Linear Congruential Generators Over Elliptic Curves

Sean Hallgren
May 1994
CMU-CS-94-143

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Random numbers are useful in many applications such as Monte Carlo simulation, randomized algorithms, games, and password generation. It is important to be able to prove facts about about pseudo-random number generators, both about the distribution and the predictability of the pseudo-random numbers. I discuss a pseudo-random number generator based on elliptic curves taken over finite fields. This class of generators can produce provably good pseudo-random numbers. Also, I prove that the analog of a faster pseudo-random number generator embedded in an elliptic curve fails to produce good pseudo-random numbers.

This report was submitted in partial fulfillment of the requirements for the Senior Honors Research Program in the School of Computer Science at Carnegie Mellon University.

Keywords: cryptography, pseudo-random number generation, elliptic curves, linear congruential generators

1 Introduction

Random numbers are useful in many applications such as Monte Carlo simulation, randomized algorithms, games, and password generation. Pseudo-random number generation involves inputting a short random seed and outputting a long pseudo-random sequence. Desirable qualities of generators include generation of uniformly distributed bits, unpredictability of the bits, and fast generation of the bits.

The basic model of a generator is shown in figure 1. The random seed is first transformed into an instance. An instance is just the state needed between each number output. On each iteration a number is output and the next instance is calculated. A function f takes an instance to an instance, and a number release function r is used to output a number as a function of the current instance.

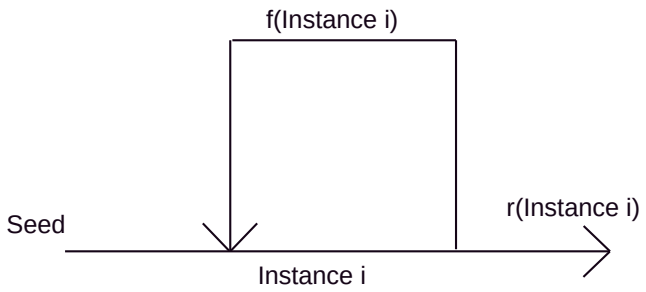

Figure 1: The basic iterative model for a generator.

For example, in the linear congruential generator $f(x) = ax + b \ (mod \ p)$ where p is a prime, the seed is transformed into the constants a, x_0, b, and n, and the number release function is $r(x) = x$. Say the random seed was $a = 3, x_0 = 7, b = 2$, and $p = 31$. Then $f(x) = 3x + 7 \ (mod \ 31)$, and the first instance is just 2. The numbers output would be 2, 13, 15, 21 ....

My goal here is to study generators with provably unpredictable output as defined by the notion of *cryptographic security*. A generator is cryptographically strong if solving some intracable problem is reducible to guessing the output of the sequence. An intractable problem is one with no algorithm to solve the problem in randomized polynomial time. Examples of problems believed to be intracable include factoring integers, the discrete log problem over finite fields, and the discrete log problem over elliptic curves.

There appears to be a tradeoff between the speed of the pseudo-random bit generator and the unpredictability of bits generated. On the fast side we have the linear congruential generator over integers mod n, but the sequence is predictable ([Plu82]). On the slower side are the cryptographically strong generators in which no bits can be predicted ([BM84]). In this paper linear congruential generators over elliptic curves are examined. One might hope that this would be a fast, unpredictable generator (Kaliski lists this as an open problem in his thesis [Kal88], originally suggested by V. Miller.) The result of this paper is that the linear congruential generator over an elliptic curve is not secure.

2 Elliptic Curves

Definition 1 *An* elliptic curve E *is the set* $\{(x,y)|y^2 = x^3 + ax + b\}$ *(with* $4a^3 - 27b^2 \neq 0$*), where the points are taken over some field, together with a point at infinity* O*.*

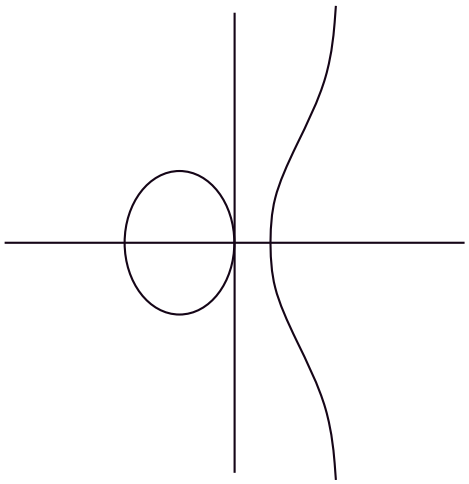

Figure 2: $y^2 = x^3 - x$

Figure 2 shows an example curve over $\mathbb{R} \times \mathbb{R}$.

The set of points is made into a group by a "tangents and chords" operation. Given two points P and Q, to determine $P + Q$:

1. draw a line through the two points (when P=Q the line is tangent at P),

2. the line will intersect the curve at exactly one other point, reflect this point across the x-axis.

This point is $P + Q$ (See Figure 3).

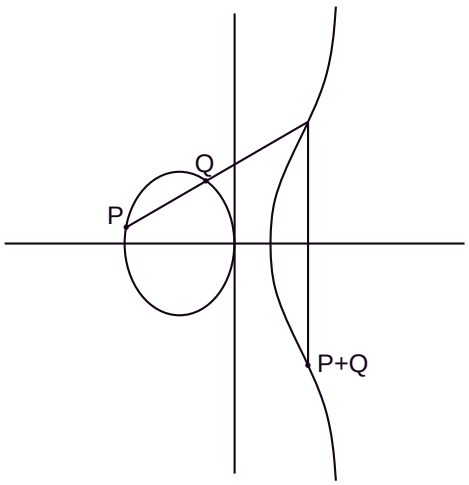

Figure 3: Adding P and Q

The identity of the group is the point at infinity O. To add points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, with neither the identity and $P \neq \pm Q$ the equation for $P + Q = (x_3, y_3)$ is:

$$x_3 = \left(\frac{y_1 - y_2}{x_1 - x_2}\right)^2 - x_1 - x_2,$$

$$y_3 = -\left(\frac{y_1 - y_2}{x_1 - x_2}\right) x_3 - \frac{x_1 y_2 - x_2 y_1}{x_1 - x_2}.$$

If $P = Q$, then

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1,$$

$$y_3 = -\left(\frac{3x_1^2 + a}{2y_1}\right) x_3 - y_1 + \frac{3x_1^3 + a x_1}{2y_1}.$$

The group operation is written as a sum since the operation is commutative. A multiple of a point X is taken by $nX = \sum_{i=1}^{n} X$.

In cryptography curves are considered over finite fields so that messages can be mapped onto and off of the curve.

Definition 2 *A* generator *of a cyclic group G is an element g such that $\{g^n | n \in \mathbb{Z}\} = G$.*

Definition 3 *Given a generator g of a field and an element y in the field the* discrete log problem *over a finite field is to find the value x such that $y = g^x$.*

Definition 4 *Given a curve E, a generator G for E, and a point X on E, the* discrete log problem *over an elliptic curve is to find $log_G(X)$ (also called the index of X), i.e. the integer a such that $aG = X$.*

A bound on the number of points on an elliptic curve over a finite field is given by Hasse's Theorem:

Theorem 5 *If $E(F_q)$ is an elliptic curve over the finite field F_q with q elements, and $\#E(F_q)$ is the number of points on the elliptic curve (including O), then $|\#E(F_q) - q - 1| \leq 2\sqrt{q}$.*

Elliptic curves thus have an advantage over finite fields. Finite fields only exist at sizes which are powers of primes, and given any size there is exactly one. Elliptic curve groups can have many groups at any given size, so there are many more structures to use.

The group structure is also known. For some n_1 and n_2, where $n_2 | n_1$, $E(F_q) \cong (\mathbb{Z}/n_1\mathbb{Z}) \times (\mathbb{Z}/n_2\mathbb{Z})$.

There are many other uses of elliptic curves in cryptography. They include Lenstra's factorization algorithm, Goldwasser-Killian and Aldeman-Huang's primality testing. Also, many cryptosystems and key exchange systems originally over finite fields or rings have analogs using elliptic curves.

For more information on elliptic curves, see [Kob87] and [Kal88].

3 Cryptographically Strong Generators

In general, a generator is cryptographically strong if solving some hard problem such as discrete log over a finite field is reducible to guessing the next number (or, better, the next bit) in the sequence, given the first polynomially many numbers (bits) in the sequence. It is better for the sequence of bits to be shown unpredictable because if it is only shown for numbers, it is not known whether or not certain bits of certain numbers can be guessed.

[Sha81] first defined the notion of a cryptographically strong sequence of numbers. In [BM84] the notion of a cryptographically strong pseudo-random bit (CSPRB) generator G is defined. G, given a seed, outputs the sequence of bits b_0, b_1, b_2, ... with the following two properties:

1. The b_i's are easy to compute (i.e., in time polynomial in the length of the seed).
2. The b_i's are unpredictable (i.e., given the first k bits of the sequence, it is computationally infeasible to predict bit $k+1$).

Definition 6 *A* predicate *is a function from the natural numbers to* $\{0,1\}$.

Definition 7 *A set of predicates B is* accessible *if given an integer k, there exists an algorithm that outputs an instance of size k which is uniformly distributed over the seeds of size k.*

Definition 8 *A set of predicates B is* unapproximable *if for some polynomial P, the size of the minimum circuit that computes B correctly for at least $1/2 + 1/P(n)$ of the inputs of size n grows faster than any polynomial in n.*

A CSPRB generator is parameterized by a set of seeds I (I_n is the set of seeds of size n), a function f from instances to instances, and a set of unapproximable and accessible predicates B. The fact that the set of predicates is unapproximable is based on an intractability assumption of some hard problem. The hard problem used here will be computing the discrete log of a point on an elliptic curve.

4 A CSPRB Generator over an Elliptic Curve

In this section we describe the ingredients for a CSPRB generator using an elliptic curve. This is just a simple example to provide motivation for the next section. This proof was also discovered by [Kal88]. The reduction used here is similar to that in [BM84] for the discrete log over a finite field with differences including that the group size will be odd instead of even.

In this section a special class of elliptic curves will be used. The *simple* class of elliptic curves contains those curves in which the constant a is 0 and p is congruent to 2 mod 3, where p is the size of the underlying field. A simple elliptic curve is cyclic and every point has a unique y-coordinate.

For a given group $E(F_q)$ and generator G, the function f for the generator is defined by $f(X) =$ y-coordinate$(X)G$. The predicate B is defined by $B(X) = LSB(log_G(X))$. This function and predicate work well because if one could approximate $B(X)$ then one could solve the discrete log problem over an elliptic curve. The theorem below shows how the reduction works when one has an oracle for $B(X)$.

Lemma 9 *When the group size N is odd it is possible to find a "square root" with respect to a generator, i.e. from $P = log_G(P)G$ it is possible to compute $P/2 = (log_G(P)/2)G$.*

Proof: $1/2 = (N+1)/2$ (here constants are taken $(mod\ N)$, where N is the group size), so $P/2 = ((N+1)log_G(P)/2)G = (log_G(P)/2)G$. ■

Comment. Finding the square root of a point is actually dividing the logarithm in half, but it is called a square root operation since it is the same as the square root in the multiplicative group. Here we write $P/2$ to indicate the square root.

Corollary 10 *When $log_G(P)$ (the index) is even, the index of $P/2$ is half of the index of P, i.e. $log_G(P) = 2log_G(P/2)$.*

Theorem 11 *Computing the discrete log of a point X on an elliptic curve $E(F_q)$ with odd size N, with respect to a generator G, is polynomially reducible to computing the least significant bit of the discrete log of X.*

Proof: Let an oracle $\mathcal{O}_{E(F_q)}$ for $LSB(log_G(X))$ and a point X be given. To compute $log_G(X)$, repeat the following (keeping track of the bit at each step, of course).

1. If $X = G$, stop.
2. If $LSB(log_G(X)) = 1$ by the oracle, then set $X = (X - G)/2$.
3. Otherwise the LSB is 0, set $X = X/2$.

On each iteration the next bit is obtained (working right to left in the index) and the index is just the concatenation of the bits. Note that this works because if $log_G(X)$ is odd, then $log_G(X-G)$ is even, and then by corollary 10 this algorithm just shifts the index right once on each iteration. ■

Assumption 12 *For every polynomial P, there is no circuit polynomial in the size of the input n that solves the discrete log problem for $1/2 + 1/P(n)$ of the points X on the curve E given a generator G for the curve E.*

[Kob87] points out that it is likely that the discrete log problem over elliptic curves is probably more intractable than the discrete log problem over finite fields. The strongest techniques that work with the finite fields case do not work in the elliptic curve case. This is especially true in fields of characteristic 2, which is a particularly easy case in finite fields.

The generator works as follows: Convert the initial random input into a curve, a point X_0 on the curve, and a generator G of the curve. Define X_{i+1} = y-coordinate$(X_i)G$. Generate the sequence of bits LSB(X_0), LSB(X_1), ..., LSB(X_n), and output them in the reverse order.

Theorem 13 *The discrete log problem over an elliptic curve is reducible to computing a bit in the sequence of the generator above.*

Proof: Suppose there is a polynomial time algorithm A which outputs bit k of the sequence given the first $k-1$ bits of the sequence. Let a point Y on the curve be given. We show how to use A to find $LSB(log_G(Y))$. Generate a $k-1$ bit sequence from the generator using Y as X_0. Give these $k-1$ bits to A in reverse order. Note that since the bits are in reverse order, the next bit in the sequence is $LSB(log_G(Y))$. This is exactly what A returns. By theorem 11, if we can compute $LSB(log_G(Y))$ we can compute $log_G(Y)$, so it follows that we can compute $log_G(Y)$. ■

This example is actually simple compared to the full blown model in [Kal88]. There, the reduction handles general groups (not just cyclic, as here). Also,

the algorithm used above to find bit k of the sequence must only approximate the bit, not give a correct answer every time.

This generator has important consequences in complexity theory. By a theorem in [Yao82], if the discrete log problem is actually intractable, then it follows that $R \subset \cap_{\epsilon>0} DTIME(2^{n^\epsilon})$, where R is the set of decision problems solvable in random polynomial time. This puts a subexponential bound on any algorithms solvable in random polynomial time.

5 Linear Congruential Generators Over Elliptic Curves

The linear congruential generator is commonly used in practice. The seed consists of a list of constants a, b, n, and x_0. The numbers output are x_0, x_1, x_2, ... , where x_i is determined by the recurrence $x_{i+1} = ax_i + b \ (mod\ n)$. However, this generator is unsecure ([Plu82], [FKL84]).

In [Plu82] it is first shown that computing the constants a, b and n is reducible to computing an a' and n such that $y_{k+1} = a'y_k \ (mod\ n)$, which generates a sequence of differences (i.e., $y_k = x_k - x_{k-1}$). Then the following algorithm gives a and b. If $y_1 = 0$ then $a = 1$. Else do the following:

1. Find the least $t \geq 1$ and the corresponding d such that $gcd(y_1, \ldots, y_t) = d$ and $d | y_{t+1}$.
2. Find u_i for $1 \leq i \leq t$ such that $\sum_{i=1}^{t} u_i y_i = d$
3. $a = (1/d) \sum_{i=1}^{t} u_i y_{i+1}$

Finally, set $b = x_1 - ax_0$.

It is natural to wonder if a linear congruential generator over an elliptic curve is secure. In this case, the sequence of points X_0, X_1, X_2, ... is computed, where the points are determined by the recurrence $X_{i+1} = aX_i + B$ for some elliptic curve E, constant a, point B, and point X_0. [Kal88] lists this as an open problem. In this case the equation produces points, so some function of the points must be output. Since cryptographically strong sequences are our goal, outputting the discrete log of the point with respect to some generator G is an obvious choice. Thus the sequence will be $log_G(X_0)$, $log_G(X_1)$, $\ldots log_G(X_m)$.

Theorem 14 *The linear congruential pseudo-random number generator over an elliptic curve (cyclic case) where the number output is $log_G(X_i)$ is not secure, and where $X_{i+1} = aX_i + B$.*

Proof: Given any instance of such a generator, there is a linear congruential generator over $\mathbb{Z}_n$ which outputs the same sequence. Let the group A, its size n, a generator G, a constant a, a point B, and a seed X_0 be given. Choose b such that $B = bG$ and x_0 such that $X_0 = x_0G$. Proof by induction on i. The first number output is $log_G(X_0) = log_G(x_0G) = x_0$ as it is for the case over $\mathbb{Z}_n$. Assume that x_i is the same for both generators. Then $log_G(X_{i+1}) = log_G(aX_i + B) = log_G(ax_iG + bG) = log_G((ax_i + b)G) = ax_i + b(mod n)$, as it is for the case over $\mathbb{Z}_n$. ■

So, if the sequence is generated by a linear congruence over an elliptic curve, and the reduction is based on the difficulty of the discrete log over an elliptic curve, it does not seem obvious that it is possible to construct a CSPRB generator. The other sequence which should be examined is the same expect output y-coordinate(X_i) instead of $log(X_i)$.

6 Conclusion

We have given a generator in which no bit in the output can be computed assuming that the discrete log problem over an elliptic curve is hard. In this generator the function f was $f(X)$ = y-coordinate$(X)G$ and the bit release function r was $r(X) = LSB(log_G(X))$. We then analyzed the analog of the linear congruential generator over an elliptic curve by changing f to $f(X) = aX + B$ and r to $r(X) = log_G(X)$. This generator turned out to be equivalent to the same generator over integers mod n, and hence is not secure. A problem still open is whether leaving f as $f(X) = aX + B$ and changing r to $r(X)$ = y-coordinate(X) would result in a secure generator.

7 Acknowledgments

I am grateful for the Small Undergraduate Research Grant from the Undergraduate Research Initiative. Thanks to my advisor Doug Tygar for suggesting the problem of using elliptic curves in pseudo-random number generation and many useful discussions. Thanks to Chris Olson for helping

me with the initial revisions of this thesis. And, finally, thanks to Mark Stehlik for many things, but especially for letting me in CMU in the first place!

References

[BM84] Blum, M., and Micali, S., How to generate cryptographically strong sequences of pseudo-random bits. In *SIAM J. Comput.* 13(4):850-864, 1984.

[FKL84] Frieze, A. M., Kannan, R., and Lagarias, J. C., Linear congruential generators do not produce random sequences. Technical Report CMU-CS-84-146, Carnegie Mellon University, 1984.

[Kal88] Kaliski, Burton S., Jr., Elliptic curves and cryptography: a pseudorandom bit generator and other tools. Ph.D. Thesis, Massachusetts Institute of Technology, 1988.

[Kob87] Koblitz, N., *A Course in Number Theory and Cryptography*, 1987.

[Plu82] Plumstead, J., Inferring a sequence generated by a linear congruence. In *Proceedings of the Twenty-third Annual Symposium on Foundations of Computer Science*, pages 153-159, 1982.

[RT88] Reif, J. H., Tygar, J. D., Efficient parallel pseudorandom number generation. In *SIAM J. Comput.*, Vol 17, No. 2, April 1988.

[Sha81] Shamir, A., On the generation of cryptographically strong pseudorandom sequences. In *Eighth International Colloquium on Automata, Languages, and Programming*, pages 554-550, 1981.

[Yao82] Yao, A., Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science*, pp. 80-91.