m'$. Now, we need only show that $kC_2 \not\equiv 0\ mod\ (m/\hat{g})$. If $s > 3$, $C_1$ divides $Y_2$, and hence $C_1$ divides $g$. Thus $Y_1 = C_1^2u$ and $Y_2 = C_1uv$ for some integers $u$ and $v$, with $C_1$ and $v$ relatively prime. Then $g = C_1u$ and $C_2 = v$. Since $k$ and $C_1^{-1}$ can be determined from $m$ and $C_1$ using the Euclidean algorithm, $|k| < |C_1|$. In addition $Y_3 = \left\lfloor \frac{C_2}{C_1} \right\rfloor Y_2 = uv^2$, so $gcd(Y_1,Y_2,Y_3) = u$ and $\hat{g}$ divides $u$. Hence $|k\hat{g}C_2| < |C_1uv| = |Y_2| < m$, so $|kC_2| < m/\hat{g}$. Since $Y_1$ does not divide $Y_2$, $C_2 \neq 0$ and $C_1 \neq 1$, so $k \neq 0$. Thus $kC_2 \not\equiv 0\ mod\ (m/\hat{g})$. ▪

## 2. Inferring the modulus

Determining the modulus, $m$, can be more difficult than finding an adequate $\hat{a}$ and $\hat{b}$. In fact, no bound less than $m$, such as those in Theorem 2 and Theorem 4, can be put on the number of $X_i$ required to determine $m$, or even an adequate $\hat{m}$. Consider the sequence of numbers generated by $X_{i+1} = aX_i + b\ mod\ m$ when $a = 1$. If only $X_0, X_1, \ldots, X_s$ have been seen, and $X_i \leq X_{i+1}$ for $0 \leq i \leq s-1$, then $m$ could still be any integer greater than $X_s$. Suppose an observer knew that the sequence was being generated by the linear congruential method. After seeing $X_0$, $X_1$, and $X_2$, he or she would know that for $\hat{a} = 1$ and $\hat{b} = X_1 - X_0$, $X_{i+1} \equiv \hat{a}X_i + \hat{b}\ mod\ m$ for all $i \geq 0$. This observer could then set his or her first guess for $m$ to infinity and begin predicting the $X_i$. For some $j$, his or her guess for $X_j$ will be wrong. But once the observer determines that $X_j$, the guess for $m$ can be updated, so that all the remaining $X_i$ can be correctly predicted. Thus, this particular random number generator is cryptographically insecure. Similarly, one can see that, for an arbitrary linear congruential random number generator, if, in inferring $m$, an observer only had to make a small number of updates to his or her guesses at $m$, that random number generator would also be cryptographically insecure. Throughout this section we will assume that after an incorrect guess is made for some $X_i$, the correct value is made known.

After determining $\hat{a}$ and $\hat{b}$, using Procedure Find-ab1 or Find-ab2, one could predict the $m$ and $X_i$ as follows:

1.) Initially set the guess for $m$ to infinity, and predict that $X_{i+1} = \hat{a}X_i + \hat{b}$ until an incorrect prediction is made.
2.) When this first incorrect prediction is made, $Y_{i+1} \neq \hat{a}Y_i$. But $Y_{i+1} \equiv \hat{a}Y_i \bmod m$, so make the new guess for $m$ equal to $|\hat{a}Y_i - Y_{i+1}|$. Call the guess $\hat{m}$.
3.) Continue predicting $X_{i+1} = \hat{a}X_i + \hat{b} \bmod \hat{m}$ (or $X_{i+1} = X_i + \hat{a}Y_i \bmod \hat{m}$). Whenever an incorrect guess is made for $X_{i+1}$, update $\hat{m}$ (the current guess for $m$) to $gcd(\hat{m}, \hat{a}Y_i - Y_{i+1})$.

If, when computing $X_{i+1}$, the current guess for $m$ is $\hat{m}$, then $X_{j+1} = \hat{a}X_j + \hat{b} \bmod \hat{m}$ for all $j \leq i$.

One major disadvantage to this method is that no guesses for $X_i$ can be made until $\hat{a}$ has been computed. This could require that as many as $O(log_2(m))$ of the $X_i$ be known. As mentioned previously, knowledge of the first $n + 3$ terms in the sequence is necessary when either the algorithm in Figure 1 or that in Figure 2 is used, and $X_0 = 0$, $a = 2^{n-1}$, $b = 2^n$, and $m = 2^n + 1$. In this case, one cannot determine $\hat{a}$ or $m$ from $\{X_0, X_1, \ldots, X_{n+1}\}$, but $\{X_3, X_4, \ldots, X_{n+1}\}$ is uniquely determined by $X_0$, $X_1$, and $X_2$, if one assumes that $m$ is odd. The algorithm given below would correctly predict $\{X_3, X_4, \ldots, X_{n+1}\}$ from $X_0$, $X_1$, and $X_2$. When $Y_1$ divides $Y_2$, this algorithm follows the three steps outlined above. For the other cases, the first part of this algorithm is a slight modification of procedure Find-ab2, and the last part is essentially step 3 of the approach described above.

**Procedure** Find-m-and-a:

{Handle the case where $Y_1 = 0$}

**if** $Y_1 = 0$ **then begin**
  $\hat{a} \leftarrow 1$
  $\hat{b} \leftarrow 0$
  $\hat{m} \leftarrow \infty$
**end**
**else**

  {Handle the case where $Y_1 \mid Y_2$}

  **if** $Y_1 \mid Y_2$ **then begin**
    $\hat{a} \leftarrow Y_2 / Y_1$
    $\hat{b} \leftarrow X_1 - \hat{a}X_0$
    $i \leftarrow 1$

    {Predict $X_{i+1} = \hat{a}X_i + \hat{b}$ until first error occurs}

    **while** $(X_{i+1} = \hat{a}X_i + \hat{b})$ **do**
      $i \leftarrow i+1$
      predict $(\hat{Y}_{i+1} = \hat{a}Y_i)$
      predict $(\hat{X}_{i+1} = \hat{a}X_i + \hat{b})$
    **end while**

    {Make first guess for $m$}

    $\hat{m} \leftarrow |\hat{a}Y_i - Y_{i+1}|$
  **end**
  **else**

    {Handle all other cases}

    $g \leftarrow gcd(Y_1, Y_2)$
    $C_1 \leftarrow Y_1 / g$
    $C_2 \leftarrow Y_2 / g$
    $i \leftarrow 1$

    {Predict $Y_{i+1} = \left\lfloor \frac{C_2}{C_1} \right\rfloor Y_i$ until first error occurs}

    **while** $(X_{i+1} = X_i + \left\lfloor \frac{C_2}{C_1} \right\rfloor Y_i)$ **do**
      $i \leftarrow i + 1$
      predict $(\hat{Y}_{i+1} = \left\lfloor \frac{C_2}{C_1} \right\rfloor Y_i)$
      predict $(\hat{X}_{i+1} = X_i + \left\lfloor \frac{C_2}{C_1} \right\rfloor Y_i)$
    **end while**

    {Make first guess at $m$, $a$, and $b$}

    $\hat{m} \leftarrow |C_2 Y_i - C_1 Y_{i+1}|$
    **repeat**
      $m' \leftarrow gcd(C_1, \hat{m} / gcd(\hat{m}, g))$
      $\hat{m} \leftarrow \hat{m} / m'$
    **until** $m' = 1$

    {$C_1^{-1}$ is the multiplicative inverse of $C_1 \bmod (\hat{m} / gcd(\hat{m}, g))$}

    $\hat{a} \leftarrow C_1^{-1} C_2 \bmod \hat{m}$
    $\hat{b} \leftarrow X_1 - \hat{a}X_0 \bmod \hat{m}$
  **end** {if $Y_1 \mid Y_2$}
**end** {if $Y_1 = 0$}

{Continue making predictions for the $X_{i+1}$, updating $\hat{m}$, $\hat{a}$ and $\hat{b}$ when necessary}

**do forever**
  $i \leftarrow i + 1$
  predict $(\hat{Y}_{i+1} = \hat{a}Y_i \bmod \hat{m})$
  predict $(\hat{X}_{i+1} = X_i + \hat{a}Y_i \bmod \hat{m})$
  **if** $\hat{X}_{i+1} \neq X_{i+1}$ **then**
    $\hat{m} \leftarrow gcd(\hat{m}, \hat{a}Y_i - Y_{i+1})$
    $\hat{a} \leftarrow \hat{a} \bmod \hat{m}$
    $\hat{b} \leftarrow \hat{b} \bmod \hat{m}$
  **end if**
**end do**
**end** Find-m-and-a.

Figure 3. Algorithm for finding the multiplier, increment, and modulus.

**Theorem 6**. Suppose $X_0$, $X_1$, and $X_2$ are given, and for all $i \geq 0$, $X_{i+1} = aX_i + b \bmod m$. Further suppose that when a mistake is made in predicting $X_i$, the correct value for $X_i$ is made known. In predicting the remainder of the $X_i$, the algorithm in Figure 3 makes at most $2 + \log_2 m$ errors.

**Proof:** If $Y_1 = 0$, then no mistakes are made, so assume $Y_1 \neq 0$. First assume $Y_1$ does not divide $Y_2$, and suppose the first error occurs in predicting $X_{j+1}$. At this point, $\hat{m}$ and $\hat{a}$ are computed as in procedure Find-ab2. The proof of Theorem 4 tells us that $m$ divides the $\hat{m}$ computed, and that $\hat{a}Y_i \equiv Y_{i+1} \bmod m$ for all $i \geq 1$. Since $|Y_i| < m$ for all $i \geq 1$, $m \leq \hat{m} = |C_2Y_i - C_1Y_{i+1}| < 2m^2$.

Now suppose $Y_1$ divides $Y_2$. Then $\hat{a} = Y_2/Y_1$, so by Theorem 2, $\hat{a}Y_i \equiv Y_{i+1} \bmod m$ for all $i \geq 1$. When the first error occurs in predicting $X_{j+1}$, $X_{j+1} \neq \hat{a}X_j + \hat{b}$, so $Y_{j+1} \neq \hat{a}Y_j$. Since $m$ divides $\hat{a}Y_j - Y_{j+1} = \hat{m}$, $m \leq \hat{m} = |\hat{a}Y_j - Y_{j+1}| < m^2 + m \leq 2m^2$.

In either case, within the *do forever* loop, whenever an error occurs in predicting $X_{j+1}$, then $X_{j+1} \neq X_j + \hat{a}Y_j \bmod \hat{m}$. This implies that $\hat{m}$ does not divide $\hat{a}Y_j - Y_{j+1}$, so $gcd(\hat{m}, \hat{a}Y_j - Y_{j+1}) \leq \hat{m}/2$. Therefore, whenever an error occurs in predicting $X_{j+1}$, $\hat{m}$ is updated to a value no greater than half its previous value, but no less than $m$. This means that no more than $1 + \lceil \log_2(2m^2/m) \rceil \leq 2 + \log_2 m$ errors can occur. ∎

## 3. Predictions from nonconsecutive data

The algorithm Find-m-and-a, when predicting $X_j$ uses its knowledge of $\{X_i \mid 0 \leq i \leq j-1\}$, but does not take into account any additional knowledge which may be available. Perhaps several $X_i$ with $i > j$ are known and could help in the prediction of $X_j$. As in the previous two sections, we will assume that $X_0$, $X_1$, and $X_2$ are known. If $X_0$, $X_1$, or $X_2$ is unavailable, but $X_j$, $X_{j+1}$, and $X_{j+2}$ are known, then, of course, similar methods could be used in predicting $\{X_i \mid i \geq j\}$. If $a$ and the $\hat{a}$ computed are both relatively prime to $m$, one could also predict $\{X_i \mid 0 \leq i < j\}$ using inverses.

Let $I_1$ be the least $i \geq 2$ such that $X_i$ and $X_{i+1}$ are known. For $j \geq 1$, define $I_{j+1}$ to be the least $i > I_j$ such that $X_i$ and $X_{i+1}$ are known. Define $U_j = X_{I_{j+1}} - X_{I_j}$ and $V_j = X_{I_{j+1}+1} - X_{I_j+1}$. In many situations, a $U_j$ can be thought of as corresponding to some $Y_i$, and $V_j$ to $Y_{i+1}$. For example, in the case where only $\{X_i \mid 0 \leq i \leq k\}$ is known, $I_j = j+1$, $U_j = Y_{j+2}$, and $V_j = Y_{j+3}$, for $1 \leq j \leq k-2$. As with the $Y_i$ and $Y_{i+1}$, by subtracting $X_{I_{j+1}+1} \equiv (aX_{I_{j+1}} + b) \bmod m$ and $X_{I_j+1} \equiv (aX_{I_j} + b) \bmod m$, one gets that $X_{I_{j+1}+1} - X_{I_j+1} \equiv (a(X_{I_{j+1}} - X_{I_j})) \bmod m$, so $V_j \equiv aU_j \bmod m$.

The second algorithm for finding $\hat{a}$ used the fact that, for all $j \geq 1$, $C_2Y_j \equiv C_1Y_{j+1}$. The proposition below shows that something similar holds for the $U_j$ and $V_j$, but first two facts are needed.

**Fact.** $U_j = \sum_{i=I_j+1}^{I_{j+1}} Y_i$ and $V_j = \sum_{i=I_j+2}^{I_{j+1}+1} Y_i$.

**Fact.** For all $j \geq 1$, $\hat{g}$ divides both $U_j$ and $V_j$.

**Proposition 7**. $C_2U_j \equiv C_1V_j \bmod m$, for all $j \geq 1$.

**Proof:** Since $aU_j \equiv V_j \bmod m$, $(Y_1/g)aU_j \equiv C_1V_j \bmod m$. Thus $\left(\frac{Y_2+vm}{g}\right)U_j \equiv C_1V_j \bmod m$, and $C_2U_j \equiv C_1V_j - \left(\frac{vU_j}{d\hat{g}}\right)m \bmod m$. But $d$ divides $v$ and $\hat{g}$ divides $U_j$, so $C_2U_j \equiv C_1V_j \bmod m$. ∎

Suppose $I_j$ is defined for $1 \leq j \leq k$. Then $m$ divides $\hat{m} = gcd(|C_2U_j - C_1V_j|$ for $1 \leq j \leq k)$. Assuming a sufficient number of the $X_i$ are known, this $\hat{m}$ will be nonzero and can be

used in place of the $[\hat{m} \leftarrow |C_2 Y_i - C_1 Y_{i+1}|]$ statement in the algorithm in Figure 2 (of course the *while* loop should also be eliminated), and $\hat{a}$ can be computed. Once $\hat{a}$ is found, one can try to improve the guess for $m$ by computing $gcd(\hat{m}, gcd(|\hat{a}U_j - V_j| \text{ for } 1 \le j \le k))$, since this must be divisible by $m$. Call this improved guess for $m$, $\hat{m}$. Then $\hat{a}$ and $\hat{m}$ can be used in predicting the $Y_i$ and from them the $X_i$. If $\hat{m}$ is less than $2|X_i|$ for some $X_i$, then $\hat{m} = m$. Otherwise, some updates may still be necessary.

## 4. Conclusion

It was shown that, from knowledge of a few of the numbers produced by a linear congruential pseudo-random number generator, one can do quite well at predicting the remainder of the sequence. Although this says little about the effectiveness of using the pseudo-random number generators in probabalistic algorithms, it does make their use in cryptographic applications questionable.

Recently the results presented here have been generalized to inferring sequences of the form $X_i = a_1 X_{i-1} + a_2 X_{i-2} + \ldots + a_n X_{i-n} \bmod m$. These results will appear at some later date.


## Acknowledgements

I would like to thank Manuel Blum for suggesting this problem and for many helpful discussions. I would also like to thank Faith Fich, Zvi Galil, Howard Karloff, Bart Plumstead, and David Shmoys for their criticisms of earlier versions of this paper.


## References


[BlMi] Blum, M., and Micali, S., *How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits*, Proc. 23rd IEEE Symp. on Foundations of Computer Science, 1982.

[K1980] Knuth, D.E., *Deciphering a Linear Congruential Encryption*, Technical Report 024800, Stanford University, 1980.

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

[NZ] Niven, I., and Zuckerman, H.S., *An Introduction to the Theory of Numbers, 3rd Edition*, John Wiley and Sons, Inc., New York, 1972.

[Reeds] Reeds, J. *"Cracking" a Random Number Generator*, Cryptologia, Vol. 1, January 1977.

[Shamir] Shamir, A., *On the Generation of Cryptographically Strong Pseudo-Random Sequences*, International Colloquium on Automata, Languages, and Programming, 7th, 1980.