Public-Key Cryptosystems with Very Small Key Lengths

Greg Harper, Alfred Menezes and Scott Vanstone

Dept. of Combinatorics and Optimization, University of Waterloo
Waterloo, Ontario, N2L 3G1, Canada

Abstract. In some applications of public-key cryptography it is desirable, and perhaps even necessary, that the key size be as small as possible. Moreover, the cryptosystem just needs to be secure enough so that breaking it is not cost-effective. The purpose of this paper is to investigate the security and practicality of elliptic curve cryptosystems with small key sizes of about 100 bits.

1 Introduction

It is sometimes convenient for the users of a public-key cryptosystem that the key sizes be as small as possible. For the applications we have in mind it is the public part of the key which should be relatively small. The size of the public key is typically difficult to control. For example, in the RSA system [23] the public key consists of the integers e and n, where n is the modulus. Although e can be chosen to be small there is not the same flexibility with the choice of n (n should be at least 512 bits in length). For the Diffie Hellman [7] and ElGamal schemes [8] based on discrete exponentiation in a finite field, the private key k, an integer, can be restricted but the public key α^k, α a generator of the field, is the size of the field (which should certainly be at least 2^{500}). For the Chor-Rivest knapsack [6], the public key is $(a_0, a_1, \ldots, a_{p-1})$, where $0 \leq a_i \leq p^m - 2$ (one choice suggested in [6] is $p = 197$, $m = 24$). The next two paragraphs describe some situations which come to mind where a small public key size might be desirable.

Consider the scenario where we have a small network where we would like to have secure e-mail exchange, or where we would like to have secure transmission of messages by fax. Rather than exchange public keys using certificates, key exchange is done verbally with authentication provided by voice recognition. If a symbol set consists of 32 alphanumeric characters represented by all 5-bit vectors then an n-bit key can be exchanged by representing it as an $\lceil n/5 \rceil$-symbol alphanumeric string. For n about 100 such a string is less than twice the length of most current international telephone numbers. Strings of this length would also be convenient for business cards, letterheads etc.

Consider also the following scenario: A software company places various programs on one distribution medium, however the purchaser can only access those programs he has paid for (the distribution medium could contain special purpose hardware that is tamper-proof for this purpose). If the user later wishes

R.A. Rueppel (Ed.): Advances in Cryptology - EUROCRYPT '92, LNCS 658, pp. 163-173, 1993.

to purchase some of the other programs, he phones the company and places his request. The company in turn replies with the appropriate access information, which is digitally signed. The signature is verified by the user's terminal, and access is granted.

Most of the known public-key cryptosystems are totally insecure if the key size is restricted to about 100 bits. For example, since factoring 100-bit integers can be readily done on a microcomputer, the RSA system is insecure for keys of that size. The same holds true for systems whose security is based on the intractability of the discrete logarithm problem in a finite field, such as the ElGamal cryptosystem; recently La Macchia and Odlyzko [14] computed logarithms in the field F_p where p is a 192-bit prime, while Gordon and McCurley [9] were able to compute logarithms in $F_{2^{401}}$.

A good candidate that remains is the elliptic curve cryptosystem, which was first proposed in 1985 by N. Koblitz [11] and V. Miller [18]. The security of these systems is based on the difficulty of the logarithm problem in the elliptic curve group. If the curve is carefully chosen, then the only known attacks on the problem are the so-called square root attacks whose running times are proportional to the square root of the largest prime dividing the order of the group. The most efficient way known to implement this algorithm is the Pollard ρ-method [22] which requires very little storage.

The paper is organized as follows. We begin with a review of elliptic curve cryptosystems in Section 2. We outline Pollard's method in Section 3, and present some results of our experiments. In Section 4 we compare two ways of performing finite field arithmetic in software. Finally, in Section 5 we present the results of our software implementation of the elliptic curve cryptosystems. All implementations were done in the C-language on a SUN SPARCstation-2.

2 Elliptic Curve Cryptosystems

We will be concerned here with elliptic curves over fields of characteristic 2. Some recent work has been done on the implementation of cryptosystems based on these curves [12, 13, 16]. For an elementary introduction to elliptic curves consult [10].

We let $q = 2^n$. Let E be a non-supersingular elliptic curve defined over the field F_q. There are precisely $2(q-1)$ such curves, whose defining equations have the form

$$E : y^2 + xy = x^3 + ax^2 + b, \tag{1}$$

where $b \in F_q^*$, and $a \in \{0, \gamma\}$, $\gamma \in F_q$ being an element of trace 1. The set of solutions (x, y) in $F_q \times F_q$ to the equation (1), together with the special point at infinity, denoted $\mathcal{O}$, form an (additively written) abelian group denoted by $E(F_q)$. By Hasse's Theorem, the order of this group is $\#E(F_q) = q + 1 - t$, where $|t| \leq 2\sqrt{q}$, and hence $\#E(F_q) \approx q$. It is this group that is used instead of the multiplicative group of a finite field to implement the Diffie-Hellman key passing, and the ElGamal message passing and signature schemes as described in [11, 16].

The security of the elliptic curve schemes is based on the presumed difficulty of computing logarithms in E; namely the problem of finding the integer k given the points $P \in E(F_q)$ and $Q = kP$. We assume that the order of P (and thus also $\#E(F_q)$) is divisible by a large prime p. To avoid the recent attack of [15], the curve must also satisfy the condition that p does not divide $q^l - 1$ for all small l (i.e. for all $l \leq s$, where the discrete logarithm problem in F_{2^s} is considered intractable). If these criteria are met, then the best known attack is the combination of the Pollard ρ and Pohlig-Hellman methods.

To select an appropriate curve one may simply pick random curves over F_{2^n} and compute $\#E(F_{2^n})$ by Schoof's algorithm [24], until the conditions of the previous paragraph are met. Schoof's algorithm appears practical for $n \leq 155$ as demonstrated by the computations in [17]. An alternate method is to pick a curve defined over a small subfield K of F_{2^n}, count the number of points over K directly, and then lift the result to F_{2^n} by using the Weil Conjecture (see [10]).

3 Pollard ρ-Method

We outline the method for computing logarithms in an elliptic curve group which is a combination of the methods of Pohlig-Hellman [21] and Pollard [22].

Let $P \in E(F_q)$ and $Q = kP$. We assume that the order m of P is known as is its prime factorization $m = p_1^{e_1} p_2^{e_2} \cdots p_t^{e_t}$. We first show how to determine k modulo p^e, where p is one of $p_1, p_2, \ldots, p_t$. One may similarly compute k modulo $p_i^{e_i}$, $1 \leq i \leq t$, and then use the Chinese remainder theorem to recover k.

Let $z = k \pmod{p^e}$, and write $z = \sum_{i=0}^{e-1} z_i p^i$, where $0 \leq z_i < p$. We have

$$\frac{m}{p} Q = \frac{km}{p} P = z\left(\frac{m}{p} P\right) = z_0\left(\frac{m}{p} P\right);$$

the latter two equations hold since $(m/p)P$ has order p. Using the method to be described below, we determine z_0, the logarithm of $(m/p)Q$ to the base $(m/p)P$, which has order p. To find z_1, we observe that

$$\frac{m}{p^2}(Q - z_0 P) = \frac{m}{p^2} Q - z_0\left(\frac{m}{p^2} P\right) = k\left(\frac{m}{p^2} P\right) - z_0\left(\frac{m}{p^2} P\right) = z_1\left(\frac{m}{p} P\right).$$

Again, by the method described below, we may then compute z_1, and continue.

We can now assume that the order of P is a prime p. If $R = (x, y)$, let $wt(R)$ denote the sum modulo 3 of the Hamming weights of the binary representations of x and y (with $wt(\mathcal{O}) = 0$). Define a sequence of points $\{R_i\}$ by $R_0 = \mathcal{O}$, and

$$R_{i+1} = Q + R_i,\ 2R_i,\ \text{or}\ P + R_i, \quad \text{if } wt(R_i)=0,1,\text{ or } 2 \text{ respectively.} \qquad (2)$$

If the sequence $\{R_i\}$ behaves like a random sequence, then the least index j with $R_j = R_{2j}$ has expected value close to $1.0308\sqrt{p}$ [22]. We may find this j by Floyd's cycle-finding algorithm as follows. Define integer sequences $\{a_i\}$, $\{b_i\}$,

by $a_0 = 0$, $b_0 = 0$, and $a_{i+1} = a_i + 1, 2a_i$ or $a_i \pmod p$, $b_{i+1} = b_i, 2b_i$ or $b_i + 1 \pmod p$, according to the three cases in (2). Observe that

$$R_i = a_i Q + b_i P \quad \text{for } i \geq 0.$$

We now compute the sequence $(R_i, a_i, b_i, R_{2i}, a_{2i}, b_{2i})$ for $i = 1, 2, 3, \ldots$ until we have $R_i = R_{2i}$. We then have $(a_i - a_{2i})Q = (b_{2i} - b_i)P$, which yields the congruence

$$(a_i - a_{2i})k \equiv (b_{2i} - b_i) \pmod p.$$

This congruence can then be solved for k.

A faster way to find indices i, j, $i \neq j$, with $R_i = R_j$ is to use Brent's cycle-finding algorithm [3], which is never slower and is on average 36% faster than Floyd's algorithm. Let the sequences $\{R_i\}$, $\{a_i\}$ and $\{b_i\}$ be defined as above. Brent's algorithm is the following.

Let $Y := R_0$; $r := 1$; $i := 0$; *done* := **false**;
repeat $X := Y$; $j := i$; $r := 2r$;
  repeat $i := i + 1$; $Y := R_i$; *done* $:= (X = Y)$
  until *done* **or** $(i \geq r)$
until *done*

When the algorithm terminates, $X = R_j$, $Y = R_i$, and $R_i = R_j$. We then have $(a_i - a_j)Q = (b_j - b_i)P$, which yields the congruence $(a_i - a_j)k \equiv (b_j - b_i) \pmod p$. This congruence is then solved for k. The expected value of i is $1.9828\sqrt{p}$, assuming that $\{R_i\}$ behaves like a random sequence [3].

For cryptographic applications, we only consider curves whose order is divisible by a large prime p. The running time of the algorithm is then dominated by the time to compute logarithms modulo this p. If we count elliptic curve additions as a basic step, then the expected running time is $1.9828\sqrt{p}$ curve additions with Brent's algorithm and $3.0924\sqrt{p}$ curve additions with Floyd's algorithm (since it takes 3 curve additions to compute the pair of points (R_{i+1}, R_{2i+2}) from (R_i, R_{2i})). Note that the amount of storage required by the algorithm is negligible.

We have implemented the Pohlig-Hellman and Pollard ρ-methods for computing logarithms in elliptic curves over F_{2^n}. Field elements are represented with respect to an optimal normal basis over F_2 (see Section 4). Tables 1, 2 and 3 give the average running times for computing logarithms in various curves over the fields $F_{2^{65}}$, $F_{2^{88}}$ and $F_{2^{105}}$.

4 Field Arithmetic

There are various methods for performing arithmetic in a finite field. The most popular seem to be through polynomial and normal basis representations. In this section we have attempted to compare software implementations for specific fields. It is extremely difficult to state with confidence which method is better

Order of curve	Number of digits in largest prime factor	Observed time (in minutes)
$2^3 \cdot 3 \cdot 79 \cdot 21755761 \cdot 894410947$	9	5
$2^2 \cdot 3^2 \cdot 3845557 \cdot 266494324513$	12	53
$2^2 \cdot 11 \cdot 2003 \cdot 418616258967271$	15	2625
$2^2 \cdot 7 \cdot 313 \cdot 4209663184776557$	16	5538

Table 1. Average times to compute logarithms in curves over $F_{2^{65}}$

Order of curve	Number of digits in largest prime factor	Observed time (in minutes)
$1360932^3 \cdot 7^2 \cdot 109 \cdot 419 \cdot 5441 \cdot 30779339$	8	1
$2^2 \cdot 3 \cdot 2309 \cdot 902977 \cdot 53353603 \cdot 463686011$	9	2
$2^3 \cdot 5 \cdot 23 \cdot 61 \cdot 1009 \cdot 2017 \cdot 135559 \cdot 39978503273$	11	11
$2^3 \cdot 3^2 \cdot 9049 \cdot 510843049 \cdot 1859726335343$	13	149
$2^2 \cdot 37 \cdot 467 \cdot 547 \cdot 942593 \cdot 17369186627491$	14	423
$2^3 \cdot 3 \cdot 71 \cdot 821 \cdot 646039 \cdot 684854210761471$	15	2268

Table 2. Average times to compute logarithms in curves over $F_{2^{89}}$

since any implementation is machine and code dependent. With this caveat we proceed.

We compare the speeds in software of the basic operations of multiplication, squaring, and inversion in the fields $F_{2^{105}}$ and $F_{2^{104}}$, the former represented with respect to an optimal normal basis over F_2, and the latter represented with respect to a polynomial basis over F_{2^8}.

Order of curve	Number of digits in largest prime factor	Observed time (in minutes)
$2^2 \cdot 11 \cdot 29 \cdot 43 \cdot 211 \cdot 421 \cdot 751 \cdot 1051 \cdot 15541 \cdot 25621 \cdot 26481841$	8	6
$2^3 \cdot 3^2 \cdot 457 \cdot 212913163 \cdot 1431658159 \cdot 4044445651$	10	17
$2^2 \cdot 7^4 \cdot 13^2 \cdot 1009 \cdot 131251 \cdot 173741 \cdot 1086211970677$	13	172
$2^3 \cdot 31 \cdot 71^2 \cdot 131 \cdot 211 \cdot 281 \cdot 3697 \cdot 1129982311077901$	16	3324

Table 3. Average times to compute logarithms in curves over $F_{2^{105}}$

4.1 Normal Basis Representation

A normal basis for $F_{2^{105}}$ over F_2 is a basis of the form

$$N = \{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{104}}\}.$$

The basis is optimal if its multiplication table is as "simple" as possible; see [19] for more details and for an easy way to construct such a basis. Given any $\alpha \in F_{2^{105}}$, we can then express $\alpha = \sum_{i=0}^{104} c_i \beta^{2^i}$, where $c_i \in F_2$, and we write $\alpha = (c_0, c_1, c_2, \ldots, c_{104})$. In software, α is represented by a bit vector of length 105, i.e. on a 32-bit machine, α is stored in an array of unsigned integers of length 4, the last 23 bits of which are unused.

Addition of elements is achieved by simply XOR-ing the vector representations. Since

$$\alpha^2 = \sum_{i=0}^{104} c_i \beta^{2^{i+1}} = \sum_{i=0}^{104} c_{i-1} \beta^{2^i}$$

(with indices reduced modulo 105), squaring α is accomplished by a cyclic shift of its vector representation.

The most efficient way we know to compute the inverse of α is to first convert to a polynomial basis representation of $F_{2^{105}}$ using a precomputed change of basis matrix, compute the inverse using an efficient implementation of the extended Euclidean algorithm as described in [2, p.41], and then transform the result back to the normal basis representation.

It has been our experience that a software implementation of an optimal normal basis is more efficient that implementing an arbitrary normal basis. The field $F_{2^{105}}$ was chosen because it is about 100 bits and an optimal normal basis exists for it.

4.2 Polynomial Basis Representation

We choose to represent $F_{2^{104}}$ as a vector space over F_{2^8}, instead of over F_2 as is usually done. The polynomial

$$g(z) = 1 + z^2 + z^3 + z^7 + z^8$$

is a primitive polynomial over F_2, and so the elements of F_{2^8} are the set of polynomials $F_2[z]$ modulo $g(z)$. For efficiency, we store two tables "log" and "antilog" which are defined as follows:

$$\log[a] = i, \quad \text{where } z^i = \alpha \text{ and } 0 \le i \le 254$$

and

$$\text{antilog}[i] = a, \quad \text{where } z^i = \alpha \text{ and } 0 \le i \le 254.$$

(a is the binary vector representation of $\alpha \in F_{2^8}^*$.) Multiplication in F_{2^8} is then simply accomplished by table-lookup. For, if $a, b \in F_{2^8}^*$, then

$$a \cdot b = \text{antilog}[(\log[a] + \log[b]) \bmod 255].$$

We also store a table of inverses of elements in $F_{2^8}^*$.

The polynomial

$$f(x) = 1 + x + x^6 + x^7 + x^{13}$$

is irreducible over F_2, and thus also over F_{2^8} since $\gcd(8, 13) = 1$. Consequently, we may represent the elements of $F_{2^{104}}$ as polynomials in $F_{2^8}[x]$ of degree at most 12 modulo $f(x)$. Each coefficient of the polynomial is stored in a single computer word.

If $a = \sum_{i=0}^{12} a_i x^i$, $b = \sum_{i=0}^{12} b_i x^i \in F_{2^{104}}$, then we compute the product $c = a \cdot b$ by initializing c to 0, and then computing $a, ax, \ldots, ax^{12}$; after each stage, we add $b_i(ax^i)$ to c. Squaring is faster than a multiplication since $a^2 = \sum_{i=0}^{12} a_i^2 x^{2i}$. Finally, the inverse of a is computed by applying the extended Euclidean algorithm in $F_{2^8}[x]$ to find $s(x)$ such that $a(x)s(x) \equiv 1 \pmod{f(x)}$.

4.3 Timings

Arithmetic in the fields $F_{2^{105}}$ and $F_{2^{104}}$ was implemented. In Table 4 we give the running times for the operations of squaring, multiplication and inversion. The arithmetic in $F_{2^{104}}$ was observed to be faster than in $F_{2^{105}}$ in our implementations, and this is primarily due to our choice of representing $F_{2^{104}}$ as a vector space over F_{2^8}, the arithmetic in F_{2^8} being essentially for free.

	$F_{2^{105}}$	$F_{2^{104}}$
10,000 squarings	0.02	0.15
10,000 multiplications	7.6	1.95
10,000 inversions	15.3	9.46

Table 4. Times for field operations (in seconds)

5 Implementation of an Elliptic Curve Cryptosystem

For a curve with equation (1), the rules for adding points are the following. The point $\mathcal{O}$ serves as the identity element. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be points on the curve. Then $-P = (x_1, x_1 + y_1)$. If $Q \neq -P$, then $P + Q = (x_3, y_3)$, where

$$x_3 = \begin{cases} \left(\dfrac{y_1 + y_2}{x_1 + x_2}\right)^2 + \dfrac{y_1 + y_2}{x_1 + x_2} + x_1 + x_2 + a_2 & P \neq Q \\ \dfrac{a_6}{x_1^2} + x_1^2 & P = Q \end{cases}$$

and

$$y_3 = \begin{cases} \left(\frac{y_1+y_2}{x_1+x_2}\right)(x_1+x_3)+x_3+y_1 & P \neq Q \\ x_1^2+\left(x_1+\frac{y_1}{x_1}\right)x_3+x_3 & P=Q. \end{cases}$$

Note that two multiplications and an inversion are needed to add two distinct points, while a point can be doubled in 3 multiplications and an inversion (the other operations are relatively inexpensive). We should also note that inversions can, in general, be avoided by changing to projective coordinates at the expensive of doing more multiplications. Hardware implementations (see for example [1]) can take advantage of this whereas, in software, the affine representation appears to be superior.

Table 5 lists the running times of the elliptic curve operations for random curves over $F_{2^{105}}$ and $F_{2^{104}}$ (which are represented as described in Section 4).

	$F_{2^{105}}$	$F_{2^{104}}$
10,000 curve additions	33.9	14.4
10,000 curve doublings	38.6	16.2

Table 5. Times for elliptic curve operations (in seconds)

For each of the Diffie-Hellman key passing and ElGamal message passing and signature schemes, one has to compute kP for a random integer k. This is accomplished by the repeated doubling and adding algorithm. When $P \in E(F_{2^{104}})$, k will have 52 bits on average that are 1 in its binary representation. Hence computing kP requires 103 curve doublings and 51 curve additions on average, for an expected time of .24 seconds when working with $E(F_{2^{104}})$. This time is certainly tolerable for many applications, including the ones described in the Introduction.

If the cryptosystems are implemented in software, then there is additional storage available. In this case, the following method of [5] is more efficient for computing kP. We first precompute the points $P_i = 8^i P$ for $0 \leq i \leq 34$. Let $1 \leq k \leq \#E(F_{2^{104}})$. We write k in its base 8 representation as

$$k = k_0 + k_1 8 + k_2 8^2 + \cdots + k_{34} 8^{34}, \quad 0 \leq k_i \leq 7.$$

(This is an easy task given the binary representation of k.) The algorithm to compute kP is the following:

(i) Compute $B := \sum_{k_i=7} P_i$, and set $A := B$.
(ii) For d from 6 to 1 do
$B := B + \sum_{k_i=d} P_i$
$A := A + B$

(iii) Output A

The method requires the storage of 35 points and needs 36 curve additions on average for random k. This reduces our time estimate for computing kP from .24 seconds to .052 seconds.

In the elliptic curve analogue of the ElGamal cryptosystem, a curve $E(F_q)$ and point $P \in E$ are public knowledge. Each user A has a public key aP, where integer a is the corresponding secret key. To send messages m_1, $m_2 \in F_q$ to A, user B selects a random integer k, and computes the points kP and $kaP = (\overline{x}, \overline{y})$. Finally, B sends $(kP, m_1\overline{x}, m_2\overline{y})$ to A. With our software implementation, the encryption rate is about 2 Kbits/sec.

The public key is an elliptic curve point which is $2n$ bits long (where $q = 2^n$). The key length can be shortened to $n+1$ bits as follows. Observe first that the change of variables $(x, y) \longrightarrow (x, xz)$ transforms equation (1) to

$$z^2 + z = x + a + bx^{-2}. \tag{3}$$

Given the x-coordinate of a point $P = (\overline{x}, \overline{y})$, we can compute the right hand side of (3). Then (3) has precisely 2 solutions, namely z' and $z' + 1$, and these solutions can be easily found. We can then select the correct solution $\overline{z}$ (and hence reconstruct $\overline{y}$ as $\overline{y} = \overline{x}\overline{z}$) if we know the least significant bit of $\overline{z}$. Thus to transmit P it is sufficient to transmit $\overline{x}$ and the least significant bit of $\overline{y}/\overline{x}$.

If E is a curve defined over F_{2^s} with equation (1), then $\#E(F_{2^s})$ is even, and $226 \leq \#E(F_{2^s}) \leq 228$. If $\#E(F_{2^s}) = 242$, 250 or 254, then $\#E(F_{2^{104}})$ is divisible by a 29 digit prime. For example, the following curve

$$E : y^2 + xy = x^3 + zx^2 + (z^3 + z^4 + z^6 + z^7) \tag{4}$$

(with z defined as in Section 4.2), has $\#E(F_{2^s}) = 250$, and

$$\#E(F_{2^{104}}) = 2 \cdot 11^2 \cdot 838116099324449169054045134 41,$$

where the last integer is a 29-digit prime.

Based on our experimental results of Section 3, computing even a single logarithm in $E(F_{2^{104}})$ using Pollard's ρ-method is computationally infeasible. For a very rough analysis of the security, observe that the Pollard ρ-method requires about 10^{15} elliptic curve operations to compute a single logarithm in the curve (4). A rough count shows that an elliptic curve operation takes at least 1000 word operations (i.e. operations such as exclusive-or's on 32-bit words). Hence the expected number of word operations to compute a logarithm is at least 10^{18}. This is roughly equal to the computing resources required to factor a 512-bit integer by the multipolynomial quadratic sieve, and for computing logarithms in the field $F_{2^{700}}$ by the index-calculus method (see [25]).

As noted by Brent [4], it does not seem possible to use parallelism to speed up the Pollard ρ-method. The Lambda method [22] is a method for computing logarithms given that the logarithm lies in some interval $[A, B]$. The running time of the method is $O(\sqrt{w})$, where $w = B - A$. If t processors are available, then to compute $\log_P Q$ where the order of P is a prime p, the interval $[0, p-1]$

is divided into t intervals of equal length, and the intervals are searched in parallel by the t processors. The expected number of elliptic operations for each processor is $\sqrt{p/t}$. For example, if 10,000 processors are available then for the curve (4) the expected number of elliptic curve operations is about 10^{13}. If this does not provide adequate security for the application at hand, then of course the underlying field can be replaced by a larger field $F_{2^{8r}}$, $r \geq 14$ (to use the polynomial basis representation of Section 4.2 we need to work in a field F_{2^n} where n is a multiple of 8).

6 Conclusions

We have demonstrated that implementing elliptic curve cryptosystems in F_{2^n} for $n \approx 100$ can be done efficiently in software on a workstation. Curves over $F_{2^{104}}$ can be selected so that by current best methods computing logarithms requires about 10^{15} elliptic curve operations (if additional security is required then, of course, n can be increased). It should be pointed out that each logarithm in such a system requires this amount of work. This is unlike the index-calculus method [20] where there are 2 phases. Once phase one is completed all other logarithms are relatively easy to find. Computing kP, where $P \in E(F_{2^{104}})$ and k is a random 104-bit integer, takes about .052 seconds. We can thus implement the ElGamal cryptosystem and achieve an encryption rate of 2 Kbits/sec in software. Public keys are only 105 bits in length. The code for this scheme is fairly compact and occupies about 40 Kbytes.

References

1. G. Agnew, R. Mullin and S. Vanstone, "An implementation of elliptic curve cryptosystems over $F_{2^{155}}$", preprint, 1992.
2. E. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.
3. R. Brent, "An improved Monte Carlo factoring algorithm", *BIT*, **20** (1980), 176-184.
4. R. Brent, "Parallel algorithms for integer factorisation", in *Number Theory and Cryptography*, Cambridge University Press, 1990, 26-37.
5. E. Brickell, D. Gordon, K. McCurley and D. Wilson, "Fast exponentiation with precomputation", preprint, 1992.
6. B. Chor and R. Rivest, "A knapsack-type public key cryptosystem based on arithmetic in finite fields", *IEEE Transactions on Information Theory*, **34** (1988), 901-909.
7. W. Diffie and M. Hellman, "New directions in cryptography", *IEEE Transactions on Information Theory*, **22** (1976), 644-654.
8. T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", *IEEE Transactions on Information Theory*, **31** (1985), 469-472.
9. D. Gordon and K. McCurley, "Computation of discrete logarithms in $GF(2^n)$", presentation at Crypto '91, Santa Barbara, 1991.
10. N. Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag, New York, 1987.

11. N. Koblitz, "Elliptic curve cryptosystems", *Mathematics of Computation*, **48** (1987), 203-209.
12. N. Koblitz, "Constructing elliptic curve cryptosystems in characteristic 2", *Advances in Cryptology: Proceedings of Crypto '90*, Lecture Notes in Computer Science, **537** (1991), Springer-Verlag, 156-167.
13. N. Koblitz, "CM-Curves with good cryptographic properties", *Advances in Cryptology: Proceedings of Crypto '91*, Lecture Notes in Computer Science, **576** (1992), Springer-Verlag, 279-287.
14. B. La Macchia and A. Odlyzko, "Computation of discrete logarithms in prime fields", *Designs, Codes and Cryptography*, 1 (1991), 47-62.
15. A. Menezes, T. Okamoto and S. Vanstone, "Reducing elliptic curve logarithms to logarithms in a finite field", *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing*, 80-89, 1991.
16. A. Menezes and S. Vanstone, "Elliptic curve cryptosystems and their implementation", submitted to *Journal of Cryptology*, 1991.
17. A. Menezes, S. Vanstone and R. Zuccherato, "Counting points on elliptic curves over F_{2^m}", to appear in *Mathematics of Computation*, 1992.
18. V. Miller, "Uses of elliptic curves in cryptography", *Advances in Cryptology: Proceedings of Crypto '85*, Lecture Notes in Computer Science, **218** (1986), Springer-Verlag, 417-426.
19. R. Mullin, I. Onyszchuk, S. Vanstone and R. Wilson, "Optimal normal bases in $GF(p^n)$", *Discrete Applied Mathematics*, **22** (1988/89), 149-161.
20. A. Odlyzko, "Discrete logarithms in finite fields and their cryptographic significance", *Advances in Cryptology – Proceedings of Eurocrypt '84*, Lecture Notes in Computer Science, **209** (1985), Springer-Verlag, 224-314.
21. S. Pohlig and M. Hellman, "An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance", *IEEE Transactions on Information Theory*, **24** (1978), 106-110.
22. J. Pollard, "Monte Carlo methods for index computation (mod p)", *Mathematics of Computation*, **32** (1978), 918-924.
23. R. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, **21** (1978), 120-126.
24. R. Schoof, "Elliptic curves over finite fields and the computation of square roots mod p", *Mathematics of Computation*, **44** (1985), 483-494.
25. P. van Oorschot, "A comparison of practical public key cryptosystems", in *Contemporary Cryptology*, IEEE Press, 1992, 289-322.