A Pseudo-Random Bit Generator Based on Elliptic Logarithms [1]

Burton S. Kaliski, Jr.
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

Abstract

Recent research in cryptography has led to the construction of several *pseudo-random bit generators*, programs producing bits as hard to predict as solving a hard problem. In this paper,

1. We present a new pseudo-random bit generator based on *elliptic curves.*
2. To construct our generator, we also develop two techniques that are of independent interest:
   (a) an algorithm that computes the order of an element in an arbitrary Abelian group; and
   (b) a new oracle proof method for demonstrating the simultaneous security of multiple bits of a discrete logarithm in an arbitrary Abelian group.
3. We present a new candidate hard problem for future use in cryptography: the *elliptic logarithm problem.*

1 Introduction

This paper describes a method for producing pseudo-random bits based on the elliptic logarithm problem. The paper contains background on elliptic curves and pseudo-random bit generation, two new results of independent interest, and the construction and proof of a pseudo-random bit generator. This section gives an overview of the paper.

1.1 Motivation and overview

Recently considerable progress has been made in formalizing the theory of pseudo-random number generation based on computational difficulty [BM84] [Yao82] [GGM84] [Lev85]. However, the generality of this theory (finally based on one-way functions in a weak sense [Lev85]) is in sharp contrast with the very few *concrete* candidates for one-way functions. Discrete logarithm and integer factorization (of which quadratic residuosity and inverting RSA are special cases) are essentially the only hard problems on which to build one-way functions. One of the main contributions of this paper is that it introduces a new hard problem *different* than those previously studied. Since cryptography stimulates mathematical research, it is interesting to note that ours is one of the first cryptographic tools based on 20th century mathematics.

In simplest form, an *elliptic curve* is the set of solutions (x, y) to an equation

$$y^2 = x^3 + Ax + B \tag{1}$$

over the finite field with p elements, where p is a prime. A well-known result is that *the points on an elliptic curve form an abelian group* under an additive composition operation called "tangents and chords." We use the group structure to apply the main ideas of the Blum-Micali generator in an entirely new context.

In the Blum-Micali case, the hard problem is "discrete logarithms" modulo p: given $g, y \in \mathbf{Z}_p^*$, find a such that $y \equiv g^a$ modulo p. In our case, the hard problem is "elliptic logarithms" on an elliptic curve modulo p: given points G, Y, find a such that $Y = aG$.

Despite the similarity of the statements and of the names, we are dealing with two very distinct problems. First, the structure of elliptic curve groups differs greatly from those groups previously studied. Second, the representation differs: points on elliptic curves require two coordinates. Third, while there

[1] This research was supported by NSF grant MCS-8006938.

A.M. Odlyzko (Ed.): Advances in Cryptology - CRYPTO '86, LNCS 263, pp. 84-103, 1987.

are closed formulas for computing the order of $\mathbf{Z}_p^*$ and other groups, there are no such formulas for elliptic curves. To summarize, elliptic logarithms involve entirely different mathematics. (They are also conjectured to be harder to compute than discrete logarithms [Mil85a].)

To construct a pseudo-random bit generator based on elliptic curves, and to prove that the bits it outputs are as hard to predict as solving the elliptic logarithm problem, is not a straightforward generalization of previous work. The differences pointed out above make previous constructions and proofs inadequate. In developing our construction and proofs, we also develop several related results.

1. We construct a novel method of finding the order of an element of an Abelian group.
2. We also introduce a new proof technique that generalizes proofs of bit security to abelian groups of arbitrary structure.
3. Furthermore, we lay the foundation for the development of cryptosystems using elliptic curves directly.

We make use of Lenstra's new factoring algorithm based on elliptic curves [Len85] [Bac85] (which, at first glance, would seem more suitable to *break* cryptosystems than to *construct* them!). Lenstra's algorithm has the remarkable property that its running time depends on the size of the *smallest prime factor* of its input. This allows us, for instance, to find elements that generate an abelian group quickly with negligible probability of error. This solves a major problem encountered in earlier constructions of pseudo-random bit generators. We believe ours is the first cryptographic application that exploits the special features of Lenstra's algorithm.

2 Background

2.1 Number theory

Let $\varphi(N)$ be the *Euler totient function*, the number of integers N or smaller relatively prime to N. Rosser and Schoenfield [RS62] give a lower bound for $N > 3$ of

$$\frac{\varphi(N)}{N} \geq \frac{1}{6 \ln \ln N}. \tag{2}$$

2.2 Groups

A group G is *additive* if its composition operation is written $+$; multiplicative if written $\times$ (or implied). *Abelian* is a synonym for commutative. In an additive group, we write ax to denote repeated composition of x with itself a times; in a multiplicative group, x^a. Given an element x in G, we say the *order* of x, written $\text{order}_G(x)$, is the smallest positive integer a such that $ax = 0$, where 0 is the identity element of G.

Definition 1 Let G be an additive Abelian group. A *generating set* for G is a set

$$\{(g_1, N_1), \ldots, (g_k, N_k)\},\ g_i \in G, N_i > 1, \tag{3}$$

such that every element $x \in G$ can be written uniquely as

$$x = a_1 g_1 + \cdots + a_k g_k,\ 0 \leq a_i < N_i. \tag{4}$$

We say a generating set is *canonical* if N_{i+1} divides N_i for $1 \leq i < k$. Every Abelian group has canonical generating sets. Furthermore, the sequence $N_1, \ldots, N_k$ is the same for all canonical generating sets. The *rank* of G is the cardinality of its canonical generating sets. Thus we may speak of the r-tuple $(g_1, \ldots, g_r)$, where r is the rank, as a *generating tuple* for G. If the rank is 2, then we have a *generating pair* (g_1, g_2); and if the rank is 1, then we have simply a *generator* g.

Also observe the isomorphism

$$G \cong \mathbf{Z}_{N_1} \times \cdots \times \mathbf{Z}_{N_r}. \tag{5}$$

To every $x \in G$ there corresponds a unique r-tuple. We call this the *index tuple*, and it is defined as

$$\underset{G, g_1, \ldots, g_r}{\text{index}}\ (x) = (a_1, \ldots, a_r) \iff x = a_1 g_1 + \cdots a_r g_r. \tag{6}$$

For groups with rank 2 and 1, we have the *index pair* and *index*. In any group, we also write

$$\underset{G,y}{\text{index}}(x) = a \iff x = ay, \tag{7}$$

for arbitrary y, but this may not be defined for all x.

The following are results of the lower bound on $\varphi(N)$ (equation 2).

Lemma 2 The probability that an element x has maximum order in G is at least

$$\frac{1}{6 \ln \ln N_1}, \tag{8}$$

where $N_1 > 3$ is the maximum order.

Lemma 3 Let G be a group with N elements, and let r be its rank. Then the probability that $(x_1, \ldots, x_r)$ is a generating tuple is at least

$$\frac{1}{6^r (\ln \ln N)^r} \tag{9}$$

assuming each $N_i > 3$.

3 Elliptic curves

Elliptic curves, like many other topics in number theory and algebraic geometry, enjoy a rich history as well as recent applications to computer science. As Cassels writes, "It has exercised a fascination throughout the centuries and the number of isolated results is immense" [Cas66]. The study of elliptic curves has led to a solution of the Congruence problem [Kob84], and a "Riemann hypothesis" [Cho65] [Jol73]. More recently, Lenstra has proposed a novel factoring algorithm using a group law that relates the points of an elliptic curve [Bac85] [Len85]. This same group law is the basis for Miller's "elliptic logarithm" adaptation [Mil85a] of the Diffie-Hellman key exchange protocol [DH76], and for the primality certification of Goldwasser and Kilian [GK86].

3.1 Definition and notation

In simplest form, an *elliptic curve* is the set of (x, y) solutions over a field K to the equation

$$E : y^2 = x^3 + Ax + B, \tag{10}$$

where $A, B \in K, (x, y) \in K^2$, and $4A^3 + 27B^2 \neq 0$. Most elliptic curves can be expressed this way, called *Weierstrass form*. Much study in this century has been devoted to elliptic curves over the fields $\mathbf{C}$, $\mathbf{R}$, $\mathbf{Q}$, $\mathbf{Z}$ [Cas66] [Cho65] [Ful69]. Also of interest are elliptic curves over finite fields and their algebraic closures, $\mathbf{F}_q$ and $\overline{\mathbf{F}_q}$. [BMP75] [Sch85] [Tat74].

An elliptic curve may also be defined in projective coordinates, as

$$E : y^2 z = x^3 + Axz^2 + Bz^3, \tag{11}$$

where $A, B \in K$ and the discriminant

$$\Delta = 4A^3 + 27B^2 \neq 0 \tag{12}$$

in K.

The point $(x, y, 1)$ in projective coordinates corresponds to the point (x, y) in affine coordinates, where 1 is the unit of the field K. The point $(0, 1, 0)$ corresponds to a *point at infinity* on the elliptic curve in affine coordinates.

Let $E(K)$ denote the set of solutions of the curve E over the field K, together with the point at infinity, denoted 0. A well-known result is that $E(K)$ is an abelian group under a composition operation called "tangents and chords." The description of this operation is easiest for $E(\mathbf{R})$. Any line in $\mathbf{R}^2$ intersects the curve E in either zero or three points. (A point of tangency is counted twice, the third point of intersection for a vertical line is considered 0.) The composition of points P and Q, written $P + Q$, is the reflection of the third point colinear with them. Thus 0 is the identity. Figure 1 illustrates this operation.

Most of the group axioms are easily verified; to show $E(\mathbf{R})$ is associative requires certain theorems of algebraic geometry [Ful69]. Since the composition operation can be expressed as a rational polynomial function, it can be generalized from $\mathbf{R}$ to any field. We will assume for analysis of algorithms that we can compose points on an elliptic curve $E(\mathbf{F}_p)$ in time $O(n^2)$, where $n = \log p$.

3.2 Group structure

Lemma 4 $E(\mathbf{F}_p)$ has rank 2.

Proof. This follows from a morphism with $\mathbf{C}/L$, where $\mathbf{C}$ is the complex plane and L is a lattice. Since two generators are sufficient for the lattice, two are sufficient for the elliptic curve. ∎

Lemma 5 If $E(\mathbf{F}_p) \cong \mathbf{Z}_{N_1} \times \mathbf{Z}_{N_2}$, where N_2 divides N_1, then N_2 divides $p-1$.

Proof. This is a fairly deep result. See, for example, Cassel's survey [Cas66]. ∎

Lemma 6 The only points of order 2 on an elliptic curve are 0 and those with y-coordinate 0.

Proof. By definition of composition, if $P = (x, y)$ then $-P = (x, -y)$, since they lie on the same vertical line. Points of order 2 are self-inverse, and thus $P = 0$ and $P = (x, 0)$ are the only solutions. ∎

3.3 Simple case

Definition 7 The *simple case* of elliptic curves consists of those curves over a finite field $\mathbf{F}_p$

$$E : y^2 = x^3 + B \tag{13}$$

for which $p \equiv 2$ modulo 3 and $B \neq 0$.

We show several useful properties.

Lemma 8 Then to every y-coordinate in $\mathbf{F}_p$ there corresponds exactly one point on an elliptic curve $E(\mathbf{F}_p)$ in the simple case.

Proof. For any $y \in \mathbf{F}_p$, the point $(\sqrt[3]{y^2 - b}, y)$ is on the curve. Since $p \equiv 2 \pmod 3$, cube roots are unique and therefore there is exactly one point for each y. ∎
Corollary. $N_p = p + 1$.

Lemma 9 Let $E(\mathbf{F}_p)$ be an elliptic curve in the simple case. Then $E(\mathbf{F}_p)$ is cyclic.

Proof. By lemmas 4 and 5, $E(\mathbf{F}_p) \cong \mathbf{Z}_{N_1} \times \mathbf{Z}_{N_2}$, where $N_1 N_2 = p + 1$ and N_2 divides $p - 1$. Thus N_2 must be either 1 or 2. But $E(\mathbf{F}_p)$ has only two points of order 2, $(\sqrt[3]{-B}, 0)$ and 0, so N_2 must be 1 and the group is cyclic. ∎

4 Pseudo-random number generators

Recent research in computational complexity has led to the notion of a *cryptographically strong pseudo-random bit generator*. Yao formalized this notion in terms in information theory [Yao82], and Blum and Micali gave sufficient conditions for constructing a generator, together with a concrete example using discrete logarithms [BM84]. Later, direct constructions were obtained for generators based on the RSA cryptosystem [ACGS] and the quadratic residuosity problem [BBS83]. Levin made research more formal with his study of weaker sufficient conditions and necessary conditions [Lev85].

A pseudo-random bit generator is interesting in at least two ways. First, it provides a source of randomness indistinguishable in polynomial time from a truly random source, and therefore it can be used reliably in probabilistic algorithms. In fact, Yao shows that the existence of such a generator implies that any randomized polynomial-time algorithm can be simulated by a deterministic sub-exponential-time algorithm [Yao82]. Ajtai and Wigderson generalize these results to probabilistic constant-depth circuits [AW85]. Second, a generator can be used both in public- and private-key cryptosystems; in the latter case, it is the polynomial-time equivalent of the "one-time pad," an ideal, provably secure cryptosystem.

4.1 Sources

We borrow the following definitions, slightly modified, from Yao's paper. We assume that probability distributions are uniform, and therefore refer to sources simply as sets of strings.

Definition 10 A *source* S is a set of strings of equal length. A *source ensemble* $\mathcal{S}$ is a sequence of sources $S_1, S_2, \ldots$. If $\xi(n)$ is the length of the strings in S_n and $\xi(n) < \xi(n+1)$, then the source ensemble is called *uniform*.

Definition 11 The *truly random* source ensemble $\mathcal{R} = R_1, R_2, \ldots$, where each R_n is the set of all strings of length n.

4.2 Statistical tests

Definition 12 A *statistical test* is a probabilistic algorithm that takes as input a string and outputs either 0 or 1.

Let T be a statistical test, and let $\mathcal{S} = S_1, S_2, \ldots$ be a source ensemble. We say that $\mathcal{S}$ *passes the statistical test* T if for all polynomials $Q(n)$, for n sufficiently large,

$$|\Pr[T(x) = 1 \mid x \in S_n] - \Pr[T(x) = 1 \mid x \in R_n]| < \frac{1}{Q(n)}. \tag{14}$$

We say that $\mathcal{S}$ *passes all polynomial-time statistical tests* if it passes all such T that run in time polynomial in the lengths of their input.

4.3 Approximation

Definition 13 Let I be a set of strings, let $b: I \to \{0,1\}$ be a predicate, and let $\epsilon : \mathbf{N} \to [0, 1/2]$ be a function. We say an algorithm $b_\epsilon: I \to \{0,1\}$ *ϵ-approximates* b if for each n,

$$\Pr[b(x) = b_\epsilon(x)] \geq 1/2 + \epsilon(n), \tag{15}$$

where the probability is taken over coin flips in b_ϵ and x of length n in I. Such an algorithm is called an *ϵ-approximator* for b.

Let $T_Q(n)$ be the running time of any algorithm that $1/Q(n)$-approximates b on inputs of length n. We say a b is *unapproximable* if for all polynomials $Q(n)$, $T_Q(n)$ grows faster than any polynomial in n.

4.4 Sufficient conditions

Definition 14 Let I be a set of strings, and let $b: I \to \{0,1\}$ and $f: I \to I$ be functions. Consider the source ensemble $\mathcal{S}$ where S_n is the set of strings

$$b(f^i(s)) \circ b(f^{i-1}(s)) \circ \cdots \circ b(f(s)), \tag{16}$$

where $s \in I$ of length n is chosen uniformly at random. If $\mathcal{S}$ is uniform, polynomial, and passes all polynomial-time statistical tests, then then $\langle I, f, b\rangle$ is called a *Blum-Micali pseudo-random bit generator*.

Theorem 15 Let I be a set of strings, and let $b: I \to \{0,1\}$ and $f: I \to I$ be functions. Also let I_n be the subset of I containing strings of length n. Then $\langle I, f, b\rangle$ is a Blum-Micali pseudo-random bit generator, if

1. *(friendship)* $b(f(x))$ is computable in time polynomial in $|x|$.
2. *(stability)* f is a permutation, $|f(x)| = |x|$, and $f(x)$ is computable in time polynomial in $|x|$.
3. *(accessibility)* There is an algorithm that, given an integer n, selects elements $x \in I_n$ uniformly (if any exist) in time polynomial in $|x|$.
4. *(unapproximability)* b is unapproximable.

Remark. Yao and Levin propose conditions less strict than these. In particular, Yao replaces I with an sequence of probability distributions. Levin shows that f need not be a permutation.

One condition which we will make less strict is accessibility, by allowing the algorithm to have negligible error. That is, the algorithm may output $x \notin I_n$ with probability asymptotically less than any inverse polynomial in n.

5 A new tool for finding generators

Finding a generating set for an Abelian group G is important both in the definition of pseudo-random bit generators, and in applications such as the Diffie-Hellman key exchange protocol. There are several parts to the problem:

1. finding the cyclic decomposition of G;
2. computing the orders of elements; and
3. testing that elements are "linearly independent."

We present in this section a new tool for solving the second part of the problem. Let G be an Abelian group, let N be its order, and let $n = \log N$. The main result is an algorithm that uses polynomially-many (in n) group operations in G to compute the order of an element $x \in G$, with negligible probability of error. For this method to be applied to a group, the group must have an efficient (*i.e.*, polynomial-time) algorithm to perform group operations, and its order (number of elements) must be known. Our order-computing algorithm utilizes Lenstra's new factorization algorithm [Bac85] [Len85].

The outline of this section is as follows. An intermediate result, a partial factorization algorithm using Lenstra's algorithm, appears in section 5.1. We then apply this algorithm in section 5.2 to show how to approximate the order of an element $x \in G$. Finally, we present an example of a generating algorithm that uses the results in section 5.3.

5.1 Partial factorization

We now present a polynomial-time algorithm that extracts all "small" prime factors of an integer N, using Lenstra's factorization algorithm. Throughout the discussion, assume that N is fixed, and let $n = \log N$. Recall that Lenstra's algorithm yields a factor p of N in expected time $O(L(p)^{\sqrt{2}+O(1)}n^3)$, time, where

$$L(p) \stackrel{\text{def}}{=} e^{\sqrt{\ln p \ln\ln p}}. \tag{17}$$

We will make much use of the key feature of Lenstra's algorithm, which is that its running time depends on the smallest prime factor. Pomerance observes that this feature can "usually" be applied to determine whether a number is smooth with respect to some bound k, *i.e.*, whether all its prime factors are less than or equal to k [Pom85]. Using this technique, it is also possible to find all prime factors less than or equal to k.

It is not known whether Lenstra's algorithm is able to find all prime factors. This depends on a conjecture concerning the distribution of smooth numbers. If this conjecture is not true, then there may be certain primes that Lenstra's algorithm never finds. In this case, the probability of error is no longer negligible. We assume the conjecture is true for the course of discussion. Indeed, Lenstra and others call this assumption "reasonable" [Len85] [Bac85].

This application of Lenstra's algorithm is significant, because to find small factors or to test smoothness using previous methods would require time proportional to k or to $\sqrt{k}$. With the new technique, it is possible to find small factors in polynomial time for much larger k, asymptotically greater than any polynomial in n. Algorithm *small-factors*, in figure 2, shows one way to do this, and the following theorem formalizes the observation.

Theorem 16 Let N be an integer and let $n = \log N$. Algorithm *small-factors* finds all factors of N less than or equal to $n^{\ln n/\ln\ln n}$ in time polynomial in n, with negligible error.

Proof. We prove the theorem in three steps. First, we expand $L(k)$ to determine the expected time to find a single small factor of N. Second, we show how to make the probability of error negligible for a single small factor. Finally, we analyze the expected time to find all small factors using algorithm *small-factors*.

Expanding $L(k)$ for $k = n^{\ln n/\ln\ln n}$ gives

$$L(k) = e^{\ln n\sqrt{(2\ln\ln n - \ln\ln\ln n)/\ln\ln n}}. \tag{18}$$

Thus, for sufficiently large n,

$$L(k) \le e^{\sqrt{2}\ln n} = n^{\sqrt{2}}, \tag{19}$$

and $L(k)^{\sqrt{2}} = O(n^2)$. Therefore, the expected time to find a factor smaller than $n^{\ln n/\ln\ln n}$, if one exists, is $O(n^{5+O(1)})$.

If M is sufficiently large in step 2 of the algorithm, then with high probability we find a factor smaller than $n^{\ln n/\ln\ln n}$, if one exists. Consider the process of finding such a factor as a Bernoulli process, since each trial in Lensta's algorithm has the same probability of success, and these probabilities are independent. Let l_1 be a random variable representing the time between successes; then

$$E[l_1] = O(n^{5+O(1)}). \tag{20}$$

Using a Poisson approximation [Dra67] we have,

$$\Pr[l_1 > nE[l_1]] = e^{-n}. \tag{21}$$

If we set the M as $O(n^{6+O(1)})$, then the probability of error is negligible.

We conclude the analysis as follows. Since N has at most n factors counting multiplicity, we run step 2 at most n times. Each takes time $O(n^{6+O(1)})$, so the total running time is $O(n^{7+O(1)})$, which is polynomial. The probability of error is e^{-n} is negligible. ∎

5.2 Approximating the order of an element

We now present a probabilistic algorithm that determines the order of an element of an Abelian group in polynomial expected time with negligible probability of error. First we discuss an algorithm to determine the order of an element x, given the complete factorization of N, the order of G; then we analyze a similar algorithm in the case when only partial factorization is known—the small factors determined in section 5.1.

Without loss of generality, we assume that G is cyclic. If it is not cyclic, then the analysis and results would apply to the subgroup of G generated by x, where N is the order of the subgroup.

5.2.1 Determining order with complete factorization

Algorithm *with-complete-factorization*, in figure 3, shows how to determine the order of $x \in G$ given all factors of N. We prove two lemmas about this algorithm: first, that it is correct; and second, that it runs in polynomial time.

Lemma 17 Algorithm *with-complete-factorization* is correct.

Proof. By a corollary of the Chinese Remainder Theorem, each $x \in G$ corresponds to a tuple in $\mathbf{Z}_{p_1} \times \cdots \times \mathbf{Z}_{p_r}$, i.e.

$$x \mapsto (x_1, \ldots, x_r), x_i \in \mathbf{Z}_{p_i^{\alpha_i}}. \tag{22}$$

For each i, computing $(N/p_i^{\alpha_i})x$ has the bijection

$$\frac{N}{p_i^{\alpha_i}} x \mapsto (0, \ldots, x_i, \ldots, 0). \tag{23}$$

Finding the smallest $p_i^{\beta_i}$ for which $(Np_i^{\beta_i}/p_i^{\alpha_i})x = 0$ is the same as finding the order of x in $\mathbf{Z}_{p_i^{\alpha_i}}$. Since the p_i are pairwise relatively prime, the order of x is the product of the orders of x_i in $\mathbf{Z}_{p_i^{\alpha_i}}$, and the algorithm is correct. ∎

Lemma 18 Algorithm *with-complete-factorization* uses polynomially-many (in n) group operations.

Proof. Since N has has no more that $\log N$ prime factors, counting multiplicity, $\alpha_1 + \cdots + \alpha_k \leq n$. Thus the number of computations of the form $(Np_i^{\beta_i}/p_i^{\alpha_i})x$ is at most n. Each computation requires $O(n)$ group operations, using successive doubling. Thus, $O(n^2)$ group operations are needed, in all. ∎

5.2.2 Determining order with partial factorization

Suppose only partial information is known about the prime factors of N. Consider the modified version of algorithm *with-complete-factorization*, called *with-partial-factorization* (figure 4). Again, we prove two lemmas about the algorithm: first, that its probability of error is negligible; and second, that it runs in polynomial time.

For the first lemma, assume that $m = p_{k+1}^{\alpha_i} \cdots p_l^{\alpha_l}$, p_i prime, $p_i < p_{i+1}$, and without loss of generality assume $p_k < p_{k+1}$. One may ask, what is the probability that the algorithm errs, *i.e.*, outputs o, where $o \neq \mathrm{order}_G(x)$?

Lemma 19 Algorithm *with-partial-factorization* errs with probability at most n/p_{k+1}, where p_{k+1} is the smallest prime dividing m.

Proof. Suppose $\mathrm{order}_G(x) = p_1^{\beta_1} \cdots p_l^{\beta_l}$, $0 \leq \beta_i \leq \alpha_i$. How does the algorithm behave? There are three cases.

1. $p_{k+1}^{\beta_{k+1}} \cdots p_l^{\beta_l} = m$; correct answer, $o = p_1^{\beta_1} \cdots p_k^{\beta_k} m = \mathrm{order}_G(x)$.
2. $p_{k+1}^{\beta_{k+1}} \cdots p_l^{\beta_l} = 1$; correct answer, $o = p_1^{\beta_1} \cdots p_k^{\beta_k} = \mathrm{order}_G(x)$.
3. $1 < p_{k+1}^{\beta_{k+1}} \cdots p_l^{\beta_l} < m$; incorrect answer, $o = p_1^{\beta_1} \cdots p_k^{\beta_k} m \neq \mathrm{order}_G(x)$.

To find the probability of the third case, consider the subgroup S of elements whose orders divide N/m, and the quotient group G/S. Since G is cyclic, the order of G/S is m, and $G/S \cong \mathbf{Z}_m$. The number of elements of order neither m nor 1 in G/S is $m - \varphi(m) - 1$. The probability of an incorrect answer is exactly the probability that an element of G/S has order neither m nor 1, or $1 - \varphi(m)/m - 1/m$.

An upper bound for the probability can be determined by expanding $\varphi(m)$ and observing that $l - k \leq n$, since each index represents a distinct prime factor.[2] Expanding $\varphi(m)$ gives

$$1 - \frac{\varphi(m)}{m} - \frac{1}{m} = 1 - \prod_{k+1 \leq i \leq l} \frac{p_i - 1}{p_i} - \frac{1}{m}. \tag{24}$$

We can compute an upper bound on this value, as follows:

$$1 - \prod_{k+1 \leq i \leq l} \frac{p_i - 1}{p_i} - \frac{1}{m} \leq 1 - \left(\frac{p_{k+1} - 1}{p_{k+1}}\right)^{l-k} \leq \frac{l-k}{p_{k+1}} \leq \frac{n}{p_{k+1}}. \tag{25}$$

This is the probability of error stated in the lemma, so we are done. ∎

Lemma 20 Algorithm *with-partial-factorization* uses polynomially-many (in n) group operations.

Proof. The proof is similar to that for algorithm *with-complete-factorization*. The running time is no greater, since the partial factorization of N has no more factors than the complete factorization. Therefore, $O(n^3)$ group operations are sufficient. ∎

The two lemmas lead to our main result, that

Theorem 21 Let G be an Abelian group with an efficient composition operation, let x be an element of G, and let $n = \log N$ where N is the order of G. Using Lenstra's factorization algorithm, given x and N, it is possible to compute the order of x in G in time polynomial in n with negligible error.

Proof. Let $p_{k+1} = n^{\ln n / \ln \ln n}$; then algorithm *small-factors* can compute a partial factorization of the form needed for algorithm *with-partial-factorization* in time polynomial in n, with negligible error. Assuming N is in this partially-factored form, the probability of error in computing the order is $n^{1 - \ln n / \ln \ln n}$, which is negligible. By lemma 20, the algorithm runs in time polynomial in n, if G has an efficient composition operation. ∎

By lemma 2, the subset of G consisting of those elements of maximum order is non-negligible. Thus we have a corollary to the theorem.

Corollary. Let G, x, n and N be as in theorem 21. Using Lenstra's factorization algorithm, it is possible to test whether an element x of G has maximum order in time polynomial in n with negligible error.

[2] Knuth and Trabb-Pardo observe that an integer N has at most $n/\log n$ distinct factors [KT76]; this is a stronger bound for $l - k$. One may also show that $l - k \leq n/\log p_{k+1}$. However, the weak upper bound is adequate for the proof.

5.3 An example

We now present an example of a generating algorithm. The algorithm, *cyclic-generating-test* (see figure 5, operates on the family of multiplicative groups modulo p, where p is a prime, and tests whether an element is a generator. The test is a straightforward application of algorithm *with-partial-factorization*. If G has N elements, then we can apply *with-partial-factorization* to answer, with negligible error, the question,

"Is $\text{order}_G(x) = N$?"

If the answer is "yes," then with high probability, x generates G. This is a result of the corollary to theorem 21. (It is interesting to note that if the answer is "no," we cannot make the claim of high probability, because the subset of G consisting of those elements not of maximum order may be negligible.)

6 A new oracle proof technique

This section presents a new method of proving the equivalence of predicting the value of a single bit of a discrete logarithm, and computing the logarithm itself. If computing the logarithm is a "hard problem," then the method is strong enough to prove that $O(\log n)$ bits are simultaneously unapproximable, where n is the length of the logarithm. The method requires only that the group is Abelian, and that its cyclic decomposition is known.

The novel idea in the oracle proof technique is a notion of "correlation" that provides a measure of closeness to the correct logarithm. This measure can be used to obtain information about the range of values an index may take; by refining this information, the index can be found.

Previous methods for proving such theorems used properties of a group which are not always available. In particular, the Blum-Micali proof method [BM84] (and others based on it, *e.g.*[LW83]) required that the order of the group be even and used "quadratic residuosity" and square root operations. Although there is a square root operation for elliptic curves (since composition is defined by rational polynomials), the order of the group may be odd, and therefore previous methods are not applicable. Furthermore, elliptic curve groups may be non-cyclic, a condition not encountered in previous proof methods.

6.1 Preliminaries

Definition 22 Let G be an additive, Abelian, cyclic group with N elements, and let g be a generator of G. (Thus $G \cong \mathbf{Z}_N$.) The *half bit* of the index $b_{G,g,0}: G \to \mathbf{Z}_2$ is defined as

$$b_{G,g,0}(x) = \begin{cases} 1, & \text{if } \text{index}_{G,g}(x) \geq N/2; \\ 0, & \text{if } \text{index}_{G,g}(x) < N/2. \end{cases} \tag{26}$$

The half bit can also be called the *most significant bit* of the index. Other "most significant bits" can be defined recursively; the i-th most significant bit $b_{G,g,i}: G \to \mathbf{Z}_2$ is defined as

$$b_{G,g,i}(x) = b_{G,g,i-1}(2x). \tag{27}$$

In the discussion following, the group G and the generator g are assumed fixed, and the shorter notation $b_i(x)$ is used. The main theorem of this section is

Theorem 23 Let G be a cyclic group with an efficient composition operation, let g be a generator of G, and let $n = \log N$ where N is the order of G. Let $\epsilon > 0, 0 \leq i < n$. The following problems are probabilistically reducible to each other in time polynomial in n, ϵ^{-1} and 2^i:

1. Compute $\text{index}(x)$.
2. Compute $b_i(x)$ correctly for $1/2 + \epsilon$ of $x \in G$.

The reduction of the second problem to the first is obvious. The following sections discuss the reduction in the other direction: the "oracle proof technique." The reduction is so named because such theorems traditionally have been proved by constructing an algorithm to compute $\text{index}(x)$ using an oracle that "guesses" one bit of the index with probability $1/2 + \epsilon$.

Computing $b_i(x)$ correctly for a fraction of $x \in G$ can be described more formally, in terms of approximation (see section 4). In particular, let $b_{i,\epsilon}$ be an ϵ-approximator to b_i. Recall that G and g are fixed. We use the notion of approximation to define *correlation*.

Definition 24 Let x_1 and x_2 be elements of G. The *i-th bit correlation* of x_1 and x_2, $\rho_{i,\epsilon}: G \times G \to [-1/2, 1/2]$, is defined as

$$\rho_{i,\epsilon}(x_1, x_2) \stackrel{\text{def}}{=} \Pr[b_i(x_1 + rg) = b_{i,\epsilon}(x_2 + rg)] - \frac{1}{2}, \tag{28}$$

where r is a uniformly-distributed random variable taking integer values between 0 and $N - 1$.

Lemma 25 The following are true for all integers k and elements x_1 and x_2 of G:

$$\begin{aligned}
&(\textit{approximation}) && \rho_{i,\epsilon}(x_1, x_1) \geq \epsilon \\
&(\textit{periodicity}) && |\rho_{i,\epsilon}(x_1, x_2) - \rho_{i,\epsilon}(x_1 + \lfloor k\frac{N}{2^i} \rfloor g, x_2)| \leq \frac{2^{i+1}}{N} \\
&(\textit{symmetry}) && |\rho_{i,\epsilon}(x_1, x_2) + \rho_{i,\epsilon}(x_1 + \lfloor k\frac{N}{2^i} + \frac{N}{2^{i+1}} \rfloor g, x_2)| \leq \frac{2^{i+1}}{N},\ 0 \leq k < 2^i \\
&(\textit{locality}) && |\rho_{i,\epsilon}(x_1 + g, x_2) - \rho_{i,\epsilon}(x_1, x_2)| \leq \frac{2^{i+1}}{N}.
\end{aligned} \tag{29}$$

Proof. *(Approximation)* Observe that by the definitions of correlation and approximation,

$$\rho_{i,\epsilon}(x, x) = \Pr[b_i(x + rg) = b_{i,\epsilon}(x + rg)] - \frac{1}{2} \geq \epsilon. \tag{30}$$

(Periodicity, symmetry and locality) View the correlation as a piece-wise constant function on a circle, taking values 0 and 1. (See figure 6.)

Comparing $\rho_{i,\epsilon}(x_1, x_2)$ to $\rho_{i,\epsilon}(x_1 + \Delta x_1 g, x_2)$ amounts to rotating the circle by Δx_1 and comparing it to itself. However, since Δx_1 is an integer, the rotation may cause regions to overlap slightly. In other words, one comparison in each contiguous region of 0's or 1's may yield an "erroneous" value. Since there are at most 2^{i+1} such regions, we arrive at the stated limits. ∎

6.2 Algorithms and analysis

We now present three algorithms to show the proof: algorithms *estimate-correlation* in figure 7, *decide-square-roots* in figure 8, and *compute-index* in figure 9. We also analyze their running times and probabilities of error.

6.2.1 Estimating correlation

By choosing random values for r, it is possible to estimate the correlation very well. First, we present an algorithm that does this estimation; then we analyze its running time and probability of error. The algorithm, *estimate-correlation*, appears in figure 7.

Lemma 26 Let δ be a positive real number, and let $\xi_{i,\epsilon}(cg, x)$, a random variable, be the estimate produced by algorithm *estimate-correlation* using M samples. If $M > \epsilon^{-1}\delta^{-1/2}$, then for all c and x,

$$\Pr[|\xi_{i,\epsilon}(cg, x) - \rho_{i,\epsilon}(cg, x)| > \frac{\epsilon}{2}] < \delta. \tag{31}$$

Proof. Let c and x be fixed, and write $\xi = \xi_{i,\epsilon}(cg, x)$. Then, since the estimate is the average value of a Bernoulli process, we have

$$E[\xi] = \rho_{i,\epsilon}(cg, x); \tag{32}$$

$$\sigma_\xi^2 = \frac{1}{M} E[\xi](1 - E[\xi]) \leq \frac{1}{4M}. \tag{33}$$

By the weak law of large numbers,

$$\Pr[|\xi - E[\xi]| > \frac{\epsilon}{2}] \leq \frac{\sigma_\xi^2}{M(\epsilon/2)^2} \leq \frac{1}{M^2\epsilon^2} < \delta, \tag{34}$$

as stated in the lemma. ∎

It is of significance to note that the upper bound on the probability of error is independent of the values of c and x, and of any other estimations made. This is because the estimation randomizes over r, and δ is an upper bound regardless of the value being estimated.

As an application of this lemma, we show that it is possible to estimate the correlation very well in polynomial time.

Lemma 27 Using polynomially-many (in n, ϵ^{-1} and δ^{-1}) coin flips, group operations and oracle invocations, algorithm *estimate-correlation* errs by $\epsilon/2$ or more with probability at most δ.

Proof. Selecting $M = \epsilon^{-1}\delta^{-1/2}$, we use $O(\epsilon^{-1}\delta^{-1/2}n)$ coin flips and group operations, and $O(\epsilon^{-1}\delta^{-1/2})$ oracle invocations, as follows. The first step of the algorithm computes M n-bit numbers, requiring $O(Mn)$ coin flips. The second step uses the oracle once per trial, or M times, and each computation of $r_j g$ requires $O(n)$ group operations, using successive doubling. The time to compute $b_i(cg + r_j g)$ (since c and r_j are known) and $x + r_j g$ is comparatively small. ∎

6.2.2 Deciding square roots

We now present an algorithm to "decide square roots," a key method in determining the logarithm. Let $[a, b]$ denote the interval between a and b, inclusive, where a and b are real numbers. When a and b are sufficiently close, it is possible answer the question,

"Given that $\text{index}(2^{i+1}x) \in [2^{i+1}a, 2^{i+1}b]$, is $\text{index}(2^i x)$ in $[2^i a, 2^i b]$ or in $[2^i a + N/2, 2^i b + N/2]$?"

Lemma 28 Let $[a, b]$ be an interval with

$$0 \le a \le b \le a + \frac{N\epsilon}{2^{i+1}} - 3 < \frac{N}{2^{i+1}}, \tag{35}$$

and let c be the integer closest to $(a + b)/2$. Let δ be any positive real number. If $\text{index}(2^{i+1}x) \in [2^{i+1}a, 2^{i+1}b]$ and $\text{index}(2^i x) \in [2^i a, 2^i b]$, then for $M > \epsilon^{-1}\delta^{-1/2}$,

$$\Pr[\xi_{i,\epsilon}(cg, x) < 0] < \delta. \tag{36}$$

Proof. The hypothesis $\text{index}(2^i x) \in [2^i a, 2^i b]$ implies

$$\text{index}(x) \in \bigcup_{0 \le k < 2^i} [a + k\frac{N}{2^i}, b + k\frac{N}{2^i}]. \tag{37}$$

Then for some value of k, it is true that

$$|c + \lfloor k\frac{N}{2^i} \rfloor - \text{index}(x)| \le \frac{b - a + 1}{2}. \tag{38}$$

Furthermore, because $\rho_{i,\epsilon}$ is periodic,

$$|\rho_{i,\epsilon}(cg + k\frac{N}{2^i}g, x) - \rho_{i,\epsilon}(cg, x)| \le \frac{2^{i+1}}{N} \tag{39}$$

for all k. By lemma 25,

$$|\rho_{i,\epsilon}(cg, x) - \rho_{i,\epsilon}(\text{index}(x)g, x)| \le \frac{2^{i+1}}{N} \cdot \frac{b - a + 1}{2} + \frac{2^{i+1}}{N} \le \frac{\epsilon}{2}. \tag{40}$$

Consequently, we have a lower bound on the correlation,

$$\rho_{i,\epsilon}(cg, x) > \frac{\epsilon}{2}. \tag{41}$$

Using the estimation algorithm with $M > \epsilon^{-1}\delta^{-1/2}$, we arrive at

$$\Pr[\xi_{i,\epsilon}(cg, x) < 0] < \delta, \tag{42}$$

proving the lemma. ∎

Using these observations, we construct an algorithm *decide-square-roots* (see figure 8) to answer the initial question.

Lemma 29 Given x, and assuming that $\text{index}(2^{i+1}x) \in [2^{i+1}a, 2^{i+1}b]$, algorithm *decide-square-roots* is correct with probability $> 1 - \delta$, where δ is the error bound in estimating the correlation.

Proof. The probability of error is less than δ, whether $\text{index}(x) \in [2^i a, 2^i b]$ or $\text{index}(x) \in [2^i a + N/2, 2^i b + N/2]$. Since these are the only intervals in which the index may lie, the algorithm is correct with probability $> 1 - \delta$. Furthermore, by lemma 27, the algorithm runs in time polynomial in n, ϵ^{-1} and $\delta^{-1/2}$. ∎

6.2.3 Computing the index

Algorithm *compute-index* (figure 9) computes the index of x by successively restricting the range of indexes in which $\text{index}(2^{j+i}x)$ may lie. To show that this algorithm runs in polynomial time, we determine the probability that the restricted range for $\text{index}(x)$ is incorrect, assuming an initial range for $\text{index}(2^{n+i}x)$ is correct. By showing the probability is sufficiently small, we determine that in few iterations of the main part of the algorithm, we can find the index.

Lemma 30 Algorithm *compute-index* finds $\text{index}(x)$ using polynomially-many (in n, ϵ^{-1} and 2^i) expected coin flips, group operations and oracle invocations.

Proof. Suppose the interval $[a, b]$ selected in step 1 is correct. By lemma 29, the probability of choosing the next $[a, b]$ correctly in step 2 is at least $1 - \delta$. Thus, the probability of success in n consecutive choices is

$$(1 - \delta)^n \geq 1 - n\delta, \tag{43}$$

since probabilities are independent by the definition of the estimation algorithm.

If $[a, b]$ contains $\text{index}(2^i x)$ upon entering step 3, then $[a, b]$ contains exactly one integer, because step 2 reduces $b - a$ by a factor of 2 each time. Thus

$$b - a \leq \frac{1}{2^n}\left(\frac{N\epsilon}{2^{i+1}} - 3\right) < \frac{N\epsilon}{2^{n+i+1}} < 1 \tag{44}$$

upon entering step 3. Step 4 then recovers the index.

To determine the running time, observe that there are about $2^{i+1}\epsilon^{-1}$ intervals, so the expected number to search is $2^i\epsilon^{-1}$. Further, if $\delta = 1/2n$, then the probability of success in equation 43 is at least $1/2$, and thus two trials of step 2 are expected given the correct interval. In all, $2^{i+1}\epsilon^{-1}$ trials of steps 1–3 are expected. Since algorithm *decide-square-roots* and all other operations are polynomial in n, ϵ^{-1} and $\delta^{-1/2}$, algorithm *compute-index* is polynomial in n, ϵ^{-1} and 2^i. ∎

6.3 Proof of main theorem

Finally, we complete the proof of the main theorem. Lemma 30 shows that an ϵ-approximator for b_i can be used to compute $\text{index}(x)$ in polynomially-many coin flips, group operations and oracle invocations. If the group has an efficient composition operation, we have a probabilistic polynomial-time reduction from index computation to ϵ-approximation, and the theorem is proved. ∎

6.4 Extensions

The oracle proof technique is easily modified to handle non-cyclic groups and to show the simultaneous security of $O(\log n)$ bits.

7 Simple case of elliptic curves

In a restricted class of elliptic curves, it is possible to construct a pseudo-random bit generator in a relatively straightforward manner. This restricted class has certain properties that make the construction and proofs easy. The intent of discussing this class is to provide a simple application of the sufficient conditions for pseudo-random bit generation, which will set a better foundation for the general case in section 8.

7.1 Statement of theorem

Definition 31 Let $E(\mathbf{F}_p)$ be an elliptic curve in the simple case (section 3.3), and let G be a generator. Then $\langle E(\mathbf{F}_p), G\rangle$ is a *curve-generator pair.*

We say an algorithm *solves the elliptic logarithm problem* for a curve-generator pair $\langle E(\mathbf{F}_p), G\rangle$ if for every $P \in E(\mathbf{F}_p)$ it computes $\text{index}_{E(\mathbf{F}_p),G}(P)$ correctly. Let $Q(n)$ be a polynomial, and let $T_Q(n)$ be the running time of any algorithm that solves the elliptic logarithm problem for at least a fraction $1/Q(n)$ of all curve-generator pairs, where n is the length of p. By Miller's arguments [Mil85a], we have

Conjecture 32 (Simple elliptic logarithm intractability assumption) $T_Q(n)$ grows faster than any polynomial in n.

This leads to our main theorem. Let an *instance of the simple case* be a tuple

$$\langle E(\mathbf{F}_p), G, P\rangle, \tag{45}$$

where $E(\mathbf{F}_p)$ is an elliptic curve in the simple case, G generates $E(\mathbf{F}_p)$, and P is a point on the curve. The set of all instances where p is an n-bit prime is denoted I_n.[3]

Theorem 33 Let I be the set of instances in the simple case. Let f and b be defined for an instance $s = \langle E(\mathbf{F}_p), G, P\rangle$ as

$$f(s) = \langle E(\mathbf{F}_p), G, \psi(P)\rangle, \tag{46}$$

where

$$\psi(P) = \phi(P)G \text{ and } \phi(P) = \begin{cases} y, & \text{if } P = (x, y); \\ p, & \text{otherwise,} \end{cases} \tag{47}$$

and

$$b(s) = b_{E(\mathbf{F}_p),G,0}(P). \tag{48}$$

Under conjecture 32, $\langle I, f, b\rangle$ is a Blum-Micali pseudo-random bit generator.

Proof. We prove the theorem by showing that the four sufficient conditions for a pseudo-random bit generator (section 4.4) are satisfied. Specifically, we show that b and f are friendship functions, f is a stable, b is accessible, and b is unapproximable. The proofs for each of these parts follow.

7.2 Friendship

Friendship is the easiest of the four parts to show. In particular, we simply prove

Lemma 34 There is an algorithm that computes bf in the simple case in polynomial time.

Proof. Let $s = \langle E(\mathbf{F}_p), G, P\rangle$ be an instance. Using the definitions of f and b, we can write bf as

$$bf(s) = \begin{cases} 1, & \text{if } \text{index}_{E(\mathbf{F}_p),G}(\psi(P)) \geq (p+1)/2; \\ 0, & \text{otherwise.} \end{cases} \tag{49}$$

Since $\text{index}_{E(\mathbf{F}_p),G}(\psi(P)) = \phi(P)$, substituting the definition of ϕ leads to

$$bf(s) = \begin{cases} 1, & \text{if } P = 0; \\ 1, & \text{if } P = (x, y) \text{ and } y \geq (p+1)/2; \\ 0, & \text{otherwise.} \end{cases} \tag{50}$$

This is easily computed in time polynomial in n. ∎

[3] A data structure representing an instance as would probably contain n, p, B, and the x- and y-coordinates of G and P (or the special symbol "0"). We write $\langle E(\mathbf{F}_p), G, P\rangle$ for convenience.

7.3 Stability

We base this on two intermediate results.

Lemma 35 ϕ is a bijection of $E(\mathbf{F}_p)$ and $\{0,\ldots,p\}$ in the simple case.

Proof. By lemma 8 each point has a unique y-coordinate. Therefore points are mapped to unique elements of $\{0,\ldots,p\}$. ∎

Lemma 36 ψ is a permutation of $E(\mathbf{F}_p)$ in the simple case.

Proof. Since G generates $E(\mathbf{F}_p)$, the set

$$\{aG \mid 0 \le a < p+1\} \tag{51}$$

contains all points on the curve. Thus ψ on input P generates $E(\mathbf{F}_p)$ based on $\phi(P)$. Since ϕ is a bijection into $\{0,\ldots,p\}$, the curve is completely generated, and ψ is a permutation. ∎

Lemma 37 (Stability) In the simple case, for each n, f is a permutation on I_n. Further, there is an algorithm that computes f in polynomial time.

Proof. The first part is true by definition of f, because ψ permutes $E(\mathbf{F}_p)$. The second part is true because computing f requires $O(\log \phi(P)) = O(n)$ group operations, which is polynomial in n. ∎

7.4 Accessibility

To show that the predicate b is accessible, we construct an algorithm and analyze it. The algorithm is probabilistic and produces elements of I_n with uniform probability in polynomial expected time. The algorithm, called *simple-accessibility*, appears in figure 10. We prove two lemmas about this algorithm, thereby showing accessibility.

We first recall the result of Bach [Bac83] also used in the Blum-Micali accessibility proof. (There it was used to produce $p-1$ in factored form.)

Lemma 38 (Bach, 1983) There is a probabilistic polynomial-time algorithm that takes as input n and outputs an integer of length n, together with its factorization. The integers are uniformly distributed.

Lemma 39 Algorithm *simple-accessibility* produces elements x of I_n in the simple case with uniform probability.

Proof. Since failure at steps 2–5 leads to "go to 1," it is sufficient to show that each step selects p, E, G, or P with uniform probability among the acceptable values for that component. Using Bach's algorithm all $p+1$ are equally likely; thus, all primes $p \equiv 2 \bmod 3$ are as well. Since all legal B are equally likely for a given p, all curves E are also. Algorithm *cyclic-generating-test* accepts all generators of $E(\mathbf{F}_p)$, because the factorization of $p+1$ is known. Thus, since ϕ is a bijection (lemma 35), generators G and points P are selected with equal likelihood. ∎

Lemma 40 Algorithm *simple-accessibility* outputs an element $x \in I_n$ in the simple case in polynomial (in n) expected time.

Proof. The expected number of trials is $O(n \log n)$, as follows. Step 1 always succeeds. By the prime number theorem, and assuming that half of all primes are congruent to 2 mod 3, the probability of success in step 2 is $1/O(n)$. In step 3, finding a B has probability at least $1/2$. In step 4, finding an a has probability at least $1/2$; a generator, $1/O(\log n)$, by lemma 3. In step 5, finding an a has probability at least $1/2$. The probability of success in every step is $1/O(n \log n)$, leading to the expected value given above.

The running time for one trial is $O(n^6)$, as follows. Bach's algorithm runs in time $O(n^6)$. Testing primality is $O(n^4)$, using the technique of Solovay and Strassen [SS77]. Checking that G is a generator, with the factorization of $p+1$ known, takes $O(n^2)$ group operations, or time $O(n^4)$, by lemma 18. Computing ϕ^{-1} involves taking cube roots modulo p, which requires time $O(n^3)$.

This leads to an expected running time of $O(n^7 \log n)$, which is polynomial in n. ∎

7.5 Unapproximability

Let A be some algorithm that computes b correctly with probability at least $1/2 + 1/Q(n)$ on elements of I_n, and let $T(n)$ be its running time.

Lemma 41 There is a fraction $1/2Q(n)$ of all curve-generator pairs $(E(\mathbf{F}_p), G)$, where p is an n-bit prime, such that algorithm $A(\langle E(\mathbf{F}_p), G, \cdot \rangle)$ $1/2Q(n)$-approximates $b_{E(\mathbf{F}_p),G,0}(\cdot)$.

Proof. This follows from a counting argument. The main idea is that the probability is minimized when A is entirely correct on some curve-generator pairs, and nearly a $1/2Q(n)$-approximator on the rest. For every $E(\mathbf{F}_p)$ where p is an n-bit prime, we have

$$2^{n-1} + 1 \leq N_p \leq 2^n. \tag{52}$$

If the curves in the curve-generator pairs on which A is correct contain the maximum number of elements, and the rest contain the minimum number, then we arrive at the stated probability. ∎

Using algorithm A as the ϵ-approximator, we can compute index(x) with algorithm *simple-index* (figure 11). See figure 9 for algorithm *compute-index*.

Lemma 42 (Unapproximability) The predicate b in the simple case is unapproximable.

Proof. By theorem 23, algorithm *simple-index* computes the index correctly for the fraction of curve-generator pairs for which A is an $1/2Q(n)$-approximator in lemma 41. Furthermore, it does so in time polynomial in n, $Q(n)$ and $T(n)$. Suppose b is not unapproximable, and $T(n)$ is a polynomial in n. Then the algorithm solves the elliptic logarithm problem in time polynomial in n for the $1/2Q(n)$ fraction. This contradicts conjecture 32, and therefore b is unapproximable. ∎

8 General case of elliptic curves

The pseudo-random bit generator for the general case is like that for the simple case, but we use a pair of elliptic curves to define the friendship function. The general case is described in detail in the author's dissertation. The following are the major differences between the general case and the simple case.

- An instance of the general case is a tuple containing two elliptic curves, two generators for each, and a point which may be on either curve. The curves are related by a transformation called *twisting*. This construction is essential to the definition of the friendship function f.
- The proof of accessibility requires a much more complicated algorithm than in the simple case. Two particular problems are that the group $E(\mathbf{F}_p)$ is not necessarily cyclic in the general case, and that the complete factorization of its order N_p is not known. The problems are solved by using the *Weil pairing* (for which Miller has recently developed a polynomial-time algorithm [Mil85b]), and algorithm *with-partial-factorization* (section 5.2).
- Since two elliptic curves are involved, the counting arguments involved in proving unapproximability are more difficult than in the simple case, though not unreasonable.

9 Conclusion

We propose several extensions to our research.

9.1 Reduction from discrete logarithm

The discrete logarithm problem stands on its own as a hard mathematical problem. Unlike quadratic residuosity and inverting RSA, it does not reduce easily to another hard problem, such as factoring. Whether it reduces to the elliptic logarithm problem is an interesting open problem.

9.2 Elliptic curves with factored order

The results in the general case (section 8) depend on an algorithm to compute the partial factorization of the order of the group. While this is sufficient for the generation of pseudo-random bits, it would be more elegant to use a completely-factored order, as Blum and Micali do [BM84]. This would require an algorithm like Bach's [Bac83] to generate an elliptic curves together with the factorization of its order.

- Is there a polynomial-time algorithm which, on input n, outputs an elliptic curve $E(\mathbf{F}_p)$ together with the factorization of its order, where $E(\mathbf{F}_p)$ is selected with uniform probability among all curves where p is an n-bit prime.

9.3 Subexponential elliptic logarithm algorithm

Adleman's algorithm for computing discrete logarithms [Adl79] was a breakthrough in 1979. Although Miller argues strongly for the ineffectiveness of techniques similar to Adleman's for computing elliptic logarithms [Mil85a], it may be possible to devise an alternative method. Ideally, this method would run in subexponential time.

- Is there an algorithm that computes elliptic logarithms in subexponential time?

References

[ACGS] Werner Alexi, Benny Chor, Oded Goldreich, and Claus P. Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. To appear, SIAM Journal of Computing.

[Adl79] Leonard Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, pages 55–60, IEEE Computer Society, 1979.

[AW85] Miklos Ajtai and Avi Wigderson. Deterministic simulation of probabilistic constant depth circuits. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, 1985.

[Bac83] Eric Bach. How to generate factored random numbers. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 184–188. Association for Computing Machinery, 1983.

[Bac85] Eric Bach. Lenstra's algorithm for factoring with elliptic curves, an exposé. 1985. Unpublished manuscript.

[BBS83] Lenore Blum, Manuel Blum, and Michael Shub. Comparison of two pseudo-random number generators. In *Proceedings of Crypto'82*, pages 61–78. Plenum Press, 1983.

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal of Computing*, 13(4):850–864, 1984.

[BMP75] I. Borosh, C.J. Moreno, and H. Porta. Elliptic curves over finite fields: II. *Mathematics of Computation*, 29(131):951–964, July 1975.

[Cas66] J.W.S. Cassels. Diophantine equations with special reference to elliptic curves. *Journal of the London Mathematical Society*, 41:193–291, 1966.

[Cho65] S. Chowla. *The Riemann Hypothesis and Hilbert's Tenth Problem*. Gordon and Breach, 1965.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions in Information Theory*, IT-22(6):644–654, November 1976.

[Dra67] Alvin W. Drake. *Fundamentals of Applied Probability Theory*. McGraw-Hill, Inc., 1967.

[Ful69] William Fulton. *Algebraic Curves*. W.A. Benjamin, 1969.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, 1984.

[GK86] Shafi Goldwasser and Joseph Kilian. A provably correct, probably fast primality testing algorithm. 1986. To appear, Proceedings of the 18th Annual ACM Symposium on Theory of Computing.

[Jol73] Jean-René Joly. Equations et varietés algébriques sur un corps fini. *L'Enseignment Mathématique*, 59:74–79, 1973.

[Kob84] Neal Koblitz. *Introduction to Elliptic Curves and Modular Forms*. Volume 97 of *Graduate Texts in Mathematics*, Springer-Verlag, 1984.

[KT76] Donald E. Knuth and Luis Trabb-Pardo. Analysis of a simple factorization algorithm. *Theoretical Computer Science*, 3:321–348, 1976.

[Len85] H.W. Lenstra. Elliptic curve factorization. Memorandum, 1985.

[Lev85] Leonid Levin. One-way functions and pseudorandom generators. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 363–365, Association for Computing Machinery, 1985.

[LW83] Douglas L. Long and Avi Wigderson. How discreet is the discrete log? In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 413–420, Association for Computing Machinery, 1983.

[Mil85a] Victor Miller. Elliptic curves and cryptography. 1985. To appear, Proceedings of Crypto'85.

[Mil85b] Victor Miller. Short programs for functions on curves. 1985. Unpublished manuscript.

[Pom85] Carl Pomerance. How to factor a number. Seminar, MIT Laboratory for Computer Science, 1985.

[RS62] J. Rosser and L. Schoenfield. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6:64–94, 1962.

[Sch85] René Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of Computation*, 44:483–494, April 1985.

[SS77] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal of Computing*, 6:84–85, 1977.

[Tat74] John T. Tate. The arithmetic of elliptic curves. *Inventiones Mathematicae*, 23:179–206, 1974.

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, IEEE Computer Society, 1982.

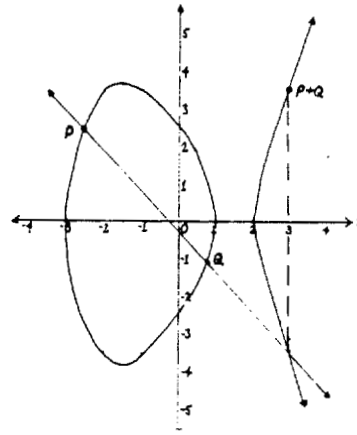

Figure 1: An elliptic curve, showing tangents and chords operation.

Assume M is given. Algorithm computes small prime factors of an integer N.

1. Initialize $S \leftarrow \{\}$.
2. Run Lenstra's algorithm for time M to find a factor p. If no factor is found in this time, go to 4.
3. Set $S \leftarrow S \cup \{p\}$, $N \leftarrow N/p$; go to 2.
4. Return S.

Figure 2: Algorithm *small-factors*.

Suppose $N = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$, p_i prime, $p_i < p_{i+1}$, $\alpha_i > 0$. Algorithm finds order of x in Abelian group G using complete factorization of N.

1. For each i, do step 2.
2. Let β_i be the smallest integer such that $(Np_i^{\beta_i}/p_i^{\alpha_i})x = 0$.
3. Output $p_1^{\beta_1} \cdots p_k^{\beta_k}$.

Figure 3: Algorithm *with-complete-factorization*.

Suppose $N = p_1^{\alpha_1} \cdots p_k^{\alpha_k} m$, p_i prime, $p_i < p_{i+1}$, $\alpha_i > 0$, m not necessarily prime. Algorithm finds order of x in Abelian group G using partial factorization of N.

1. For each i, do step 2.
2. Let β_i be the smallest integer such that $(Np_i^{\beta_i}/p_i^{\alpha_i})x = 0$.
3. If $(N/m)x = 0$, then output $p_1^{\beta_1} \cdots p_k^{\beta_k}$; else output $p_1^{\beta_1} \cdots p_k^{\beta_k} m$.

Figure 4: Algorithm *with-partial-factorization*.

Suppose N is the order of G. Algorithm determines whether x generates Abelian group G.

1. Use algorithm *small-factors* to compute a partial factorization of N.
2. Apply algorithm *with-partial-factorization* to compute an approximation o to $\text{order}_G(x)$.
3. If $o = N$, output "yes"; else output "no."

Figure 5: Algorithm *cyclic-generating-test*.

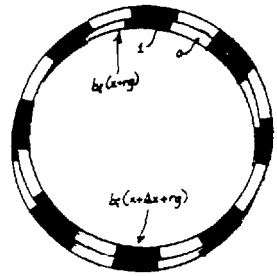

Figure 6: Correlation circle.

Estimates $\rho_{i,\epsilon}(cg, x)$ for an integer c.

1. Choose r_j at random, $0 \leq r_j < N$, $0 \leq j < M$.
2. For each r_j, compare $b_i(cg + r_j g)$ to $b_{i,\epsilon}(x + r_j g)$. Set *correct* to the number of equal results.
3. Return $correct/M$.

Figure 7: Algorithm *estimate-correlation.*

Given x, and assuming that $\text{index}(2^{i+1}x) \in [2^{i+1}a, 2^{i+1}b]$, determines in which interval $\text{index}(2^i x)$ lies.

1. Let c the integer closest to $(a+b)/2$, and compute $\xi_{i,\epsilon}(cg, x)$ using algorithm *estimate-correlation.*
2. If the estimate is positive, then return $[2^i a, 2^i b]$; else return $[2^i a + N/2, 2^i b + N/2]$.

Figure 8: Algorithm *decide-square-roots.*

1. Choose an interval $[a, b]$ with
$$0 \leq a \leq b \leq a + \frac{N\epsilon}{2^{i+1}} - 3 < \frac{N}{2^{i+1}}.$$
2. For j from $n-1$ to 0, set $y \leftarrow 2^j x$ and use algorithm *decide-square-roots* to determine whether $\text{index}(2^i y)$ is in $[2^i a, 2^i b]$ or $[2^i a + N/2, 2^i b + N/2]$, given $\text{index}(2^{i+1}y) \in [2^{i+1}a, 2^{i+1}b]$. In the former case, set $[a, b] \leftarrow [a/2, b/2]$; in the latter, set $[a, b] \leftarrow [a/2 + N/2^{i+1}, b/2 + N/2^{i+1}]$.
3. Let c be the integer in $[2^{i+1}a, 2^{i+1}b]$. If there is no integer, or if $cg \neq 2^i x$, then go to 2.
4. Given that $\text{index}(2^i x) = c$, test up to 2^i integer values of the form
$$\frac{c}{2^i} + k\frac{2^i}{N},$$
$0 \leq k < 2^i$, to find $\text{index}(x)$.

Figure 9: Algorithm *compute-index.*

Algorithm selects $x \in I_n$ with uniform probability in polynomial expected time.

1. Apply Bach's algorithm to produce $p+1$ in factored form. If $p+1 = 2^{n-1}$, then let $p = 2^n - 1$.
2. Test that p is prime, and $p \equiv 2 \bmod 3$. If not, go to 1.
3. Guess an n-bit number B. If $B = 0$ or $B \geq p$, go to 1. The elliptic curve is $E : y^2 = x^3 + B$.
4. Guess an n-bit number a. If $a > p$, go to 1. Else let $G = \phi^{-1}(a)$. Check that G generates $E(\mathbf{F}_p)$ using algorithm *cyclic-generating-test* (section 5.3). If not, go to 1.
5. Guess an n-bit number a. If $a > p$, go to 1. Else let $P = \phi^{-1}(a)$
6. Output $\langle E(\mathbf{F}_p), G, P \rangle$.

Figure 10: Algorithm *simple-accessibility.*

Computes $\text{index}_{E(\mathbf{F}_p),G}(P)$ for some $\langle E(\mathbf{F}_p), G\rangle$.

1. Run algorithm *compute-index* using algorithm A with fixed $\langle E(\mathbf{F}_p), G\rangle$ as the approximator, to produce c.
2. Output c.

Figure 11: Algorithm *simple-index*.