"Chinese & Match", an alternative to Atkin's "Match and Sort" method used in the SEA algorithm. Mathematics of Computation, 70(234):827–836, April 2001.

"Chinese & Match", an alternative to Atkin's "Match and Sort" method used in the SEA algorithm*

Antoine Joux[1] and Reynald Lercier[2]

[1] DCSSI, 18 rue du Dr. Zamenhoff, F-92131 Issy-les-Moulineaux, France
Antoine.Joux@m4x.org
[2] CELAR, Route de Laillé, F-35998 Rennes Armées, France
lercier@celar.fr

Abstract. A classical way to compute the number of points of elliptic curves defined over finite fields from partial data obtained in SEA (Schoof Elkies Atkin) algorithm is a so-called "Match and Sort" method due to Atkin. This method is a "baby step/giant step" way to find the number of points among C candidates with $O(\sqrt{C})$ elliptic curve additions. Observing that the partial information modulo Atkin's primes is redundant, we propose to take advantage of this redundancy to eliminate the usual elliptic curve algebra in this phase of the SEA computation. This yields an algorithm of similar complexity, but the space needed is smaller than what Atkin's method requires. In practice, our technique amounts to an acceleration of Atkin's method, allowing us to count the number of points of an elliptic curve defined over $\mathbb{F}_{2^{1663}}$. As far as we know, this is the largest point-counting computation to date. Furthermore, the algorithm is easily parallelized.

1 Introduction

As outlined by Elkies at the very beginning of [**?**], "the problem of calculating the trace of an elliptic curve over a finite field has attracted considerable interest in recent years". Thanks to the work of many people in this field of research, we now have adequate tools to perform this task.

From a theoretical point of view, Schoof's deterministic polynomial time algorithm [**?**, **?**] which enables us to perform this task in $O(\log^8 q)$ elementary operations was first largely improved by Atkin [**?**, **?**] and Elkies [**?**, **?**] to yield a probabilistic algorithm whose complexity is $O(\log^6 q)$ for finite fields $\mathbb{F}_q$ of large characteristic (using usual finite field arithmetic). In finite fields of small characteristic, we are indebted to Couveignes [**?**] for filling the remaining gaps and obtaining an algorithm of the same complexity. In this paper we will refer to these contributions as the SEA algorithm (Schoof, Elkies, Atkin).

The aim of this paper is to explain how one can speed up the last phase of the computation from a practical point of view. It is known that implementations [**?**, **?**, **?**, **?**, **?**] already were enough to compute the number of points of an elliptic curve defined over $\mathbb{F}_{10^{499}+153}$ [**?**]. But, thanks to some new ideas described in this paper, we were able to improve this point-count record by performing a computation over $\mathbb{F}_{2^{1663}}$ (cf. Section 4).

More precisely, it is well known that the number of points of an elliptic curve E defined over $\mathbb{F}_q$ is equal to $q + 1 - c$, where the trace c must satisfy $|c| \leq 2\sqrt{q}$. The aim of the SEA algorithm is to get a minimal number of candidates for c modulo coprime small integers ℓ such that

$$\prod_{\substack{\text{coprime} \\ \text{integers } \ell}} \ell > 4\sqrt{q}. \tag{1}$$

It would be nice to have a unique candidate for each ℓ, since a straightforward application of the Chinese Remainder Theorem would yield the number of points.

Unfortunately, one has a unique candidate for $c \bmod \ell$ for approximately half of the integers ℓ, in general for so called "Elkies primes" and their powers. For the other primes, called "Atkin primes", we

* January 10, 2015

have an even number of candidates. One obvious solution is to keep only Elkies primes. Even if this causes us to use a larger integer ℓ, the asymptotic complexity of the algorithm remains the same. But in practice, it is much better to decrease the size of these integers ℓ by replacing some Elkies primes by Atkin primes ℓ such that the number of candidates for $c \bmod \ell$ is as small as possible compared with the size of ℓ. Of course, we have in this case a number C of candidates for the cardinality we seek.

A naïve approach to find the number of points among these C candidates consists in multiplying a point P of the curve by each candidate until we get the neutral element of the elliptic curve. Of course, such an approach is clearly inefficient, that is why one uses instead a "baby step/giant step" algorithm due to Atkin [**?**], called "Match and Sort". We will not describe this algorithm in this paper, but one important fact is that it decreases the complexity to $O(C^{\frac{1}{2}})$ additions in $E(\mathbb{F}_q)$ (see [**?**] for some further insights); that is to say, a complexity of $O(C^{\frac{1}{2}} \log^2 q)$ elementary operations. The storage needed is $O(C^{\frac{1}{2}} \log q)$ bits.

The method we call "Chinese & Match" follows a different approach, since we get rid of elliptic curve additions. It takes advantage of the fact that we have, in records like $\mathbb{F}_{2^{1663}}$, much more information about c than what is actually used by Atkin's approach, since the number of Atkin primes used is generally small (to have a reasonable number C) compared to the number of Atkin primes available. Instead of looking for an equality of points, the basic idea behind this new approach is to eliminate bad candidates, checking that these candidates yield wrong results modulo the remaining Atkin primes (Section 2). We explain in this paper how we are able to do such a thing thanks to an algorithm with satisfactory complexities (Section 3). Then, we show how we applied this algorithm for computing the number of points of an elliptic curve defined over $\mathbb{F}_{2^{1663}}$ (Section 4).

2 "Chinese & Match" method

For the sake of clarity, we describe the algorithm at intermediate levels of detail (Section 2.1 and Section 2.2) before giving the final algorithm (Section 2.3).

Let $\mathcal{L}$ be a set of coprime integers ℓ such that $\prod_{\ell \in \mathcal{L}} \ell > 4\sqrt{q}$ and such that for each $\ell \in \mathcal{L}$, we have a set $\mathcal{T}_\ell$ of C_ℓ candidates for $c \bmod \ell$. The aim of the algorithm is to return the set of integers $c \in [-2\sqrt{q}, 2\sqrt{q}]$ which satisfy $c \bmod \ell \in \mathcal{T}_\ell$. The basic idea consists, in a first phase thanks to an algorithm due to Nicolas (given in [**?**]), in constructing a set $\mathcal{T}_\mathfrak{M}$ of C candidates for c thanks to a "minimal subset" of $\mathcal{L}$, that is to say, using moduli ℓ of a subset $\mathfrak{M}$. More precisely, the two sets $\mathfrak{M}$ and $\mathcal{T}_\mathfrak{M}$ are given by

$$\mathfrak{M} = \left\{ \ell \in \mathcal{L} \mid M_\mathfrak{M} = \prod \ell > 4\sqrt{q} \text{ and } \prod \#\mathcal{T}_\ell \text{ minimal} \right\},$$
$$\mathcal{T}_\mathfrak{M} = \left\{ c \in [-M_\mathfrak{M}/2, M_\mathfrak{M}/2] \mid \forall \ell \in \mathfrak{M}, c \bmod \ell \in \mathcal{T}_\ell \right\}.$$

Then, in a last phase, we remove candidates which do not satisfy congruences $\mathcal{T}_\ell$ given by integers ℓ of the set $\mathfrak{C} = \mathcal{L} \setminus \mathfrak{M}$.

2.1 An overview of the algorithm

To implement these ideas while avoiding an exhaustive enumeration, we first have to partition $\mathfrak{M}$ into two subsets,

$$\mathfrak{M} = \mathfrak{B} \cup \mathfrak{G}.$$

For S either $\mathfrak{B}$ or $\mathfrak{G}$, we define

$$M_S = \prod_{\ell \in S} \ell,$$
$$C_S = \prod_{\ell \in S} \#\mathcal{T}_\ell = \prod_{\ell \in S} C_\ell.$$

Until the analysis (Section 3), we do not assume anything about these subsets, but the reader should keep in mind that these subsets are chosen with $C_\mathfrak{B} \simeq C_\mathfrak{G} \simeq C^{\frac{1}{2}}$.

Then, in a precomputation step, we compute using the Chinese Remainder Theorem two sets $\mathcal{T}_{\mathfrak{B}/\mathfrak{G}}$ and $\mathcal{T}_{\mathfrak{G}/\mathfrak{B}}$ which contain what we call "partial candidates" associated to $\mathfrak{B}$ and $\mathfrak{G}$. Precisely, we have

$$\begin{aligned}\mathcal{T}_{\mathfrak{B}/\mathfrak{G}} &= \{c \in [-M_{\mathfrak{M}}/2, M_{\mathfrak{M}}/2] \mid \forall \ell \in \mathfrak{B}, c \bmod \ell \in \mathcal{T}_\ell \text{ and } \forall \ell \in \mathfrak{G}, c \bmod \ell = 0\}, \\ \mathcal{T}_{\mathfrak{G}/\mathfrak{B}} &= \{c \in [-M_{\mathfrak{M}}/2, M_{\mathfrak{M}}/2] \mid \forall \ell \in \mathfrak{B}, c \bmod \ell = 0 \text{ and } \forall \ell \in \mathfrak{G}, c \bmod \ell \in \mathcal{T}_\ell\}.\end{aligned}$$

It is not difficult to see that

$$\mathcal{T}_{\mathfrak{M}} \subseteq \left\{\alpha + \beta + \lambda M_{\mathfrak{M}} \mid \ (\alpha, \beta) \in \mathcal{T}_{\mathfrak{B}/\mathfrak{G}} \times \mathcal{T}_{\mathfrak{G}/\mathfrak{B}} \text{ and } \lambda \in \{-1, 0, 1\}\right\}.$$

So, using a "baby-giant" step way, a first version of the algorithm consists of two steps :

Baby step For $\alpha \in \mathcal{T}_{\mathfrak{B}/\mathfrak{G}}$ and for $\ell \in \mathfrak{C}$, store the set of integers

$$H_{\alpha,\ell} = \{\theta - \alpha \bmod \ell \mid \theta \in \mathcal{T}_\ell\}.$$

Giant step For $\lambda \in \{-1, 0, 1\}$ and for $\beta \in \mathcal{T}_{\mathfrak{G}/\mathfrak{B}}$, if there exists one integer $\alpha \in \mathcal{T}_{\mathfrak{B}/\mathfrak{G}}$ such that for every $\ell \in \mathfrak{C}$, the integer $\beta + \lambda M_{\mathfrak{M}} \bmod \ell$ is in $H_{\alpha,\ell}$, then $c = \alpha + \beta + \lambda M_{\mathfrak{M}}$ is one of the integers we seek—that is to say, an integer such that

$$\forall \ell \in \mathcal{L},\ c \bmod \ell \in \mathcal{T}_\ell.$$

2.2 A first variant

One can slightly improve the previous algorithm by dividing the previous baby and giant steps into numerous smaller such steps. The idea is to consider candidates $\alpha + \beta$ which already satisfy conditions given by a subset $\mathfrak{C}_1$ of $\mathfrak{C}$. Then, one partitions $\mathcal{T}_{\mathfrak{B}/\mathfrak{G}}$ and $\mathcal{T}_{\mathfrak{G}/\mathfrak{B}}$ into $M_{\mathfrak{C}_1}$ subsets,

$$\mathcal{T}_{\mathfrak{B}/\mathfrak{G}} = \bigcup_{h=0}^{M_{\mathfrak{C}_1}-1} \mathcal{T}^{(h)}_{\mathfrak{B}/\mathfrak{G}} \quad \text{and} \quad \mathcal{T}_{\mathfrak{G}/\mathfrak{B}} = \bigcup_{h=0}^{M_{\mathfrak{C}_1}-1} \mathcal{T}^{(h)}_{\mathfrak{G}/\mathfrak{B}},$$

where the subsets $\mathcal{T}^{(h)}_{\mathfrak{B}/\mathfrak{G}}$ and $\mathcal{T}^{(h)}_{\mathfrak{G}/\mathfrak{B}}$ are given by

$$\forall h \in \{0, \ldots, M_{\mathfrak{C}_1} - 1\} \begin{cases} \mathcal{T}^{(h)}_{\mathfrak{B}/\mathfrak{G}} = \{\alpha \in \ \mathcal{T}_{\mathfrak{B}/\mathfrak{G}} |\ \alpha = h \bmod M_{\mathfrak{C}_1}\}, \\ \mathcal{T}^{(h)}_{\mathfrak{G}/\mathfrak{B}} = \{\beta \in \ \mathcal{T}_{\mathfrak{G}/\mathfrak{B}} |\ \beta = h \bmod M_{\mathfrak{C}_1}\}. \end{cases}$$

Then, the previous algorithm can be rewritten as follows (where $\mathfrak{C}_2 = \mathfrak{C} \setminus \mathfrak{C}_1$ and $\mathcal{T}_{\mathfrak{C}_1}$ is defined as $\mathcal{T}_{\mathfrak{M}}$).

Loop For $h \in \{0, \ldots, M_{\mathfrak{C}_1} - 1\}$,

Small baby step For $\Theta \in \mathcal{T}_{\mathfrak{C}_1}$, for $\alpha \in \mathcal{T}^{((\Theta-h) \bmod M_{\mathfrak{C}_1})}_{\mathfrak{B}/\mathfrak{G}}$ and for $\ell \in \mathfrak{C}_2$, store the set of integers

$$H_{\Theta,\alpha,\ell} = \{\theta - \alpha \bmod \ell \mid \theta \in \mathcal{T}_\ell\}.$$

Small giant step For $\lambda \in \{-1, 0, 1\}$ and for $\beta \in \mathcal{T}^{((h-\lambda M_{\mathfrak{M}}) \bmod M_{\mathfrak{C}_1})}_{\mathfrak{G}/\mathfrak{B}}$, if there exist an integer Θ in $\mathcal{T}_{\mathfrak{C}_1}$ and an integer α in $\mathcal{T}^{((\Theta-h) \bmod M_{\mathfrak{C}_1})}_{\mathfrak{B}/\mathfrak{G}}$ such that for every integer ℓ in $\mathfrak{C}_2$ the integer $\beta + \lambda M_{\mathfrak{M}} \bmod \ell$ is in $H_{\Theta,\alpha,\ell}$, then $c = \alpha + \beta + \lambda M_{\mathfrak{M}}$ is an integer such that $\forall \ell \in \mathcal{L}$, $c \bmod \ell \in \mathcal{T}_\ell$.

2.3 The final version

One of our main considerations while designing this algorithm was to reduce the space needed. In the previous schemes, most of the space is used for $\mathcal{T}_{\mathfrak{B}}$. To reduce this, one again partitions $\mathfrak{B}$ and $\mathfrak{G}$ into two subsets,

$$\mathfrak{B} = \mathfrak{B}_1 \cup \mathfrak{B}_2 \quad \text{and} \quad \mathfrak{G} = \mathfrak{G}_1 \cup \mathfrak{G}_2.$$

Then, in a precomputation step, we compute four sets[3] $\mathcal{T}_{\mathfrak{B}_1}$, $\mathcal{T}_{\mathfrak{B}_2}$, $\mathcal{T}_{\mathfrak{G}_1}$ and $\mathcal{T}_{\mathfrak{G}_2}$ which contain partial candidates associated to $\mathfrak{B}_1$, $\mathfrak{B}_2$, $\mathfrak{G}_1$ and $\mathfrak{G}_2$. Precisely, we have

$$\mathcal{T}_{\mathfrak{B}_1} = \left\{ \begin{array}{l|r} c \in [-M_{\mathfrak{M}}/2, M_{\mathfrak{M}}/2] & \forall \ell \in \mathfrak{B}_2 \cup \mathfrak{G}_1 \cup \mathfrak{G}_2,\ c \bmod \ell = 0 \\ & \text{and } \forall \ell \in \mathfrak{B}_1,\ c \bmod \ell \in \mathcal{T}_\ell \end{array} \right\},$$
$$\mathcal{T}_{\mathfrak{B}_2} = \left\{ \begin{array}{l|r} c \in [-M_{\mathfrak{M}}/2, M_{\mathfrak{M}}/2] & \forall \ell \in \mathfrak{B}_1 \cup \mathfrak{G}_1 \cup \mathfrak{G}_2,\ c \bmod \ell = 0 \\ & \text{and } \forall \ell \in \mathfrak{B}_2,\ c \bmod \ell \in \mathcal{T}_\ell \end{array} \right\},$$
$$\mathcal{T}_{\mathfrak{G}_1} = \left\{ \begin{array}{l|r} c \in [-M_{\mathfrak{M}}/2, M_{\mathfrak{M}}/2] & \forall \ell \in \mathfrak{B}_1 \cup \mathfrak{B}_2 \cup \mathfrak{G}_2,\ c \bmod \ell = 0 \\ & \text{and } \forall \ell \in \mathfrak{G}_1,\ c \bmod \ell \in \mathcal{T}_\ell \end{array} \right\},$$
$$\mathcal{T}_{\mathfrak{G}_2} = \left\{ \begin{array}{l|r} c \in [-M_{\mathfrak{M}}/2, M_{\mathfrak{M}}/2] & \forall \ell \in \mathfrak{B}_1 \cup \mathfrak{B}_2 \cup \mathfrak{G}_1,\ c \bmod \ell = 0 \\ & \text{and } \forall \ell \in \mathfrak{G}_2,\ c \bmod \ell \in \mathcal{T}_\ell \end{array} \right\}.$$

and, once again, it is not difficult to see that

$$\mathcal{T}_{\mathfrak{M}} \subseteq \{\alpha + \beta + \gamma + \delta + \lambda M_{\mathfrak{M}} \mid (\alpha, \beta, \gamma, \delta) \in \mathcal{T}_{\mathfrak{B}_1} \times \mathcal{T}_{\mathfrak{B}_2} \times \mathcal{T}_{\mathfrak{G}_1} \times \mathcal{T}_{\mathfrak{G}_2} \text{ and } \lambda \in \{-3, -2, -1, 0, 1, 2, 3\}\}.$$

Then, we partition $\mathcal{T}_{\mathfrak{B}_2}$ and $\mathcal{T}_{\mathfrak{G}_2}$ into $M_{\mathfrak{C}_1}$ subsets,

$$\mathcal{T}_{\mathfrak{B}_2} = \bigcup_{h=0}^{M_{\mathfrak{C}_1}-1} \mathcal{T}_{\mathfrak{B}_2}^{(h)} \quad \text{and} \quad \mathcal{T}_{\mathfrak{G}_2} = \bigcup_{h=0}^{M_{\mathfrak{C}_1}-1} \mathcal{T}_{\mathfrak{G}_2}^{(h)},$$

where the subsets $\mathcal{T}_{\mathfrak{B}_2}^{(h)}$ and $\mathcal{T}_{\mathfrak{G}_2}^{(h)}$ are given by $\forall h \in \{0, \ldots, M_{\mathfrak{C}_1} - 1\}$,

$$\mathcal{T}_{\mathfrak{B}_2}^{(h)} = \{\beta \in \mathcal{T}_{\mathfrak{B}_2} \mid \beta = h \bmod M_{\mathfrak{C}_1}\},$$
$$\mathcal{T}_{\mathfrak{G}_2}^{(h)} = \{\delta \in \mathcal{T}_{\mathfrak{G}_2} \mid \delta = h \bmod M_{\mathfrak{C}_1}\}.$$

With these notations, the algorithm is as follows.

Loop For $h \in \{0, \ldots, M_{\mathfrak{C}_1} - 1\}$,

Small baby step For $\Theta \in \mathcal{T}_{\mathfrak{C}_1}$, for $\alpha \in \mathcal{T}_{\mathfrak{B}_1}$, for $\beta \in \mathcal{T}_{\mathfrak{B}_2}^{(\Theta - h - \alpha) \bmod M_{\mathfrak{C}_1}}$ and for $\ell \in \mathfrak{C}_2$, store the set of integers

$$H_{\Theta,\alpha,\beta,\ell} = \{\theta - \alpha - \beta \bmod \ell \mid \theta \in \mathcal{T}_\ell\}.$$

Small giant step For $\lambda \in \{-3, -2, -1, 0, 1, 2, 3\}$, for $\gamma \in \mathcal{T}_{\mathfrak{G}_1}$ and for $\delta \in \mathcal{T}_{\mathfrak{G}_2}^{((h - \gamma - \lambda M_{\mathfrak{M}}) \bmod M_{\mathfrak{C}_1})}$, if there exist an integer Θ in $\mathcal{T}_{\mathfrak{C}_1}$, an integer $\alpha \in \mathcal{T}_{\mathfrak{B}_1}$ and an integer β in $\mathcal{T}_{\mathfrak{B}_2}^{((\Theta - h - \alpha) \bmod M_{\mathfrak{C}_1})}$ such that for every integer ℓ in $\mathfrak{C}_2$, the integer $\gamma + \delta + \lambda M_{\mathfrak{M}} \bmod \ell$ is in $H_{\Theta,\alpha,\beta,\ell}$, then $c = \alpha + \beta + \gamma + \delta + \lambda M_{\mathfrak{M}}$ is one of the integers we seek.

2.4 Practical improvements

We describe in this section some improvements which do not change the complexity of the algorithm but which greatly improve times in practice.

[3] Following previous notations, $\mathcal{T}_{\mathfrak{B}_1}$ should be written as $\mathcal{T}_{\mathfrak{B}_1/\mathfrak{B}_2 \cup \mathfrak{G}_1 \cup \mathfrak{G}_2}$, however it would be too cumbersome. The same remark applies to $\mathcal{T}_{\mathfrak{B}_2}$, $\mathcal{T}_{\mathfrak{G}_1}$ and $\mathcal{T}_{\mathfrak{G}_2}$.

Precomputations Instead of using $\mathcal{T}_{\mathfrak{B}_1}$, $\mathcal{T}_{\mathfrak{B}_2}$, $\mathcal{T}_{\mathfrak{G}_1}$ and $\mathcal{T}_{\mathfrak{G}_2}$, one prefers to compute these sets modulo each prime of $\mathfrak{C}_2$. That is to say, we substitute in the algorithm the sets $\mathcal{T}_{\mathfrak{B}_1}$, $\mathcal{T}_{\mathfrak{B}_2}$, $\mathcal{T}_{\mathfrak{G}_1}$ and $\mathcal{T}_{\mathfrak{G}_2}$ with the sets $\mathcal{T}_{\mathfrak{B}_1,\ell}$, $\mathcal{T}_{\mathfrak{B}_2,\ell}$, $\mathcal{T}_{\mathfrak{G}_1,\ell}$ and $\mathcal{T}_{\mathfrak{G}_2,\ell}$ defined for $\ell \in \mathfrak{C}_2$ by

$$\mathcal{T}_{\mathfrak{B}_1,\ell} = \{\alpha \bmod \ell \mid \alpha \in \mathcal{T}_{\mathfrak{B}_1}\}, \ \mathcal{T}_{\mathfrak{B}_2,\ell} = \{\beta \bmod \ell \mid \beta \in \mathcal{T}_{\mathfrak{B}_2}\},$$
$$\mathcal{T}_{\mathfrak{G}_1,\ell} = \{\gamma \bmod \ell \mid \gamma \in \mathcal{T}_{\mathfrak{G}_1}\}, \ \mathcal{T}_{\mathfrak{G}_2,\ell} = \{\delta \bmod \ell \mid \delta \in \mathcal{T}_{\mathfrak{G}_2}\}.$$

Of course, we also have sets $\mathcal{T}^{(h)}_{\mathfrak{B}_2,\ell}$ and $\mathcal{T}^{(h)}_{\mathfrak{G}_2,\ell}$ instead of $\mathcal{T}^{(h)}_{\mathfrak{B}_2}$ and $\mathcal{T}^{(h)}_{\mathfrak{G}_2}$. Let us note that to construct $\mathcal{T}^{(h)}_{\mathfrak{B}_2,\ell}$ and $\mathcal{T}^{(h)}_{\mathfrak{G}_2,\ell}$, we also have to compute $\mathcal{T}_{\mathfrak{B}_2,\ell}$ and $\mathcal{T}_{\mathfrak{G}_2,\ell}$ for $\ell \in \mathfrak{C}_1$.

Small baby steps In practice, the sets $H_{\Theta,\alpha,\beta,\ell}$ can be stored very efficiently, as an array of bits. This array has $\sum_{\ell\in\mathfrak{C}_2} \ell$ lines and one column for each pair (α, β). Each line of the array can also be seen to have a bit string $S_{\ell,\tau}$ where $\ell \in \mathfrak{C}_2$ and $\tau \in \{0, \ldots, \ell - 1\}$. We store a 1 in $S_{\ell,\tau}$, for each position (α, β) such that $\tau = \theta - \alpha - \beta \bmod \ell \in H_{\Theta,\alpha,\beta,\ell}$, and a 0 otherwise. All strings $S_{\ell,\tau}$ have the same length; however, it may change from one round of the main loop to the next. On average, this length is $C_{\mathfrak{C}_1} C_{\mathfrak{B}_1} C_{\mathfrak{B}_2} / M_{\mathfrak{C}_1}$.

Small giant steps To detect if an integer $\tau = \gamma + \delta + \lambda M_{\mathfrak{M}} \bmod \ell$ is stored in $H_{\Theta,\alpha,\beta,\ell}$ for every integer $\ell \in \mathfrak{C}_2$, we simply perform a "logical &" between the $\#\mathfrak{C}_2$ strings $S_{\ell,\tau}$ (this explains the symbol & in the name of this method). If the resulting string is non-nil, we have won: any bit equal to 1 corresponds to a pair (α, β) such that $c = \alpha + \beta + \gamma + \delta + \lambda M_{\mathfrak{M}}$ is in $\mathcal{T}_\ell$ for any integer ℓ in $\mathcal{L}$.

3 Analysis of the algorithm

The aim of this section is an attempt to explain the good behavior of this algorithm that we observed in practice. That is why we perform a small analysis of the "Chinese & Match" method, both in space (Section 3.1) and in time (Section 3.2), before comparing them to Atkin's method (Sections 3.3 and 3.4).

3.1 Storage complexity

Clearly, there are two phases which need some storage, the "precomputation" and the "small baby step" phases.

Precomputations The main cost of this phase is the memory needed to store $\mathcal{T}_{\mathfrak{B}_1,\ell}$, $\mathcal{T}_{\mathfrak{B}_2,\ell}$ and $\mathcal{T}_{\mathfrak{G}_1,\ell}$, $\mathcal{T}_{\mathfrak{G}_2,\ell}$. It is at most the size of integers $\ell \in \mathfrak{C}$ times the cardinality of $\mathcal{T}_{\mathfrak{B}_1}$, $\mathcal{T}_{\mathfrak{B}_2}$, $\mathcal{T}_{\mathfrak{G}_1}$ and $\mathcal{T}_{\mathfrak{G}_2}$, that is to say,

$$\boxed{O\left((C_{\mathfrak{B}_1} + C_{\mathfrak{B}_2} + C_{\mathfrak{G}_1} + C_{\mathfrak{G}_2}) \sum_{\ell\in\mathfrak{C}} \log \ell\right)}.$$

Small baby steps Since there are $\displaystyle\sum_{\ell\in\mathfrak{C}_2} \ell$ strings of $C_{\mathfrak{C}_1} \times C_{\mathfrak{B}_1} \times \frac{C_{\mathfrak{B}_2}}{M_{\mathfrak{C}_1}}$ bits in average, the storage needed is given by

$$\boxed{O\left(C_{\mathfrak{C}_1} \frac{C_{\mathfrak{B}_1} C_{\mathfrak{B}_2}}{M_{\mathfrak{C}_1}} \sum_{\ell\in\mathfrak{C}_2} \ell\right)}.$$

3.2 Time complexity

Precomputations In this phase, we have, first of all, to compute $\mathcal{T}_{\mathfrak{B}_1,\ell}$, $\mathcal{T}_{\mathfrak{B}_2,\ell}$, $\mathcal{T}_{\mathfrak{G}_1,\ell}$ and $\mathcal{T}_{\mathfrak{G}_2,\ell}$ via the Chinese Remainder Theorem, and then to sort $\mathcal{T}_{\mathfrak{B}_2,\ell}$ and $\mathcal{T}_{\mathfrak{G}_2,\ell}$ to obtain the sets $\mathcal{T}^{(h)}_{\mathfrak{B}_2,\ell}$ and $\mathcal{T}^{(h)}_{\mathfrak{G}_2,\ell}$ for $h \in \{0, \ldots, M_{\mathfrak{C}_1} - 1\}$.

Therefore, the asymptotic complexity of this phase is given by

$$\boxed{\max\left(\begin{array}{r} O((C_{\mathfrak{B}_1} + C_{\mathfrak{B}_2} + C_{\mathfrak{G}_1} + C_{\mathfrak{G}_2}) \sum_{\ell \in \mathfrak{C}} \log^2 \ell), \\ O((C_{\mathfrak{B}_2} \log C_{\mathfrak{B}_2} + C_{\mathfrak{G}_2} \log C_{\mathfrak{G}_2}) \sum_{\ell \in \mathfrak{C}_2} \log \ell) \end{array}\right).}$$

Small baby steps Since this complexity is conditioned by the number of loops, it is equal to

$$O\left(C_{\mathfrak{C}_1} C_{\mathfrak{B}_1} \left(1 + \frac{C_{\mathfrak{B}_2}}{M_{\mathfrak{C}_1}} \sum_{\ell \in \mathfrak{C}_2} \ell\right)\right).$$

We have to perform this phase $M_{\mathfrak{C}_1}$ times; therefore the total complexity is

$$\boxed{O\left(C_{\mathfrak{C}_1} C_{\mathfrak{B}_1} \max\left(M_{\mathfrak{C}_1}, C_{\mathfrak{B}_2} \sum_{\ell \in \mathfrak{C}_2} \ell\right)\right).}$$

Small giant steps The complexity of this phase is the number of loops times the average size of a string, that is to say,

$$O\left(C_{\mathfrak{G}_1} \left(1 + \frac{C_{\mathfrak{G}_2}}{M_{\mathfrak{C}_1}} \#\mathfrak{C}_2 \times C_{\mathfrak{C}_1} \frac{C_{\mathfrak{B}_1} C_{\mathfrak{B}_2}}{M_{\mathfrak{C}_1}}\right)\right).$$

Since, again, we have to perform this phase $M_{\mathfrak{C}_1}$ times, the total complexity is equal to

$$\boxed{O\left(C_{\mathfrak{B}_1} \max\left(M_{\mathfrak{C}_1}, C_{\mathfrak{C}_1} \#\mathfrak{C}_2 \frac{C_{\mathfrak{B}_2} C_{\mathfrak{G}_1} C_{\mathfrak{G}_2}}{M_{\mathfrak{C}_1}}\right)\right).}$$

3.3 Which parameters?

For computing the number of points, it is advantageous to express complexities as functions of $\log q$ and C. With this in mind, we can now choose the values of the many parameters seen in the previous section. The rationales for these choices are explained in Section 3.4.

First of all, we select $\mathfrak{C}_1$ such that

$$M_{\mathfrak{C}_1} \simeq O(C^{\frac{3}{8}}).$$

For counting points on elliptic curves, it turns out that the corresponding constant $C_{\mathfrak{C}_1}$ is very small; let us say that in the worst cases we have $C_{\mathfrak{C}_1} \simeq O(C^{\frac{1}{8}})$.

Then we classically build sets $\mathfrak{B}_1$, $\mathfrak{B}_2$, $\mathfrak{G}_1$ and $\mathfrak{G}_2$ such that

$$C_{\mathfrak{B}_1} \simeq C_{\mathfrak{B}_2} \simeq C_{\mathfrak{G}_1} \simeq C_{\mathfrak{G}_2} \simeq O(C^{\frac{1}{4}}).$$

With these assumptions completed by the assumption that we have nearly $\log q$ integers ℓ of size nearly $\log q$, the complexities in space and time of the algorithm are given in Table 1.

Table 1. Asymptotic complexities.

	Precomputations	Baby steps	Giant steps
Space	$O\left(C^{\frac{1}{4}} \log q \log\log q\right)$	$O\left(C^{\frac{1}{4}} \log^2 q\right)$	$O(1)$
Time	$O\left(\begin{array}{c} C^{\frac{1}{4}} \log q \log\log q \times \\ \max(\log\log q, \log C) \end{array}\right)$	$O\left(C^{\frac{5}{8}} \max(C^{\frac{1}{8}}, \log^2 q)\right)$	$O(C^{\frac{3}{4}} \log q)$

3.4 Comparison with Atkin's method

First of all, the storage needed for this algorithm is much smaller than required in Atkin's algorithm, namely $O\left(C^{\frac{1}{4}} \log^2 q\right)$ instead of $O\left(C^{\frac{1}{2}} \log q\right)$.

The situation is not as clear for our time analysis. In particular, it seems at first glance that what we announced in [**?**] was, at least, enthusiastic, since the largest asymptotic cost of the "Chinese & Match" method is $O(C^{\frac{3}{4}} \log q)$ instead of $O\left(C^{\frac{1}{2}} \log^2 q\right)$ for Atkin's one. But, in order to compare these algorithms, the point is that it is necessary not only to bind $C^{\frac{1}{4}}$ and $\log q$, but also to take into account that the the constant C can be larger in our case since our memory requirements are much smaller.

Classically, a good choice for C would be to have a similar time for the SEA algorithm and this last step. For Atkin's method, this gives

$$C^{\frac{1}{2}} \log^2 q \simeq \log^6 q, \text{ i.e., } C \simeq \log^8 q.$$

The problem with such an optimum is that we simply cannot, in practice, handle the corresponding storage ($O(\log^5 q)$). So, we decreased the size of C computing c modulo larger Elkies primes ℓ until we could handle the needed storage. But such a computation is so prohibitive that one is often tempted to decrease the number of points stored in the first phase of Atkin's Method instead. Let us assume with $C \simeq \log^8 q$ that we can only accept a storage in $O(\log^4 q)$; this yields a corresponding time complexity equal to $O(\log^7 q)$.

With the "Chinese & Match" method, such an optimum yields

$$C^{\frac{3}{4}} \log q \simeq \log^6 q, \text{ i.e., } C \simeq \log^{\frac{20}{3}} q \simeq \log^{6.63} q,$$

and the corresponding memory is quite reasonable since it is only equal to

$$O(\log^{\frac{11}{3}} q) \simeq O(\log^{3.63} q).$$

Conclusion From a theoretical point of view, it appears that with Atkin's method, one must use the SEA algorithm for larger ℓ than required in our method. And, we think our approach and Atkin's approach eventually have similar asymptotic complexities in time.

From a practical point of view, we observe that this new algorithm has much better performance in practice, in particular because it substitutes elliptic curve additions with logical & in such a way that these operations are much more easily handled by computers. Moreover, it is obvious how to parallelize it among a network of workstations, and the storage needed is much smaller.

4 Application to $\mathbb{F}_{2^{1663}}$

The "Chinese & Match" method was initially designed to complete in

$$\mathbb{F}_{2^{1663}} = \mathbb{F}_2[t]/(t^{1663} + t^9 + t^8 + t^6 + t^4 + t^3 + 1),$$

the computation of the number of points $2^{1663} + 1 - c$ of the curve E given by

$$E \ : Y^2 + XY = X^3 + a_6$$

where

$$a_6 = t^{16} + t^{14} + t^{13} + t^9 + t^8 + t^7 + t^6 + t^5 + t^4 + t^3.$$

At the end of the SEA algorithm, we had, first, $C_\ell = 1$ for $\ell \in \mathfrak{U}$, where

$$\begin{aligned}\mathfrak{U} = \{&2^{12}, 3^7, 5^2, 7^2, 11^3, 13, 17^2, 19, 23^2, 29, 31, 37^2, 41, 43^2, 47^2, 53^2, 59, 67^2, 73,\\ &79,\ 89^2,\ 101, 103, 109, 127, 131, 151, 157, 163, 167, 179, 223, 227, 229, 233,\\ &251^2, 257, 263, 271, 293, 337, 347, 373, 383, 401, 431, 433, 439, 449, 461, 463,\\ &467, 491, 503, 509, 523, 547, 563, 569, 571, 577, 587, 599, 613, 619, 631, 643,\\ &647, 677, 701, 709, 719, 727, 797, 811, 853, 857, 911, 929, 937, 947, 983, 991\}.\end{aligned}$$

On the other hand, Atkin primes yield the following results.

ℓ	C_ℓ	ℓ	C_ℓ	ℓ	C_ℓ	ℓ	C_ℓ	ℓ	C_ℓ
61	30	239	8	389	48	617	102	827	264
71	12	241	110	397	198	641	212	829	328
83	24	269	72	409	4	653	108	839	48
97	42	277	138	419	24	659	160	859	336
107	4	281	46	421	210	661	330	863	36
113	18	283	140	443	72	673	336	877	438
137	44	307	20	457	228	683	216	881	252
139	12	311	24	479	64	691	344	883	384
149	40	313	156	487	60	751	92	887	144
173	56	317	52	499	8	757	378	907	452
181	72	331	164	521	56	761	252	919	22
191	32	349	120	541	270	773	84	941	312
193	96	353	58	557	60	787	392	953	312
197	10	359	4	593	180	809	216	967	220
199	20	367	88	601	252	821	272	971	324
211	104	379	8	607	72	823	204		

First of all, we chose

$$\mathfrak{M} = \{107, 113, 139, 197, 199, 307, 311, 359, 379, 409, 419, 499, 863, 919\} \cup \mathfrak{U}.$$

So, we have $C_{\mathfrak{M}} = 1.6\ 10^{15}$. Then, we fix

$$\begin{aligned}&\mathfrak{B}_1 = \{199, 307, 919\} \cup \mathfrak{U}, \qquad \mathfrak{B}_2 = \{113, 197, 419\},\\ &\mathfrak{G}_1 = \{107, 359, 379, 409, 863\}, \ \mathfrak{G}_2 = \{139, 311, 499\},\\ &\mathfrak{C}_1 = \{239, 839\}.\end{aligned}$$

Thus,

$$C_{\mathfrak{B}_1} = 8800,\ C_{\mathfrak{B}_2} = 4320,\ C_{\mathfrak{G}_1} = 18432,\ C_{\mathfrak{G}_2} = 2304,$$
$$C_{\mathfrak{C}_1} = 384 \text{ and } M_{\mathfrak{C}_1} = 200521.$$

In our particular case, we have $\forall \ell \in \mathcal{L} \setminus \mathfrak{U}$, $\forall \theta \in \mathcal{T}_\ell$, $-\theta \bmod \ell \in \mathcal{T}_\ell$. Therefore, it is enough to take $\lambda \in \{0, 1, 2, 3\}$ in the algorithm; that's why $C_{\mathfrak{G}_2} \simeq C_{\mathfrak{G}_1}/4$.

After a single night on a network of four PII 300 MHz based PC's, where the PC number i was in charge of the iterations $h \in \{50131(i-1), \ldots, 50131i - 1\}$, we were able to find

$$\begin{aligned}c = -&3805068377240724240560419948872646673663242814390586530165014569\\ &6291678140867793409648249751945783763787485565423231176330267525 5\\ &2374659917580100657091283605529024089538963443344828094088563012 5\\ &6482261901817531054013613283088840680251956201442027043 83.\end{aligned}$$

Each PC used a storage of at most 12 megabytes.

5 Conclusion

We presented an alternative to Atkin's "Match and Sort" algorithm designed for completing the computation of the number of points of an elliptic curve defined over a finite field with the SEA algorithm. It turns out that this algorithm has good complexities, in both space and time.

These complexities corroborate what we initially observed in practice. We were able to compute the number of points of an elliptic curve defined over $\mathbb{F}_{2^{1663}}$ in a single night (real time) on a network of four PII 300 MHz based PC's using only 12 megabytes of memory.

Moreover, since the goal of this algorithm is to extract one integer from sets of congruences modulo integers, we have the feeling that applications in other fields of research could take advantage of such an algorithm.