Schoof's Original Algorithm is Practical for Elliptic Curves of Cryptographic Size

Nikki Pitcher
supervised by Daniel J. Bernstein

July 6, 2006

Abstract

In 1985, Schoof's algorithm for counting points on elliptic curves was introduced. It is widely believed that Schoof's original algorithm is not practical for use with elliptic curves of cryptographic size — currently between 160 bits and 256 bits. For example, consider the following statements:

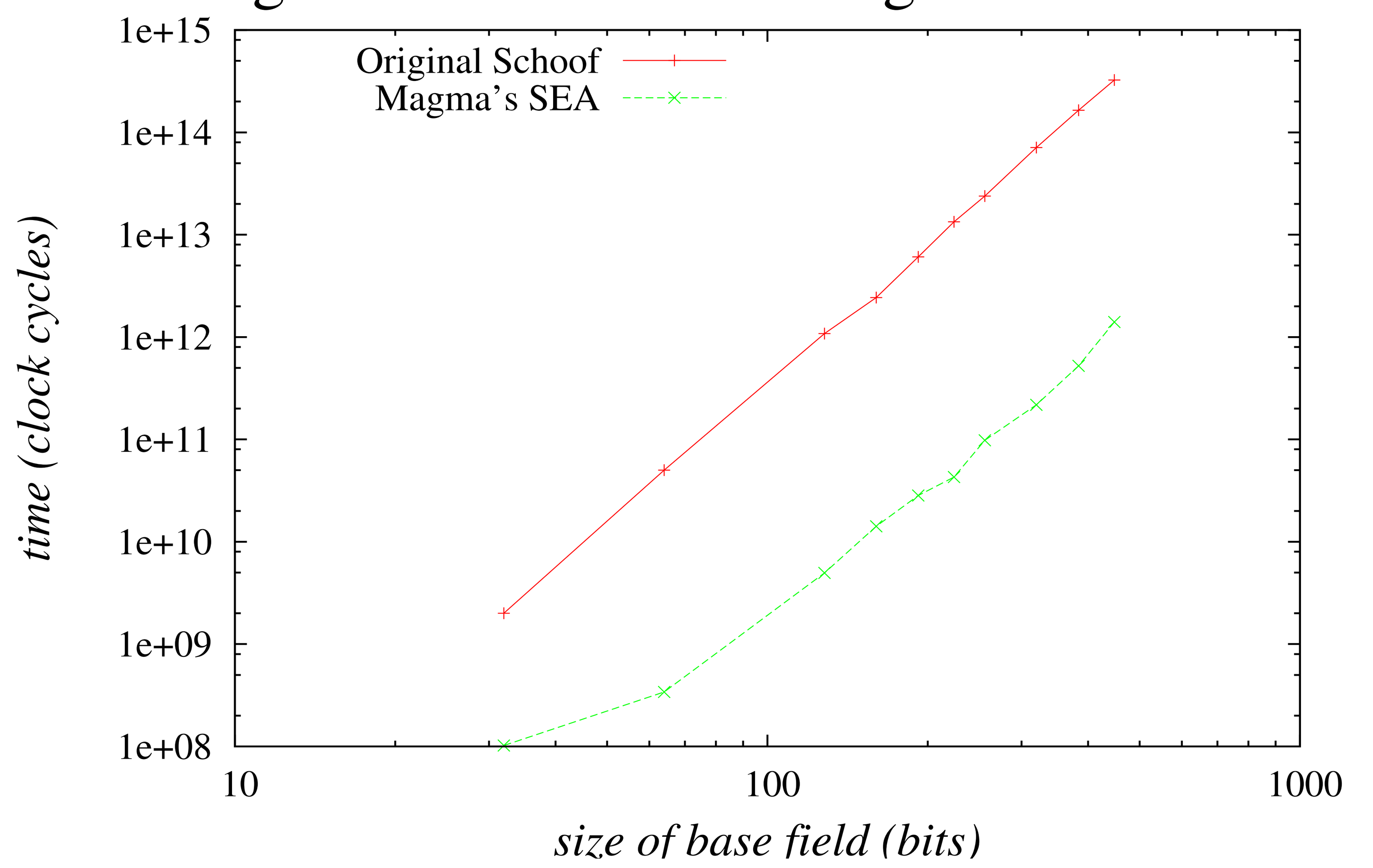
> Schoof, "Counting points on elliptic curves over finite fields," Journal de Théorie des Nombres de Bordeaux **7** (1995), page 219: "This deterministic polynomial time algorithm was impractical in its original form."
>
> Couveignes, Dewaghe, and Morain, "Isogeny cycles and the Schoof-Elkies-Atkin algorithm," LIX/RR/96/03 (1996), page 1: "From a practical point of view, the problem is the size of the torsion polynomials. Indeed, $f_\ell^E(x)$ is of degree $O(\ell^2)$. In practice one cannot hope to compute t mod ℓ in this way for $\ell > 31$, say."
>
> Blake, Seroussi, and Smart, "Elliptic curves in cryptography," London Mathematical Society (1999), pages 111–112: The benefit of fast multiplication in Schoof's original algorithm is "mostly theoretical, and hard to realize in practical implementations"; the algorithm "will generally not suffice for the parameter ranges of practical interest."
>
> Vanstone, 5 July 2004 talk discussing Schoof's algorithm in the 1990s, according to notes by Bernstein: "For cryptographically interesting curves it just couldn't be used."

As a consequence, it is widely believed that point counting became practical only after Schoof's algorithm was improved by Elkies and Atkin.

Schoof's original algorithm is, in fact, practical for use with cryptographic-sized elliptic curves. For example, my implementation of Schoof's original algorithm (without improvements due to Elkies, Atkin, or Baby-Step Giant-Step) computes $\#\boldsymbol{E}(\boldsymbol{F_p})$ in 1285 seconds (on a 2000MHz Athlon 64 X2) for a 160-bit prime p, and in 11948 seconds for a 256-bit prime p. The 160-bit computation could have been performed in under a day on a computer available in 1985. For comparison, Magma's implementation of the Schoof-Elkies-Atkin algorithm takes 413 seconds for a 448-bit prime p on a 3400MHz Pentium 4. Certainly the Elkies and Atkin improvements are valuable, but these improvements are not as drastic as is widely believed.

The speed of Schoof's original algorithm is of interest not only for historical reasons, but also for other applications of the underlying computational techniques, such as counting points in the Jacobian of a genus-2 hyperelliptic curve.

Original Schoof versus Magma's SEA Times



Methodology

The software reported in this poster computes $\#\boldsymbol{E}(\boldsymbol{F_p})$ given an odd prime p and an elliptic curve $\boldsymbol{E}$ over $\boldsymbol{F_p}$ in short Weierstrass form $y^2 = x^3+ax+b$. The software has several levels of subroutines that work as follows:

To multiply in $\mathbb{Z}$: Use the GMP package.

To multiply in $\mathbb{Z}[x]$, input coefficients below p: Evaluate polynomials at $x = 256^{2\lfloor \log_{256} p \rfloor + 6}$, multiply in $\mathbb{Z}$, and extract the product in $\mathbb{Z}[x]$.

To multiply in $(\mathbb{Z}/p\mathbb{Z})[x]$: Multiply in $\mathbb{Z}[x]$ and reduce coefficients mod p by first multiplying a precomputed reciprocal of p.

To multiply in $(\mathbb{Z}/p\mathbb{Z})[x]/\psi_\ell$ where ℓ is an odd prime and ψ_ℓ is the ℓth division polynomial: Multiply in $(\mathbb{Z}/p\mathbb{Z})[x]$ and reduce mod ψ_ℓ by first multiplying a precomputed reciprocal of ψ_ℓ.

To multiply in $\boldsymbol{R} = ((\mathbb{Z}/p\mathbb{Z})[x]/\psi_\ell)[y])/(y^2 - x^3 - ax - b)$: Use $((\mathbb{Z}/p\mathbb{Z})[x]/\psi_\ell)[y]$ and reduce mod $(y^2 - x^3 - ax - b)$ by replacing y^2 by $x^3 - ax - b$.

To add points on curve over $\boldsymbol{R}$: Add points on the curve using projective coordinates over $\boldsymbol{R}$.

To compute $[n]P$ on curve over $\boldsymbol{R}$: Compute scalar multiplications using repeated point doubling and addition on the curve over $\boldsymbol{R}$.

To compute t mod ℓ where $t = p+1-\#\boldsymbol{E}(\boldsymbol{F_p})$: Exploit the equation $[t \bmod \ell][x^p : y^p : 1] = [x^{p^2} : y^{p^2} : 1] + [p][x : y : 1]$ over $\boldsymbol{R}$. First compute $[x^{p^2} : y^{p^2} : 1]$, $[x^p : y^p : 1]$, and $[p][x : y : 1]$ over $\boldsymbol{R}$, and then compute the discrete logarithm in 0,1,...,ℓ-1 of the point $[x^{p^2} : y^{p^2} : 1] + [p][x : y : 1]$ base $[x^p : y^p : 1]$.

To compute t where $t = p+1-\#\boldsymbol{E}(\boldsymbol{F_p})$: Find enough small primes ℓ to have $\prod \ell > 4\sqrt{p}$. Compute t mod ℓ for each ℓ; recover t mod $\prod \ell$ by the Chinese remainder theorem; recover t using the fact that $|t| \le 2\sqrt{p}$.

Time

The following table shows the average time in clock cycles to calculate $\#\boldsymbol{E}(\boldsymbol{F_p})$ for various sizes of p using Schoof's original algorithm on an AMD Athlon 64 X2 Dual Core Processor 3800+ and Magma version V2.12-10's SEA algorithm on an Intel Pentium 4 CPU 3400MHz.

bits in p	Original Schoof	maximum ℓ	Magma SEA
32	2000000000	17	102000000
64	50000000000	31	340000000
128	1080000000000	59	4964000000
160	2432000000000	67	14143000000
192	6082000000000	79	28288000000
224	13382000000000	89	42806000000
256	23896000000000	103	98056000000
320	71106000000000	131	217872000000
384	164712000000000	151	522376000000
448	324060000000000	173	1404370000000

Conclusion

Schoof's Algorithm in its original form is practical for use with elliptic curves of cryptographic size.