COUNTING THE POINTS OF AN ELLIPTIC CURVE ON A LOW-MEMORY DEVICE

JÁNOS A. CSIRIK

Abstract. An important but very memory consuming step in elliptic curve cryptography is that of coming up with an elliptic curve where the number of rational points has a large prime factor. This note describes how the Schoof–Elkies–Atkin algorithm can be carried out to count the number of points on a given elliptic curve over a (large) prime finite field using little memory. For field sizes between 160 and 256 bits, the process requires between 15 and 50 kilobytes of memory. This will allow for a more complete implementation of elliptic curve cryptography on smart cards and other low-memory devices.

1. Introduction

Elliptic curve public key cryptosystems ([9, 7, 1]) are popular since they (presumably) provide high security at low key lengths. The low key lengths and low computational complexity make them attractive for use in small devices. The choice of which elliptic curve to use involves a much more elaborate calculation and is typically not carried out on the device itself. Instead, pre-certified curves are used.

In this note we shall deal only with elliptic curves defined over finite fields of prime order, although finite fields of characteristic 2 are also often used in practice. The current state of belief is that for elliptic curves over the finite field $\mathbb{F}_p$, the difficulty of the discrete logarithm problem increases exponentially with the size of the largest prime factor of the number of rational points on the curve, except for a few special curves that can be detected merely by looking at the order (see [1, Chapter IV and the references therein]). If the status of the DLP problem does not change then it is reasonable to use stored curves. To guard against some (presently unknown) vulnerability of the stored curve, one might want the small device to generate its curves itself. This will also increase security by easily allowing a large number of such devices to all have different curves. A way to allow the devices to generate their own curves is described here.

The first polynomial time algorithm for counting the number of points on an elliptic curve over $\mathbb{F}_p$ was proposed by Schoof in [11]. This method was later improved by Elkies and Atkin to become the Schoof–Elkies–Atkin algorithm. For details on the theory and implementation of this algorithm, see [11, 4, 10, 3, 6]. Here we shall primarily use the exposition in [3] (which in turn is based on the algorithm as presented in [10]).

Given an elliptic curve E defined over $\mathbb{F}_p$, the number of rational points on E is

$$\#E(\mathbb{F}_p) = p + 1 - t,$$

where $|t| < 2\sqrt{p}$. The SEA algorithm proceds by calculating the modular polynomial $\Psi_\ell(F, J)$ for various small primes ℓ and using it to derive information about t modulo ℓ. These two steps will be referred to as the *modular curve calculation* and the *elliptic curve calculation*, respectively. When enough information has been gathered, the baby step–giant step algorithm (and the Chinese Remainder Theorem) can be used to determine t. The crucial observation is that only Ψ_ℓ mod p is used and that it can be calculated on the fly for each ℓ. Also, the memory in the modular curve calculation can largely be freed up for the elliptic curve calculation. The approximate amount of memory used for primes p of various sizes to get $\#E(\mathbb{F}_p)$ with a probability of at least $1/2$ is given here[1]:

bits in p	kB
160	15
192	24
224	40
256	50

The above sizes for p should provide adequate security for the immediate future, for more details see [8]. Increasing the amount of memory used allows the calculation to succeed with a higher probability. A test implementation on a slow Pentium computer took on the order of an hour per curve.

Note however that the above figures don't account for the size of the code implementing the algorithm. Taking into account the fact that an elliptic curve crypography device will already have subroutines fore dealing with finite field and elliptic curve arithmetic, we estimate that the code is of comparable size to the memory used for data.

Since the actual memory use for both code and data will be highly dependent on the implementation, we shall resist the temptation of going into more refined variants of the algorithm which increase the size of the code by more than their savings in data storage, for example the use of Atkin primes. The various improvements in the references might of course be added "to taste" by the implementer. Similar remarks apply to fine-tuning the implementation (such as substituting Pollard's rho method for the baby step–giant step process), and in particular, making sure that our device does not run out of power before the calculation ends.

Plug: Readers of this paper will almost certainly be interested in the paper by A. K. Lenstra and E. Verheul about their XTR system.

2. The Algorithm

We shall describe various ways the algorithm described in [3] needs to be modified to lower memory consumption. We shall also calculate the memory use at each step in terms of 32-bit words. A word was chosen as a unit of measurement for convenience, notwithstanding the fact that our small device might well contain an 8-bit processor.

Given an elliptic curve E defined over $\mathbb{F}_p$, we need to find Ψ_ℓ modulo p. Let N denote the length (in words) of p; for example, $N = 5$ if the length of p is 160 bits. It turns out that a very memory-efficient way to get Ψ_ℓ is to calculate it modulo various 32-bit primes, get the coefficients over $\mathbb{Z}$ by the Chinese Remainder Theorem, and reduce them modulo p straight away.[2] To reduce the penalty associated with the

[1] If we wish to precalculate and store the modular polynomials Ψ_ℓ, then the memory requirements in the four cases treated go up to 35, 75, 140 and 200 kB respectively

[2] The optimal solution would be to vary the length of the modulus according to how far along we are in the calculation, but that only yields a small decrease in storage over this more simplistic method.

calculation of Ψ_ℓ on the fly, the algorithm should be run on several elliptic curves simultaneously, perhaps as many as needed to find a curve of the right security level (see [5]). We shall see that this will increase the memory use only insignificantly.

Let $R(\ell)$ denote the length (in words) of the longest coefficient of Ψ_ℓ. In each case, $R(\ell)$ is also the number of 32-bit primes we need to get Ψ_ℓ. As we have seen in [3], we need to store $T(\ell) = 8\ell v + \ell + 3$ elements of a given finite field in order to determine Ψ_ℓ over that finite field, where $v = v(\ell)$ is as defined in [3] (it is less than $\ell/24$). The number of coefficients in Ψ_ℓ that need to be evaluated is $C(\ell) = 3(v+1)(\ell+1)/2 + 1$ (the unique coefficient that does not need to be laboriously worked out is that of $F^{\ell+1}$, since it is always one). Since $C(\ell) < T(\ell)$, the number of words of memory used for the whole Chinese Remainder Theorem calculation is

$$(R-1)C + T.$$

In the next step of the algorithm, we need to find the linear factors of $f(F) = \Psi_\ell(F, j)$ for the j-invariants of the elliptic curves under consideration. Since the degree of f is $\ell + 1$, we can calculate the linear factors using $4(\ell+1)N$ words of memory, by taking greatest common divisors with $F^p - F$ modulo f, $(F+k)^{(p-1)/2} - 1$ modulo f for various small integers k. Finding the roots of $\Psi_\ell(f, J)$, which is of degree $2v$, can clearly accomplished using strictly less storage. Since we also need to store Ψ_ℓ, the memory use (expressed in words) of this step is

$$CN + 4(\ell+1)N.$$

For each of the curves where ℓ is an Elkies prime, we can clearly get the parameters $(\tilde{a}_4, \tilde{a}_6, p_1)$ by using less memory than above. Therefore, the memory use of the whole modular curve calculation is

$$\max((R-1)C + T, CN + 4(\ell+1)N).$$

For the elliptic curve calculation, we can get from $(\tilde{a}_4, \tilde{a}_6, p_1)$ to the kernel polynomial $h(X)$ using less than $4(\ell+1)N$ words of storage. The actual calculation of e can be carried out using no more than 8 polynomials modulo $h(X)$ at any one time (see [3]), using $8\deg(h)N = 4(\ell-1)N$ words of memory. Using the isogeny cycles of [2], we can get t modulo ℓ^k using

$$4(\ell^k - 1)N$$

words of memory.

The only remaining step that takes a significant amount of storage is the baby step–giant step process when all the auxiliary primes ℓ are used up. If there are only λ^2 possibilities for t then a list of λ baby steps and λ giant steps will do. In fact only one list needs to be stored, and clearly it suffices to store only the x-coordinates of the points in that list.[3] The memory use is

$$\lambda N.$$

We remark here that once the point counting is done, we will typically want to check if the order of the curve as a large prime factor. This can be accomplished using very little memory by dividing out all the small factors (up to, say, 50), and subjecting the resulting quotient to a probabilistic primality test.

[3]It might be even better to use a hash list here if the bigger list size does not slow the calculation down too much.

3. The choice of parameters

The table below presents all the necessary information needed to make our choices for various lengths of p. All the columns are self-explanatory, except possibly the third, which shows $\log_2$ of the absolute value of the longest coefficient in Ψ_ℓ.

ℓ	$v(\ell)$	$\log_2(\lvert\Psi_\ell\rvert_\infty)$	$R(\ell)$	ℓ	$v(\ell)$	$\log_2(\lvert\Psi_\ell\rvert_\infty)$	$R(\ell)$
29	1	38.7	2	101	2	109.0	4
31	1	40.2	2	103	3	177.3	6
41	1	41.2	2	107	3	170.0	6
47	1	56.2	2	131	2	139.9	5
59	1	57.2	2	109	3	161.8	6
53	2	104.3	4	97	4	195.5	7
71	1	70.6	3	113	4	196.0	7
61	2	91.9	3	151	3	191.6	7
79	2	100.1	4	127	4	237.7	8
83	2	109.7	4	167	3	254.8	8
89	2	111.2	4	139	4	219.7	7
73	3	130.1	5	149	4	234.8	8

For $p \approx 2^{160}$, we shall try to determine t modulo 2^7, 3^4, 5^3, 7^2, 11^2, 13^2, 17, 19, 23, 29, 31, 41, 47, 53, 59, 61, 71, 73, 79, 83, 89, 101. The maximal storage requirement for the modular curve calculation is 3830 words (at $\ell = 101$), the maximal storage requirement for the elliptic curve calculation is 3360 words (at $\ell = 13$). The total memory use therefore is 3830 words, or 15 kilobytes, allowing the use of the baby step–giant step algorithm with $\lambda = 766$. For the other suggested sizes for p, the calculations work similarly, and the moduli for determining t are listed below.

2^{160}	2^7,3^4,5^3,7^2,11^2,13^2,17,19,23,29,31, 41,47,53,59,61,71,73,79,83,89,101
2^{192}	2^7,3^5,5^3,7^2,11^2,13^2,17,19,23,29,31,41,47,53, 59,61,71,73,79,83,89,101,103,107,131,109
2^{224}	2^8,3^5,5^3,7^3,11^2,13^2,17^2,19,23,29,31,41,47,53,59,61, 71,73,79,83,89,101,103,107,131,109,97,113,151,127
2^{256}	2^8,3^5,5^3,7^3,11^2,13^2,17^2,19^2,23,29,31,41,47,53,59,61,71,73, 79,83,89,101,103,107,131,109,97,113,151,127,167,139,149

References

[1] Ian F. Blake, Gadiel Seroussi, and Nigel Smart. *Elliptic Curves in Cryptography*, volume 265 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1999.

[2] J.-M. Couveignes and François Morain. Schoof's algorithm and isogeny cycles. In Leonard M. Adleman and Ming-Deh Huang, editors, *Algorithmic number theory, Proceedings of the First International Symposium (ANTS-I) held at Cornell University, Ithaca, New York, May 6–9, 1994*, number 877 in Lecture Notes in Computer Science, pages 43–58. Springer, Berlin, 1994.

[3] János A. Csirik. An exposition of the SEA algorithm. preprint, 2000.

[4] Noam D. Elkies. Elliptic and modular curves over finite fields and related computational issues. In D. A. Buell and J. T. Teitelbaum, editors, *Computational Perspectives in Number Theory: Proceedings of a Conference in Honor of A. O. L. Atkin, (Chicago, IL, 1995)*, pages 21–76. AMS, 1998.

[5] Steven Galbraith and James McKee. The probability that the number of points on an elliptic curve over a finite field is prime. preprint CORR 99–51, University of Waterloo, Centre for Applied Cryptographic Research, 1999.

[6] T. Izu, J. Kogure, M. Noro, and K. Yokoyama. Efficient implementation of Schoof's algorithm. In Kazuo Ohta and Dingyi Pei, editors, *Proceedings of ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 66–79. Springer, 1998.

[7] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[8] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. preprint, 1999.

[9] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *Advances in cryptology—CRYPTO '85, Proceedings of the conference on the theory and application of cryptographic techniques (CRYPTO '85) held at the University of California, Santa Barbara, Calif., August 18–22, 1985*, pages 417–426. Springer, Berlin, 1986.

[10] François Morain. Calcul du nombre de points sur une courbe elliptique dans un cops fini: aspects algorithmiques. *Journal de Théorie des Nombres de Bordeaux*, 7:255–282, 1995.

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

AT&T Shannon Lab, 180 Park Ave, Florham Park NJ 07932-0971
E-mail address: janos@research.att.com