

A Fast Java Implementation of a Provably Secure Pseudo Random Bit Generator Based on the Elliptic Curve Discrete Logarithm Problem

Harald Baier

Darmstadt Center of IT-Security,
Alexanderstr. 10, D-64283 Darmstadt, Germany
hbaier@dzi.tu-darmstadt.de

Abstract. We present a pseudo random bit generator whose security is based on the intractability of the discrete logarithm problem in the group $E(\mathbb{F}_p)$ of rational points on an elliptic curve over a finite prime field \mathbb{F}_p . The bit generator is implemented within the framework of the Java Cryptography Architecture (JCA). It uses an elliptic curve E chosen such that both $E(\mathbb{F}_p)$ and its twist $E^{tw}(\mathbb{F}_p)$ are of prime order and cryptographically strong. We show how to efficiently find such curves. As attacking the bit generator is at least as difficult as the elliptic curve discrete logarithm problem we propose to use it for generating key pairs for elliptic curve cryptography.

Keywords: complex multiplication, cryptography, elliptic curve, finite field, Java Cryptography Architecture, pseudo random bit generator

1 Introduction

As of today the ability of producing random integers is crucial for the security of most cryptographic applications. For instance, in public key cryptography the security depends on hiding a secret key. This key has to be accessible only to its authorized owner. It is computed using a random number generator. Once, this generator is corrupted, an adversary is able to reveal the secret key and hence break the system.

However, sources of truly random integers are hard to use in practice. It is therefore common to search for pseudorandom number generators. Roughly speaking, a pseudorandom source may not be distinguished from a truly random source by any polynomial time algorithm. Various instances of pseudorandom number generators were proposed in the past ([MOV97], [BG99]). In this paper we develop a variant of a pseudorandom number generator proposed by B. Kaliski ([Kal86], [Kal88]). Kaliski's generator makes use of elliptic curve groups over finite fields. We therefore refer to our pseudorandom number generator as ECPRNG.

Let E be an elliptic curve defined over some finite prime field \mathbb{F}_p . We denote a twist of E over \mathbb{F}_p by E^{tw} . In [Kal88] it is shown that breaking the ECPRNG is at least as difficult as solving the elliptic curve discrete logarithm problem (ECDLP) in both groups $E(\mathbb{F}_p)$ and $E^{tw}(\mathbb{F}_p)$. The ECDLP is known to be fully exponential if the groups are chosen with care. The security of the ECPRNG may therefore be reduced to a well known difficult problem. Thus the ECPRNG is said to be *provably secure*. It is this property which makes the ECPRNG attractive for use in highly sensitive applications.

The pseudorandom number generator as proposed by Kaliski leaves one key problem open: Which general elliptic curve groups should be used in order to have an efficient ECPRNG and how do we find them? Additionally, no efficient implementation of an ECPRNG is

publicly available. Our contribution is therefore twofold: We first show how to optimize the ECPRNG. Second, we yield an efficient open source implementation of the number generator.

In order to solve the first problem we make use of elliptic curve groups $E(\mathbb{F}_p)$ and $E^{tw}(\mathbb{F}_p)$ of prime order, respectively. Elliptic curve groups of prime order are very attractive for use in practice, as they allow to use a field of minimal bitlength for a given security level. Finding such group pairs was believed to be very difficult. Based on work of Spallek ([Spa92]), Atkin, Morain ([AM93]), Lay ([LZ94]), and Baier ([BB00], [Bai01]) we present an algorithm to efficiently find such curve pairs. The run time on an ordinary PC for p of bitlength 160 turns out to be about 14 seconds without any precomputation.

Our open source implementation of the ECPRNG is based on the framework of the Java Cryptography Architecture (JCA). Thus, once the ECPRNG is compiled successfully, it may be used on any platform and like any other class of the JCA. This makes our ECPRNG easily accessible to developers of cryptographic applications. Our ECPRNG is part of the FlexiProvider and freely available at [Fle02]. We use efficient algorithms for the elliptic curve group law.

We propose to make use of our ECPRNG to generate public/private key pairs for elliptic curve cryptography. Then the security of the whole system only relies on the intractability of the elliptic curve discrete logarithm problem. This independency of any other problem makes our ECPRNG superior to different pseudorandom number generators for use in elliptic curve cryptography. We show that the bit rate of our ECPRNG is sufficiently large for use in practice. For example, the ECPRNG generates about 9 private keys per second on an ordinary PC using freely available software. Thus a Certification Authority may find $7.62 \cdot 10^3$ private keys per day.

Some different pseudorandom number generators using elliptic curves were proposed recently. We refer for example to [Hal94], [GBS00], [GL01], and [BD01]. Their security is based on a different problem than the elliptic curve discrete logarithm problem. Hence, it may not be compared directly to our ECPRNG. However, in [LW01] pseudorandom bits are generated using a walk through a subgroup of some elliptic curve group $E(\mathbb{F}_p)$. The authors claim that their generator passes standard statistical tests as described in FIPS 140-2. Nevertheless, the relation of the security of the generator in [LW01] to the discrete logarithm problem in the underlying subgroup of $E(\mathbb{F}_p)$ is not discussed.

The rest of the paper is organized as follows: First, in Section 2 we introduce the necessary theory of elliptic curves in our context. Next, in Section 3 we present the ECPRNG. In Section 4 we describe our method to find elliptic curve groups of prime order. Finally, in Section 5 we describe our implementation and discuss its performance in practice.

2 Elliptic curves

We review some basic facts concerning elliptic curves over finite fields. In addition, we introduce a twist of an elliptic curve. Finally, we list the requirements for an elliptic curve group to be cryptographically strong.

We first explain the term of an elliptic curve in our context. Let p be a prime number, $p > 3$. An *elliptic curve* over the prime field \mathbb{F}_p of characteristic p is a pair $E = (a, b) \in \mathbb{F}_p^2$ with $4a^3 + 27b^2 \neq 0$. We set $f_E = t^3 + at + b$. A *point* on E is a solution $(x, y) \in \mathbb{F}_p^2$ of $y^2 = f_E(x)$ or the point at infinity O obtained by considering the projective closure of this

equation. The set of points on E over \mathbb{F}_p is denoted by $E(\mathbb{F}_p)$. It carries a group structure with the point at infinity acting as the identity element. This group is called the *group of rational points* of E over \mathbb{F}_p . The group operation is written as an addition.

Let $P \in E(\mathbb{F}_p)$, $P = (x, y)$. If $n \in \mathbb{N}_0$, we write $[n]P$ for the point $P + \dots + P$, where P is added $n - 1$ times to itself. In addition, we define $-P = (x, -y)$. If $n \in \mathbb{Z}_{<0}$, we set as usual $[n]P = [-n](-P)$. Now let $R = [n]P$, where n is a non-negative integer smaller than the order of P in $E(\mathbb{F}_p)$. The *elliptic curve discrete logarithm problem* is to recover n from P and R .

We next introduce a twist of an elliptic curve over \mathbb{F}_p . Let γ be a quadratic non-residue in \mathbb{F}_p , that is the equation $t^2 = \gamma$ has no solution $t \in \mathbb{F}_p$. If we set $a^{tw} = a\gamma^2$, $b^{tw} = b\gamma^3$, and $E^{tw} = (a^{tw}, b^{tw})$, then E^{tw} is called a *twist* of E over \mathbb{F}_p . We remark that for every non-square γ the corresponding elliptic curve E^{tw} is said to be a twist of E over \mathbb{F}_p . In addition, we set $f_{E^{tw}} = t^3 + a^{tw}t + b^{tw}$.

The pseudorandom number generator proposed by Kaliski ([Kal86], [Kal88]) or its extension by Lippert ([Lip00]) uses either a supersingular elliptic curve or the whole groups $E(\mathbb{F}_p)$ and $E^{tw}(\mathbb{F}_p)$ of twisted elliptic curves, respectively. As the first family of curves has turned out to be cryptographically insecure ([MOV91]), we only consider the second proposal. However, we have to know a set of generating points of the respective groups. The group structure is well known (see [Rüc87]):

Theorem 1. *Let p be a prime and $E = (a, b)$ be an elliptic curve over \mathbb{F}_p . Then*

$$E(\mathbb{F}_p) \simeq \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}, \quad n_2 \mid \gcd(n_1, p - 1). \quad (2.1)$$

The computation of the isomorphism type $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$ of an elliptic curve group $E(\mathbb{F}_p)$ and finding generators of the cyclic subgroups is a non-trivial task. In addition, in practice it turns out to be time consuming. We therefore propose to use an elliptic curve E such that both $E(\mathbb{F}_p)$ and $E^{tw}(\mathbb{F}_p)$ are of prime order, respectively. This was believed to be very difficult. However, in Section 4 we describe an algorithm for this task. The algorithm turns out to be very efficient in practice.

Once we have found non-trivial points $G \in E(\mathbb{F}_p)$ and $G^{tw} \in E^{tw}(\mathbb{F}_p)$, respectively, we have found generators of both groups, too. A non-trivial point in a group $E(\mathbb{F}_p)$ is simply computed by choosing random values $x \in \mathbb{F}_p$ and testing, if $f_E(x)$ is a square in \mathbb{F}_p . In practice, this proceeding is very fast. The same holds for $E^{tw}(\mathbb{F}_p)$.

We point to a further advantage of using elliptic curve groups of prime order. Due to a famous theorem of Hasse we know $|E(\mathbb{F}_p) - p - 1| \leq 2\sqrt{p}$. Thus $|E(\mathbb{F}_p)|$ and p are of same order of magnitude. The difficulty of the elliptic curve discrete logarithm problem in $E(\mathbb{F}_p)$ is known to be essentially as hard as the difficulty of computing discrete logarithms in the largest subgroup of prime order in $E(\mathbb{F}_p)$. Let r denote this prime. The security of an elliptic curve cryptosystem is measured in terms of the bitlength of r . Hence, if we choose $|E(\mathbb{F}_p)| = r$, the field \mathbb{F}_p may be chosen to be of minimal bitlength for some given security level. As the arithmetic in $E(\mathbb{F}_p)$ is implemented as arithmetic in \mathbb{F}_p , our choice of curves allow the most efficient field arithmetic, if a security level is chosen.

We finally explain the requirements we have to impose on an elliptic curve group $E(\mathbb{F}_p)$ to ensure the discrete logarithm problem in $E(\mathbb{F}_p)$ to be intractable. We refer to such a group as *cryptographically strong*. The conditions may be found in international standards like [P1363], [SEC1], or [X9.62]. As explained above we assume $|E(\mathbb{F}_p)| = r$ with a prime

r . The first condition on r is to be of bitlength ≥ 160 . This avoids the application of general discrete logarithm algorithms such as Pollard's ρ -algorithm ([Pol78]). Second, by $r \neq p$ we ensure that the anomalous curve attack as proposed by Smart et.al is not feasible (e.g. [Sma99], [Sem98]). Finally, r is not allowed to divide $p^d - 1$ for small values d , say $d \leq 20$. In fact, our curves satisfy the last condition with $d \leq 10000$. It makes the attacks of Menezes, Okamoto, Vanstone ([MOV91]) and Frey, Rück ([FR94]) impossible .

We point out that the crucial condition is the primality of the group orders of $E(\mathbb{F}_p)$ and $E^{tw}(\mathbb{F}_p)$, respectively. We show in Section 4 how to efficiently solve this problem for a large family of elliptic curves.

3 The Pseudo Random Number Generator

In this section we explain our ECPRNG. It is very similar to a pseudorandom number generator which was proposed by Kaliski ([Kal86], [Kal88]) and later investigated by Lippert ([Lip00]).

Let a prime p be given. The fundamental idea of Kaliski's pseudorandom number generator is the use of a one-way permutation on the set $\{0, \dots, 2p + 1\}$. We denote the permutation by ϕ_p and the set by S_p . Informally speaking this means that we may efficiently evaluate the value of $\phi_p(n)$ for any $n \in S_p$. On the other hand, if $m \in S_p$ the inverse operation $\phi_p^{-1}(m)$ may *not* be computed by some polynomial time algorithm. We say that ϕ_p has the *one-way property*. In order to explain the one-way permutation ϕ_p , we first have to show that for each $s \in \mathbb{F}_p$ there are two points in either $E(\mathbb{F}_p)$ or $E^{tw}(\mathbb{F}_p)$ with x -coordinate equal to s or γs , respectively.

Let $s \in \mathbb{F}_p$ in what follows. First, let $f_E(s) = 0$. Then it is easy to see that $(s, 0) \in E(\mathbb{F}_p)$ and $(\gamma s, 0) \in E^{tw}(\mathbb{F}_p)$. Next, let $f_E(s)$ be a quadratic residue in \mathbb{F}_p^\times . Then $(s, \pm\sqrt{f_E(s)})$ are two different points in $E(\mathbb{F}_p)$. However, we have $f_{E^{tw}}(\gamma s) = \gamma^3 f_E(s) = \gamma \cdot (\gamma\sqrt{f_E(s)})^2$. As γ is a quadratic non-residue in \mathbb{F}_p^\times , the element $f_{E^{tw}}(\gamma s)$ is a quadratic non-residue, too. Thus there is no element in $E^{tw}(\mathbb{F}_p)$ with x -coordinate equal to γs . Finally, let $f_E(s)$ be a quadratic non-residue modulo p . Then there is no point in $E(\mathbb{F}_p)$ with x -coordinate s . Similar as above it is easy to see that $f_{E^{tw}}(\gamma s)$ is a square in \mathbb{F}_p^\times . Thus $(\gamma s, \pm\sqrt{\gamma^3 f_E(s)})$ are two different points in $E^{tw}(\mathbb{F}_p)$. Hence in all taking the points at infinity into account we have $|E(\mathbb{F}_p)| + |E^{tw}(\mathbb{F}_p)| = 2p + 2$. We remark that this result does not require $E(\mathbb{F}_p)$ or $E^{tw}(\mathbb{F}_p)$ to be of prime order.

The permutation ϕ_p is a composition of two bijections. We next explain these bijective maps. We denote the bijections by χ and g , respectively.

We first turn to the function χ . Let $E(\mathbb{F}_p)$ and $E^{tw}(\mathbb{F}_p)$ be cryptographically strong elliptic curve groups of prime order r and r^{tw} , respectively. From the above discussion we know $|S_p| = r + r^{tw}$. The function χ is a map from $E(\mathbb{F}_p) \cup E^{tw}(\mathbb{F}_p)$ onto S_p . However, $E(\mathbb{F}_p)$ and $E^{tw}(\mathbb{F}_p)$ may not be disjoint. Thus we have to keep track of the curve to which a point $P \in E(\mathbb{F}_p) \cup E^{tw}(\mathbb{F}_p)$ belongs to. We therefore introduce a bit c , which is 0 if and only if $P \in E(\mathbb{F}_p)$, and 1 otherwise. We set $T_{p,a,b,\gamma} := (E(\mathbb{F}_p) \times \{0\}) \cup (E^{tw}(\mathbb{F}_p) \times \{1\})$. We call $T_{p,a,b,\gamma}$ a *twisted pair*. Furthermore, in order to distinguish a 'positive' and a 'negative' y -coordinate we define a sign-function

$$\text{sgn} : \mathbb{F}_p^\times \rightarrow \{0, 1\}, \quad \text{sgn}(y) = \begin{cases} 0, & \text{if } 1 \leq y \leq \frac{p-1}{2} \quad , \\ 1, & \text{if } \frac{p+1}{2} \leq y \leq p-1 \quad . \end{cases} \quad (3.1)$$

We are now able to introduce the function $\chi_{p,a,b,\gamma} : T_{p,a,b,\gamma} \rightarrow S_p$. First, we set $\chi_{p,a,b,\gamma}(O, 0) = 2p$ and $\chi_{p,a,b,\gamma}(O^{tw}, 1) = 2p + 1$, where O and O^{tw} denote the points at infinity in $E(\mathbb{F}_p)$ and $E^{tw}(\mathbb{F}_p)$, respectively. Second, if $P = (x, y)$ denotes some non-zero point, we set

$$\chi_{p,a,b,\gamma}((x, y), c) = \begin{cases} 2x + \operatorname{sgn} y, & \text{if } y \neq 0, c = 0 \text{ ,} \\ 2 \left(\frac{x}{\gamma} \bmod p \right) + \operatorname{sgn} y, & \text{if } y \neq 0, c = 1 \text{ ,} \\ 2x, & \text{if } y = 0, c = 0 \text{ ,} \\ 2 \left(\frac{x}{\gamma} \bmod p \right) + 1, & \text{if } y = 0, c = 1 \text{ .} \end{cases} \quad (3.2)$$

We abbreviate $\chi_{p,a,b,\gamma}$ by χ if the parameters p, a, b , and γ are obvious from the context. Proposition 1 yields that χ actually is a bijection.

Proposition 1. *Let $T_{p,a,b,\gamma}$ be a twisted pair. Then $\chi_{p,a,b,\gamma}$ is bijective.*

The proof is straightforward and may be found in [Kal88].

In addition, we next define the second bijection $g_p : S_p \rightarrow T_{p,a,b,\gamma}$ by setting

$$g_p(s) = \begin{cases} ([s]G, 0), & \text{if } 0 \leq s \leq r - 1 \text{ ,} \\ ([s - r]G^{tw}, 1) & \text{if } r \leq s \leq 2p + 1 \text{ .} \end{cases} \quad (3.3)$$

It is obvious that g_p actually is bijective. In addition, as long as we consider the elliptic curve discrete logarithm problem to be intractable in both $E(\mathbb{F}_p)$ and $E^{tw}(\mathbb{F}_p)$ the function g_p has the one-way property.

We set $\phi_p := \chi_{p,a,b,\gamma} \circ g_p$. The function ϕ_p is a permutation of S_p . Obviously, it may be evaluated efficiently. However, its inverse $\phi_p^{-1} = g_p^{-1} \circ \chi_{p,a,b,\gamma}^{-1}$ is not computable in polynomial time as g_p^{-1} has this property.

We now introduce a further function needed in the framework of the ECPRNG. We denote this function by B . It is a map from S_p onto $\{0, \dots, 2^8 - 1\}$. B bases on a further function denoted by b_1 which requires an integer m and a positive integer n as argument. It returns 0 if $m \bmod n < n/2$, and 1 otherwise. As usual $m \bmod n$ denotes the smallest non-negative integer which is congruent m modulo n . For $i > 1$ we recursively set $b_i(m, n) := b_{i-1}(2m, n)$. It is easy to see that we have $b_i(m, n) = b_1(2^{i-1}m, n)$. Using the functions b_i up to $i = 8$ we define B as

$$B(s) = \begin{cases} \sum_{k=0}^7 b_{k+1}(s, r)2^k, & \text{if } 0 \leq s \leq r - 1 \text{ ,} \\ \sum_{k=0}^7 b_{k+1}(s - r, r^{tw})2^k, & \text{if } r \leq s \leq 2p + 1 \text{ .} \end{cases} \quad (3.4)$$

The output of $B(s)$ may be interpreted as a string of 8 pseudorandom bits. In our implementation, these bits are stored in a byte. The choice $i = 8$ is a good tradeoff between security (output as few bits as possible per round) and performance (return as much bits as possible per round) (see also [Kal88], [Lip00] for a detailed discussion).

We are now able to explain the ECPRNG. A survey of it is given below. Input of the ECPRNG are positive integers N and l . The algorithm outputs N pseudorandom bytes. The value of l determines the bitlength of p .

The algorithm first computes a twisted pair $T_{p,a,b,\gamma}$. It uses algorithm `twistedPair`(l) as explained in Section 4. The result of `twistedPair`(l) is a prime p of bitlength l and twisted elliptic curve groups $E(\mathbb{F}_p)$ and $E^{tw}(\mathbb{F}_p)$ of prime order r and r^{tw} , respectively.

In addition, both groups are cryptographically strong. As we have $r \approx p \approx r^{tw}$ due to the Theorem of Hasse, l determines the security level of the generator. In order to respect the requirements of Section 2, we assume $l \geq 160$.

Once the elliptic curves are known, the ECPRNG computes generators G and G^{tw} of the respective group by trial and error as explained in Section 2.

The current state of the generator is represented by an integer $s \in S_p$. The initial state of the generator is called the *seed*. In order to get a seed, the ECPRNG next invokes our function `getSeed(p)`. We explain `getSeed(p)` in Section 5.

For all i , $1 \leq i \leq N$, the ECPRNG now proceeds as follows: It first computes the pseudorandom byte $B(s)$ corresponding to the current state s as explained in Equation (3.4). Next it computes the subsequent state by evaluating $\phi_p(s)$. Thus the ECPRNG invokes $g_p(s)$ as introduced in Equation (3.3). Let (P, c) be the result. Then it invokes $\chi_{p,a,b,\gamma}(P, c)$ as defined in Equation (3.2) to get the next state of the generator.

Algorithm 3.1. ECPRNG(N, l)

Input: A positive integer N .

An integer $l \geq 160$.

Output: A sequence $v(N) \dots v(1)$ of N pseudorandom bytes.

```

 $T(p, a, b, \gamma) \leftarrow \text{twistedPair}(l);$ 
Compute generators  $G$  of  $E(\mathbb{F}_p)$  and  $G^{tw}$  of  $E^{tw}(\mathbb{F}_p)$  by trial and error;
 $s \leftarrow \text{getSeed}(p);$  // Set initial state to  $s$ 
for  $i = 1$  to  $N$  do
     $v(i) \leftarrow B(s);$  // Compute a byte using Equation (3.4)
     $(P, c) \leftarrow g_p(s);$  // Compute a new point using Equation (3.3)
     $s \leftarrow \chi_{p,a,b,\gamma}(P, c);$  // Update the state using Equation (3.2)
end for
return(  $v(N) \dots v(1)$  );

```

We remark that once the seed is known the ECPRNG proceeds deterministically. Thus the seed has to be chosen with care. We present our approach in Section 5. We point out that the ECPRNG reveals nothing about the elliptic curves in use. It only returns the pseudorandom bytes.

We close this section with a security remark. In [Kal88] and [Lip00], the authors show that if an adversary is able to guess bits produced by the ECPRNG with a probability $1/2 + \epsilon$ where $\epsilon > 0$, then the attacker is able to compute discrete logarithms in $E(\mathbb{F}_p)$ and $E^{tw}(\mathbb{F}_p)$, too. Conversely, if we believe the elliptic curve discrete logarithm problem to be intractable, the ECPRNG may be considered to be secure.

4 Finding Suitable Elliptic Curves

In this section we deal with the problem of how to efficiently find a twisted pair $T_{p,a,b,\gamma}$. To remember a twisted pair is essentially a pair of twisted elliptic curves $E = (a, b)$ and $E^{tw} = (a\gamma^2, b\gamma^3)$ both defined over \mathbb{F}_p and such that the respective group of rational points over \mathbb{F}_p is of prime order and cryptographically strong. No such algorithm was previously known for this task.

For lack of space we only describe `twistedPair`(l) on an abstract level. A more comprehensive explanation of it may be found in [Bai02]. The algorithm makes use of the theory of complex multiplication. A detailed discussion of this theory is out of the scope of this paper. We only sketch the main points. Our algorithm is based on work of Spallek ([Spa92]), Atkin, Morain ([AM93]), Lay ([LZ94]), and Baier ([BB00], [Bai01]). We extend the latter work to find a twisted pair.

Input of `twistedPair`(l) is an integer $l \geq 160$. Its output is a twisted pair $T_{p,a,b,\gamma}$ with $\lceil \log_2 p \rceil = l$. We give evidence that `twistedPair`(l) is very fast in practice for a large family of elliptic curves. We explain at the end of this section what we mean by the term 'large family'. For instance, if $l = 160$ the run time is about 14 seconds on an ordinary PC, as we will see in Section 5. We are not aware of any comparable algorithm.

Let l be given. The first step of `twistedPair` is to solve a *norm equation*. A norm equation is an equation of the form

$$t^2 - \Delta y^2 = 4p \ , \tag{4.1}$$

where t and y are integers, p is a prime, and Δ is an imaginary quadratic discriminant, that is a negative integer with $\Delta \equiv 0, 1 \pmod{4}$.

Once the norm equation is solved, using complex multiplication elliptic curves E and E^{tw} over \mathbb{F}_p are constructed such that

$$|E(\mathbb{F}_p)| = p + 1 - t, \quad |E^{tw}(\mathbb{F}_p)| = p + 1 + t . \tag{4.2}$$

In the context of our ECPRNG we have to solve Equation (4.1) such that $\lceil \log_2 p \rceil = l$ and such that both numbers $p + 1 - t$ and $p + 1 + t$ are prime and orders of cryptographically strong elliptic curve groups, respectively. This imposes the condition $\Delta \equiv 5 \pmod{24}$ on the imaginary quadratic discriminant, as explained in [Bai02].

There are essentially two different approaches to solve the norm equation. The first one fixes the prime p and searches for an imaginary quadratic discriminant Δ , such that Equation (4.1) has a solution with the above mentioned boundary conditions. This turns out to be rather slow (see [Bai02]). The second approach fixes the discriminant Δ and searches for values t and y such that the right side of Equation (4.1) is four times a prime and such that the boundary conditions hold. This is by far more efficient than the first approach. A detailed investigation how to efficiently solve the norm equation may be found in [Bai02]. We do not know a further efficient algorithm for this task.

Now let the norm equation successfully be solved for an imaginary quadratic discriminant Δ . The next step of `twistedPair` is to compute a polynomial in $\mathbb{Z}[X]$ corresponding to Δ . This polynomial is called the *class polynomial*. It is due to Atkin and Morain ([AM93]) and denoted by G_Δ . Its degree is called the *class number* of Δ . We write $h(\Delta)$ for the class number. It is well known that $G_\Delta \pmod{p}$ splits into pairwise different linear factors. If $c_p \in \mathbb{F}_p$ denotes a root of $G_\Delta \pmod{p}$, the elliptic curves E and E^{tw} are easily recovered from c_p . Again we refer to [Bai02] for details how to efficiently compute the polynomial G_Δ .

If the class number is large, say ≥ 200 , the coefficients of G_Δ become very large, too. For example, let us consider the case $\Delta = -356131$. It is (with respect to its absolute value) the smallest discriminant with $h(\Delta) = 200$ and $\Delta \equiv 5 \pmod{24}$. The largest coefficient of G_Δ (again with respect to its absolute value) is of bitlength 3549. The large coefficients of G_Δ make its computation very burdensome. For this reason it was believed that only polynomials of degree at most 50 were computable in reasonable time ([MP97]). However,

we show in Section 5 that our implementation is very fast on an ordinary PC even if $h(\Delta) \geq 200$. We point out that the implementation is freely available at [LiDIA].

Again we point to the remarkable feature of algorithm `twistedPair` that it is *not* restricted to imaginary quadratic discriminants of small class numbers, say class numbers at most 50. This may turn our elliptic curves out to be more secure, as some people have expressed their concern at using discriminants of small class numbers. However, no attack is known for elliptic curves being constructed using a discriminant of small class number. Nevertheless, the argument of not using such curves has weighty supporters in the cryptographic community (see e.g.[GIS01]).

When saying that our algorithm `twistedPair` is applicable to a large family of elliptic curves, we mean that our algorithm is *not* restricted to imaginary quadratic discriminants of a small class number. A sample output of `twistedPair(160)` is given in the Appendix.

5 Implementation and Performance Issues

We finally describe our implementation. In addition, we give some performance details. All our practical tests are performed on an Athlon XP1600+ running Linux 2.4.10 at 1.4 GHz and having 1 GByte main memory. We remark that all software in use is freely available.

From Section 3 we know that the ECPRNG proceeds in mainly three steps. Let N and l be given as described in Section 3. The algorithm first computes a twisted pair $T_{p,a,b,\gamma}$ as explained in Section 4. In addition, it determines generating points of both elliptic curve groups. Second, the ECPRNG sets its initial state, that is its seed. Finally, the ECPRNG computes the N pseudorandom bytes. We discuss these steps in what follows.

First, we turn to the computation of a twisted pair. All our programs for this task are implemented in C++ using the GNU compiler `gcc 3.0.1`, the GNU multiprecision package `gmp 3.2.1`, and a prerelease of the computer algebra library `LiDIA 2.1` ([LiDIA]). The actual parameter generation is done by our library `gec` (generate elliptic curves, [LiDIA]). As explained in Section 4 the run time is heavily dependent on the imaginary quadratic discriminant in use. In Table 5.1 we list run times for input values $160 \leq l \leq 200$, $10 \mid l$. We choose $\Delta = -356131$ as we have $h(\Delta) = 200$, $\Delta \equiv 5 \pmod{24}$, and $|\Delta|$ is minimal with these properties. The run times are the average times of 10 tests. We conclude from Table 5.1 that our algorithm `twistedPair(l)` is very fast, even for values $l = 200$.

l	160	170	180	190	200
CPU time in seconds	13.6	15.5	18.2	20.2	24.2

Table 5.1. CPU time of algorithm `twistedPair(l)` for $160 \leq l \leq 200$, $10 \mid l$. We make use of $\Delta = -356131$ with $h(\Delta) = 200$. In addition, generating points of both elliptic curve groups are computed.

In addition, we show that our generating algorithm is fast even for discriminants of class numbers > 200 . In Table 5.2 we fix the bitlength $l = 160$. We test our algorithm `twistedPair` for discriminants Δ of class numbers $250 \leq h(\Delta) \leq 500$, $50 \mid h(\Delta)$. For each given class number h the corresponding discriminant Δ is chosen such that $h(\Delta) = h$, $\Delta \equiv 5 \pmod{24}$, and such that $|\Delta|$ is minimal with these properties. As above we per-

formed 10 tests for each discriminant. Again we see that our algorithm is very efficient even for a discriminant of class number ≥ 250 .

Δ	-467011	-881011	-1190251	-1531339	-1825291	-2299699
$h(\Delta)$	250	300	350	400	450	500
CPU time in seconds	18.1	32.5	44.1	59.3	74.0	98.8

Table 5.2. CPU time of algorithm `twistedPair(160)` for discriminants Δ with $250 \leq h(\Delta) \leq 500$, $50 \mid h(\Delta)$. In addition, generating points of both elliptic curve groups are computed.

Second, we explain our algorithm `getSeed(p)`. It requires a positive integer p as input and returns an integer s , $0 \leq s \leq 2p + 1$. It is implemented in Java. We use the JDK 1.2.2 and the corresponding Sun compiler. `getSeed` proceeds as proposed in [Lip00]. As usual we write $l + 1$ for the bitlength of p . We write $s = \sum_{j=0}^l s_j 2^j$. For each j , $0 \leq j \leq l$, the following procedure yields the j -th bit of s : Let T_A be the current thread. T_A starts a new thread T_B and then sleeps for 50 milliseconds. This is done using the `sleep` member function of the Java class `Thread`. Let c_{T_B} be a counting variable in thread T_B . We simply initialize $c_{T_B} \leftarrow 0$ and then perform $c_{T_B} \leftarrow c_{T_B} + 1$ until thread T_A wakes up. If the final value of c_{T_B} is odd, we set $s_j \leftarrow 1$. Otherwise, s_j is set to 0. Finally, if the resulting s is larger than $2p + 1$, we set $s \leftarrow s \bmod 2p + 2$. Sample CPU timings of `getSeed(p)` for primes of different bitlengths are listed in Table 5.3.

l	160	170	180	190	200
CPU time in seconds	10.6	11.0	11.5	12.0	13.0

Table 5.3. CPU time of `getSeed(p)` for primes p of bitlength l .

Finally, we turn to the computation of the pseudorandom bytes. This step is implemented in Java, too. In what follows we denote by C one of the curves E or E^{tw} returned by algorithm `twistedPair`, and by P one of the corresponding points G or G^{tw} . It is obvious that the most time consuming operation is the computation of $[n]P$, where n is some non-negative integer of bitlength l . This operation is done by the algorithm `multiply(n, C, P)`. This algorithm is implemented by Henhapl, who investigates different representations of elliptic curves and different multiplication methods to compute multiples $[n]P$ in elliptic curve groups $C(\mathbb{F}_p)$ (see [Hen02]). The investigated multiplication methods are available within the FlexiProvider ([Fle02]), which is a provider for the Java Cryptography Architecture. Algorithm `multiply` uses weighted projective coordinates as proposed in [CC87] or [CMO98] and a multiplication method due to Lim and Lee ([LL94]).

Table 5.4 lists CPU times to perform one for-loop in `ECPRNG(N, l)` for $N = 1250000$ and various values of l . N is chosen such that the `ECPRNG` outputs 10 MBit of pseudorandom bits. The timings are rather promising for a Java implementation.

We next give a different representation of Table 5.4. Let $B(l)$ denote the bit rate of our `ECPRNG`, that is the number of pseudorandom bits per second. In addition, let $T(l)$ be the timing in Table 5.4. As one for-loop yields 8 pseudorandom bits, we have $B(l) = 8/T(l)$.

l	160	170	180	190	200
CPU time in milliseconds	5.67	6.13	6.82	7.18	7.76

Table 5.4. CPU time to perform one for-loop in ECPRNG(1250000, l).

Some values of $B(l)$ are listed in Table 5.5. In addition Table 5.5 presents the number of private keys for elliptic curve cryptography which may be output by our ECPRNG (if r is the security level of the elliptic curve as introduced in Section 2 a private key is simply a positive integer at most $r - 1$). We deduce that the number of generated private keys is sufficiently large for real life applications. Thus our ECPRNG may be used by a Certification Authority to generate elliptic curve cryptography key pairs.

l	160	170	180	190	200
Bit rate	1411	1305	1173	1114	1031
Number of private keys per second	8.82	7.68	6.52	5.86	5.16

Table 5.5. Bit rate and number of private keys per second of the ECPRNG.

Finally, for security reasons we propose to change the twisted pair within 24 hours. Due to the good performance of algorithm `twistedPair` this constitutes no problem for the Certification Authority. However, even if an elliptic curve group turned out to be cryptographically weak later on, an attacker would only be able to compute a limited number of private keys. These keys would have to be revoked by the Certification Authority.

References

- AM93. A.O.L. ATKIN AND F. MORAIN. Elliptic curves and primality proving. *Mathematics of Computation*, 61:29–67, 1993.
- Bai01. H. BAIER. Elliptic Curves of Prime Order over Optimal Extension Fields for Use in Cryptography. In *Progress in Cryptology - INDOCRYPT 2001*, LNCS 2247, pages 99–107, Berlin, 2001. Springer-Verlag. Second International Conference on Cryptology in India, Chennai, December 16-20.
- Bai02. H. BAIER. *Efficient Algorithms for Generating Elliptic Curves over Finite Fields Suitable for Use in Cryptography*. PhD thesis, Darmstadt University of Technology, 2002.
- BB00. H. BAIER AND J. BUCHMANN. Efficient Construction of Cryptographically Strong Elliptic Curves. In *Progress in Cryptology - INDOCRYPT2000*, LNCS 1977, pages 191–202, Berlin, 2000. Springer-Verlag. First International Conference on Cryptology in India, Calcutta, December 10-13.
- BD01. P.H.T. BEELEN AND J.M. DOUMEN. Pseudorandom sequences from elliptic curves. Technical Report, Technische Universiteit Eindhoven, 2001.
- BG99. M. BELLARE AND S. GOLDWASSER. Lecture Notes on Cryptography, 1999.
- CC87. D.V. CHUDNOVSKY AND G.V. CHUDNOVSKY. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Adv. in Appl. Math.*, 7:385–434, 1987.
- CMO98. H. COHEN, A. MIYAJI, AND T. ONO. Efficient Elliptic Curve Exponentiation using mixed coordinates. In *Proceedings of ASIACRYPT '98*, LNCS 1514, pages 51–65, Berlin, 1998. Springer-Verlag.
- Fle02. FLEXIPROVIDER. A Provider for the Java Cryptography Architecture. <http://www.flexiprovider.de>, 2002.
- FR94. G. FREY AND H.-G. RÜCK. A remark concerning m -divisibility and the discrete logarithm problem in the divisor class group of curves. *Mathematics of Computation*, 62:865–874, 1994.
- GBS00. G. GONG, T.A. BERSON, AND D.R. STINSON. Elliptic curve pseudorandom sequence generators. In *Selected areas in cryptography (Kingston, ON, 1999)*, pages 34–48, 2000.

- GIS01. Geeignete Kryptoalgorithmen, In Erfüllung der Anforderungen nach §17(1) SigG vom 16. Mai 2001 in Verbindung mit §17(2) SigV vom 22. Oktober 1997, July 2001. Bundesanzeiger Nr. 158 - Seite 18 562 vom 24. August 2001.
- GL01. G. GONG AND C.C.Y. LAM. Recursive sequences over elliptic curves. In *Proceedings of the International Conference on Sequences and their Applications*, 2001.
- Hal94. S. HALLGREN. Linear Congruential Generators Over Elliptic Curves. Technical Report CS-94-143, Cornegie Mellon University, 1994.
- Hen02. B. HENHAPL. Platform Independent Elliptic Curve Cryptography over \mathbb{F}_p . Technical Report No. TI-6/02, Darmstadt University of Technology, 2002.
- Kal86. B. KALISKI. A pseudorandom bit generator based on elliptic logarithms. In *Advances in Cryptology-CRYPTO'86*, LNCS 293, pages 84–103. Springer-Verlag, 1986.
- Kal88. B. KALISKI. *Elliptic curves and cryptography*. PhD thesis, M.I.T., 1988.
- LiDIA. LiDIA. A library for computational number theory. Darmstadt University of Technology. URL: <http://www.informatik.tu-darmstadt.de/TI/LiDIA/Welcome.html>.
- Lip00. M. LIPPERT. Ein beweisbar sicherer Pseudozufallsbit-Generator auf der Basis des DL-Problems in elliptischen Kurven. Master's thesis, Darmstadt University of Technology, 2000.
- LL94. C. LIM AND P. LEE. More Flexible Exponentiation with Precomputation. In *Advances in Cryptology - CRYPTO'94*, LNCS 839, pages 95–107, Berlin, 1994. Springer-Verlag.
- LW01. L. LEE AND K. WONG. An Elliptic Curve Random Number Generator. In *Communications and Multimedia Security Issues of the new Century*, Fifth Joint Working Conference on Communications and Multimedia Security (CMS'01), pages 127–133, 2001.
- LZ94. G.-J. LAY AND H.G. ZIMMER. Constructing elliptic curves with given group order over large finite fields. In *Proceedings of ANTS I*, LNCS 877, pages 250–263, 1994.
- MOV91. A. MENEZES, T. OKAMOTO, AND S. VANSTONE. Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field. In *Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing*, pages 80–89, 1991.
- MOV97. A. MENEZES, P.V. OORSCHOT, AND S. VANSTONE. Handbook of Applied Cryptography. CRC Press, 1997.
- MP97. V. MÜLLER AND S. PAULUS. On the Generation of Cryptographically Strong Elliptic Curves. Technical Report, Darmstadt University of Technology, 1997. Technical Report No. TI-25/97.
- P1363. P1363 Standard Specifications for Public Key Cryptography. IEEE, 2000.
- Pol78. J.M. POLLARD. Monte Carlo methods for index computation mod p . *Mathematics of Computation*, 32/143:918–924, 1978.
- Rüc87. H.-G. RÜCK. A note on elliptic curves over finite fields. *Mathematics of Computation*, 49:301–304, 1987.
- Sem98. I. SEMAEV. Evaluation of discrete logarithms in a group of p -torsion points of an elliptic curve in characteristic p . *Mathematics of Computation*, 67:353–356, 1998.
- Sma99. N.P. SMART. The Discrete Logarithm Problem on Elliptic Curves of Trace One. *Journal of Cryptology*, 12/3:193–196, 1999.
- Spa92. A.-M. SPALLEK. Konstruktion einer elliptischen Kurve über einem endlichen Körper zu gegebener Punktgruppe. Master's thesis, University of Essen, 1992.
- SEC1. SEC1 Standards for Efficient Cryptography: Elliptic Curve Cryptography. Version 1.0, 2000.
- X9.62. X9.62 Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). ANSI, 1998.

A A Sample Twisted Pair

We present a sample output of `twistedPair(160)`. We use $\Delta = -356131$ as explained in Section 5.

p = 965627713414686037773998887356363665495489701319
 r = 965627713414686037773998839797242751054848709919
 a = 965627713414686037773998887356363665495489701316
 b = 69258789294063637963571905734367242074831358211
 G = (180154915808782548921909613234120456530839054533,
322027385971935282160204922182313022993137363211)
 γ = 412034918207550707882137137021287579344082387488
 r^{tw} = 965627713414686037773998934915484579936130692721
 a^{tw} = 965627713414686037773998887356363665495489701316
 b^{tw} = 896368924120622399810426981621996423420658343108
 G^{tw} = (366500700833382456097277943560492718018620012876,
386244879790635939190694479506736774122111430939)