SCIENTIFIC COMPUTING PROJECT

SCHOOF'S ALGORITHM FOR COUNTING POINTS ON AN ELLIPTIC CURVE

REPORT OF THE MAGMA IMPLEMENTATION

Paolo Piasenti

Brixen, January 2020

ABSTRACT. In this report I'm going to describe and comment the MAGMA code which has been written by Federico Mazzone, Giovanni Tognolini and me in order to implement our version of the Schoof's algorithm to count points on an elliptic curve over a finite field. The dissertation is divided into four parts: in the first, I consistently explain the theoretical environment with all the definitions and technical results we heavily exploit; in the second, I describe the two most important backgrounds of this computational problem and I underline the importance of the Schoof's machinery; the third part is fully dedicated to the algorithm itself and to an analysis concerning our implementation choices (e.g. choice of the programming language and MAGMA primitives) and empirical results in order to carry out an efficiency comparison between some well known algorithms solving the elliptic curve's cardinality issue. A fourth part regarding a concrete cryptographic application follows.

Contents

1 Preamble and preliminaries

This section is itself divided into some parts. At first, I explain the importance – for the cryptographic world – of solving the problem with counting points on an elliptic curve, which is the only reason of existence of the algorithm we implemented. Then, a consistent survey on the theory of the elliptic curves is exhibited, in order to provide the settings in which Schoof's algorithm comes to life.

1.1 Why counting points?

Philosophically, the nature of the human being has always led him to count things around him, to quantify everything engaging him. For the same reason, given a finite field $\mathbb{F}_q$ and an elliptic curve E, one can be interested in knowing how many points with coordinates in $\mathbb{F}_q$ satisfy the equation describing E.

Actually, there is another – deeper and more concrete – reason for which it makes sense to count points on an elliptic curve, and it is related to cryptography. Nowadays, many of the security protocols that keep certain information secret are based on certain kinds of operations, sometimes called *trapdoors*, which are easy to compute in one way but which are very difficult (where "very difficult" stands for "computationally impossible") to invert. Examples of this are the easy task to multiply two natural numbers and its counterpart, the unfeasible task to factorize in prime factors a given (huge) natural number, or the so called *Discrete Logarithm Problem* (DLP), which is the one we are interested in. Recalling its formulation, let's suppose to have a group G and $a, b \in G$ such that

$$a^k = b$$

with the equation holding in G. The issue is finding k. No efficient methods for quickly computing discrete logarithms are this far known. The group G is often taken to be the multiplicative group $\mathbb{F}_q^\times$ of a finite field $\mathbb{F}_q$ (with q power of a prime) or the group given by the set of points of an elliptic curve defined over

a finite field with the known addition operation. If we consider as group G the one given by the latter, we say we are dealing with the *Elliptic Curve Discrete Logarithm Problem* (ECDLP).

The ECDLP is the cornerstone of one of the most contemporary – other than most effective and most efficient – public key cryptosystems available at these times. In fact, it is sufficient to look at the following table to understand the cost-effectiveness of ECC (*Elliptic Curve Cryptography*) compared to RSA:

ECC KEY SIZE (BITS)	RSA KEY SIZE (BITS)	KEY SIZE RATIO
163	1024	1:6
256	3072	1:12
384	7680	1:20
512	15360	1:30

How one can easily understand, ECC means a consistent storage saving. Furthermore, ECC offers the pretty nice possibility to construct a huge amount of different abelian groups starting from the same elliptic curve and the same finite field by simply varying the coefficients of the equation of the curve.

Despite this, efficiency should always be assisted by efficacy, in the sense that one has to be sure that his machinery does indeed what one expects it to do, and for a cryptographic protocol this translates into robustness of the system. The strength of a system relies on its capacity to be immune to all possible attacks; with respect to ECC, there are many conditions that should be respected in order to "guarantee" the security of a specific elliptic curve, and how one can imagine all of them have to deal with the cardinality of the elliptic curve, which is a sort of index of difficulty of the ECDLP: the larger, the safer. But there are also some other more specific conditions that one should check, however these arguments are going to be discussed in the fourth section.

1.2 Theory of Elliptic Curves

Elliptic curves are one of the most important and intriguing topics of Algebraic Geometry. Recently, over the last forty years, many researchers have been conducting studies about elliptic curves for both an applicative motivation leading to new public key cryptosystems and a pure theoretical interest with all its curious links with other math branches and its considerable repercussions on the horizons of mathematics (e.g. their use in Wiles's work on Fermat's Last Theorem).

This subsection is dedicated to the main concepts about elliptic curves. Our work only focuses on elliptic curves defined over a finite field whose characteristic is different from 2 and from 3, since more efficient methods based on

curves' classification after adequate transformations have already been studied for curves defined on fields with those characteristics. A group structure is then established and all the theorems and related results are consequently provided.

1.2.1 Weierstraß Equation

As we have already said, our interest focuses on curves defined over a finite field $\mathbb{F}_q$ satisfying $char(\mathbb{F}_q) \notin \{2, 3\}$. An elliptic curve is a smooth, projective, algebraic curve of genus one and under these assumptions we can always represent it as the plane algebraic curve defined by an equation of the form

$$y^2 = x^3 + Ax + B$$

in conjunction with a specified point O on it (which is nearly always chosen to be the *point at infinity* in the projective plain) and with $A, B \in \mathbb{F}_q$ such that the discriminant of the curve

$$\Delta := -16(4A^3 + 27B^2)$$

is different from zero, since we are only interested in *non-singular* curves. The equation describing the curve of the form as above is referred to as the *Weierstraß Equation* of an elliptic curve E over the finite field $\mathbb{F}_q$ (which will be concisely denoted with $E/\mathbb{F}_q$).

Given an extension field $\mathbb{F}/\mathbb{F}_q$ (which may even be $\mathbb{F}_q$ itself), what we are interested in is the set

$$E(\mathbb{F}) := \{(x, y) \in \mathbb{F} \times \mathbb{F} \mid y^2 = x^3 + Ax + B\} \cup \{O\}$$

or, more particularly, its cardinality. A stupid and naive way to accomplish this task is to try out which of all the possible *finite* couples $(x, y) \in \mathbb{F} \times \mathbb{F}$ satisfy the relation $y^2 = x^3 + Ax + B$, adding then 1 which counts for the O point. For example, given the elliptic curve $E/\mathbb{F}_7$ defined by $y^2 = x^3 + x + 1$ we have that

$$E(\mathbb{F}_7) := \{O, (2, 2), (0, 6), (1, 0), (2, 5)\}$$

by a direct calculus exhausting all the possibilities.

1.2.2 $E(\mathbb{F})$ as a group

It is possible to define on $E(\mathbb{F})$ a *sum operation* between points which one can show to be a group law for $E(\mathbb{F})$. Actually, let $E/\mathbb{F}$ be an elliptic curve of equation $y^2 = x^3 + Ax + B$ and let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points such that $P_1, P_2 \in E$ and $P_1 \neq O \neq P_2$. We define $P_1 + P_2 := P_3 = (x_3, y_3)$ as follows:

- $\diamond$ if $x_1 \neq x_2$ then

$$(x_3, y_3) := \left(\left(\frac{y_2 - y_1}{x_2 - x_1} \right)^2 - x_2 - x_1, \left(\frac{y_2 - y_1}{x_2 - x_1} \right) (x_1 - x_3) - y_1 \right)$$

- ◇ if $x_1 = x_2$ but $y_1 \neq y_2$ then $P_1 + P_2 = O$
- ◇ if $P_1 = P_2$ but $y_1 \neq 0$ then

$$(x_3, y_3) := \left(\left(\frac{3x_1^2 + A}{2y_1} \right)^2 - 2x_1, \left(\frac{3x_1^2 + A}{2y_1} \right) (x_1 - x_3) - y_1 \right)$$

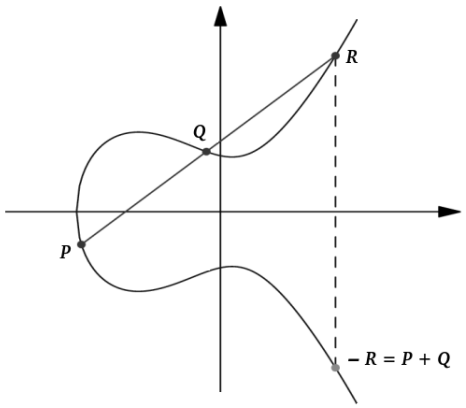
- ◇ if $P_1 = P_2$ and $y_1 = 0$ then $P_1 + P_2 = O$

Finally, in order to contemplate all possible cases, we define

$$P + O = O + P = O$$

$\forall P \in E$.

Within this framework, it is possible to show that $(E(\mathbb{F}), +, O)$ is an abelian group. Note that its trivial to see that neutral element O and inverse element always exist, also commutativity is obvious; what is tricky to see is associativity. The group structure is one of the most important – albeit simple and premature – results for what concerns elliptic curves, because it allows to make inference about them using the huge power of the Group Theory. There is an interesting geometric interpretation of the just defined sum operation, which is more evident if we consider an elliptic curve defined on an infinite field, e.g. the real field. Let's consider a generic elliptic curve of equation $y^2 = x^3 + Ax + B$ over $\mathbb{R}$, which will almost always look more or less like this:



If one considers two point on the curve, let's say P and Q, one can also track the line through the two, which will *always* intersect the curve in another point (including the point at infinity) as ensured by some arguments of Geometry 1 (see *Geometria 1, E. Sernesi, 1989, chap. 4*). The sum $P + Q$ is hence defined to be the symmetrical of the previously found point with respect to the x-axis, which always belongs to the elliptic curve because of its defining equation.

1.2.3 Hasse's Bound

There is a famous result about the cardinality of $E(\mathbb{F})$ and it is due to Helmut Hasse, who proved its theorem in 1933. It provides a quite rough estimate from both above and below of the number of points on an elliptic curve over a finite field. In particular, given an elliptic curve $E/\mathbb{F}_q$ defined by $y^2 = x^3 + Ax + B$, Hasse's Theorem states that $|E(\mathbb{F}_q)|$ is such that

$$|q + 1 - |E(\mathbb{F}_q)|| \leq 2\sqrt{q}$$

which explicitly means that

$$q + 1 - 2\sqrt{q} \leq |E(\mathbb{F}_q)| \leq q + 1 + 2\sqrt{q}$$

with $|E(\mathbb{F}_q)|$ obviously varying *discretely* between those bounds. Now, if we define $t := q + 1 - |E(\mathbb{F}_q)|$, Hasse's Theorem is equivalent to

$$-2\sqrt{q} \leq t \leq 2\sqrt{q}$$

and the cardinality of $E(\mathbb{F}_q)$ is $|E(\mathbb{F}_q)| = q + 1 - t$.

It is now clear that to determine $|E(\mathbb{F}_q)|$ it is sufficient to know t, and to know t it is sufficient to know t modulo N, where N is such that $N > 4\sqrt{q}$, so that an unambiguous determination is achievable (concretely: $N \geq \lfloor 4\sqrt{q} \rfloor + 1$). This is exactly what we are going to exploit with the Schoof's algorithm.

1.2.4 Division Polynomials and l-torsion subgroups

A very clever way to compute the multiples of a given point $P \in E(\mathbb{F}_q)$ on a generic elliptic curve E is to use some particular bivariate polynomials of $\mathbb{F}_q[x, y]$, which are called *division polynomials* of E and which only depend on the parameters A and B identifying the curve. Their definition is recursive, as follows:

$$\begin{aligned}
&\psi_0(x, y) = 0 \\
&\psi_1(x, y) = 1 \\
&\psi_2(x, y) = 2y \\
&\psi_3(x, y) = 3x^4 + 6Ax^2 + 12Bx - A^2 \\
&\psi_4(x, y) = 4y(x^6 + 5Ax^4 + 20Bx^3 - 5A^2x^2 - 4ABx - 8B^2 - A^3) \\
&\vdots \\
&\psi_{2m}(x, y) = (2y)^{-1}\psi_m(x, y)\left(\psi_{m+2}(x, y)\psi_{m-1}^2(x, y) - \psi_{m-2}(x, y)\psi_{m+1}^2(x, y)\right) \\
&\psi_{2m+1}(x, y) = \psi_{m+2}(x, y)\psi_m^3(x, y) - \psi_{m-1}(x, y)\psi_{m+1}^3(x, y)
\end{aligned}$$

A very interesting result shows how strictly these polynomials and the torsion subgroups of E are connected. Recalling this last definition, we define the *l-torsion subgroups* as the following subgroup of E (or, appropriately generalizing, of any group):

$$E[l] := \{P \in E : lP = O\}$$

One can prove that ψ_{2n+1} (hence all the division polynomials with odd index), though belonging *a priori* to $\mathbb{Z}[x,y]$, actually belongs to $\mathbb{Z}[x]$ if we perform the substitution $y^2 = x^3 + Ax + B$ (more precisely, if we consider the projection of ψ_{2n+1} onto the quotient ring $\mathbb{Z}[x,y]/(y^2-x^3-Ax-B)$, where (y^2-x^3-Ax-B) is the ideal generated by the polynomial in brackets) and consequently one can also prove that the roots of ψ_{2n+1} are the x coordinates of the points of $E[2n+1] \setminus \{O\}$. In a similar way, one can prove that ψ_{2n} (hence all the division polynomials with even index) actually belongs to $2y\mathbb{Z}[x]$ (where this notation means the subset of $\mathbb{Z}[x,y]$ containing polynomials of the form $2y \cdot p(x)$ with $p(x) \in \mathbb{Z}[x]$) and consequently one can also prove that the roots of $\psi_{2n}/(2y)$ are the x coordinates of the points of $E[2n] \setminus E[2]$.

All these premises lead to a crucial application assisting our computing purposes, which is indispensable for the architecture which will follow. It can be proved that, given a point $P = (x,y) \in E$ and $t \in \mathbb{N}$, the following formula holds:

$$tP = t(x,y) = \left(x - \frac{\psi_{t-1}(x,y)\psi_{t+1}(x,y)}{2\psi_t^2(x,y)}, \frac{\psi_{2t}(x,y)}{2\psi_t^4(x,y)} \right)$$

1.2.5 Frobenius Endomorphism and Characteristic Equation

If we consider an elliptic curve $E/\mathbb{F}_q$, we can consider $E(\overline{\mathbb{F}_q})$ where $\overline{\mathbb{F}_q}$ is the algebraic closure of $\mathbb{F}_q$. What we have obtained is the extension of the curve allowing points with coordinates in $\overline{\mathbb{F}_q}$. The Frobenius Endomorphism is the map

$$\phi : E(\overline{\mathbb{F}_q}) \longrightarrow E(\overline{\mathbb{F}_q})$$

which acts like

$$\begin{cases} (x,y) \longmapsto (x^q, y^q) \\ O \longmapsto O \end{cases}$$

Its name is not accidental, in the sense that one can easily prove that it is indeed a group morphism. It is then trivial to show that it preserves the order of points, in the sense that $|P| = |\phi(P)|$. Since $x^q = x \ \forall x \in \mathbb{F}_q$, the Frobenius Endomorphism is the identity if restricted to $E(\mathbb{F}_q)$.

A theorem of utmost importance says that the Frobenius Endomorphism satisfies a polynomial relation called Characteristic Equation:

Theorem 1. *Let E be an elliptic curve defined over $\mathbb{F}_q$. Let $t = q+1-|E(\mathbb{F}_q)|$. Then*

$$\phi^2 - t\phi + q = 0$$

and t is the unique integer which satisfies a relation of this kind.

The relevance of this result is evident, being a bridge linking the Frobenius Endomorphism to the cardiniality of $E(\mathbb{F}_q)$. Notice that it is a relation between *operators*, meaning that the map on the left of the equal sign is, in every point,

identical to the map on the right (where the "0" map is the one which sends everything in the neutral element). If we rewrite it for a generic point $P = (x, y) \in E(\overline{\mathbb{F}_q})$ it becomes

$$(\phi^2 - t\phi + q)(x, y) = 0(x, y)$$

$$\phi^2(x, y) - t\phi(x, y) + q(x, y) = O$$

$$(x^{q^2}, y^{q^2}) + q(x, y) = t(x^q, y^q)$$

where $+$ denotes addition on the elliptic curve and t and q are scalar coefficients involved in scalar multiplication on the curve.

2 Two classic early algorithms

Two are the most important attempts which have tried to solve the counting problem before Schoof's idea. One is the trivial one, the other is the so called "Baby Step - Giant Step" approach.

2.1 Naive Algorithm

A very basic algorithm is trivially based on the definition of elliptic curve itself. Since an elliptic curve over a finite field is a set (group) of pairs $(x, y) \in \mathbb{F}_q \times \mathbb{F}_q$ that satisfy $y^2 = x^2 + Ax + B$ and since there is a finite number of such pairs (all possible pairs $(x, y) \in \mathbb{F}_q \times \mathbb{F}_q$ are $|\mathbb{F}_q \times \mathbb{F}_q| = q^2$), one could eventually try all possibilities considering that a pair is a valid point of the curve if and only if it satisfies its Weierstraß Equation. At the end of the procedure, one counts all the "good" points which were found and adds 1 for the point at infinity. Clearly, this procedure has computational cost $O(q^2)$. Some tricks can be done in order to improve this approach; these expedients lead to what is effectively known as the *Naive* Algorithm.

For each $x \in \mathbb{F}_q$ one can follow the following "switch" scheme:

- $\diamond$ if $x^3 + Ax + B$ is a nonzero square in $\mathbb{F}_q$ (also known as a *quadratic residue* modulo q) then, denoting with y one of its roots, we get that (x, y) and $(x, -y)$ are two points in $E(\mathbb{F}_q)$;
- $\diamond$ if $x^3 + Ax + B = 0$ then it is a trivial quadratic residue modulo q with 0 as unique root, and hence $(x, 0)$ is a point in $E(\mathbb{F}_q)$;
- $\diamond$ if $x^3 + Ax + B$ is not a quadratic residue modulo q then it is clear that the equality given by the Weierstraß Equation cannot be satisfied by any $y \in \mathbb{F}_q$ and hence, in correspondence of this case, there is no point having this particular x coordinate which also belongs to the curve.

This algorithm has computational cost $O(q)$ (therefore it is better than the basic one) since all $x \in \mathbb{F}_q$ have to be plumbed. Notice that computing if a certain value is quadratic residue of $\mathbb{F}_q$ or not is not an issue, because squares in a finite field can be precomputed and stored in a sorted (for convenience) lookup table. Our MAGMA implementation of the Naive Algorithm can be found at `Naive.txt` among the attached files. An example showing how the algorithm works follows.

Example 1 (Naive)**.** *Let E be the curve $y^2 = x^3 + x + 1$ over $\mathbb{F}_5$. To count points on E, we make a list of the possible values of x, then of the quadratic residues of x* mod 5 *(for lookup purpose only), then of $x^3 + x + 1$* mod 5 *and then of the valid y, roots of x^3+x+1 (when they exist). The last column collects all the compatible matches, i.e. all the points of E except the point at infinity.*

x	x^2 (QUAD. RES.)	$x^3 + x + 1$	y	POINTS OF E
0	0	1	1; 4	$(0, 1)$; $(0, 4)$
1	1	3	–	–
2	4	1	1; 4	$(2, 1)$; $(2, 4)$
3	4	1	1; 4	$(3, 1)$; $(3, 4)$
4	1	4	2; 3	$(4, 2)$; $(4, 3)$

Observe that 3 is not a quadratic residue modulo 5, hence no point with $x = 1$ as abscissa can lie on the curve. The curve has cardinality $|E(\mathbb{F}_5)| = 8 + 1 = 9$, taking into account also the O point.

2.2 Baby Step - Giant Step

This method is based on the following observation: Hasse's Theorem ensure us that $|E(\mathbb{F}_q)| \in \left[q + 1 - 2\sqrt{q},\, q + 1 + 2\sqrt{q}\right]$, but from Lagrange's Theorem we also know that the order of $E(\mathbb{F}_q)$ seen as a group must live in that interval. Hence, remembering the group property stating that for every point in $E(\mathbb{F}_q)$ it must hold that

$$|E(\mathbb{F}_q)| \cdot P = O$$

one could take a generic non-neutral point $P \in E(\mathbb{F}_q)$ (by simply computing $x^3 + Ax + B$ for x varying in $\mathbb{F}_q$ until it is a square, taking y as one of its roots and considering $P = (x, y)$) and compute mP for every integer value $m \in \left[q + 1 - 2\sqrt{q},\, q + 1 + 2\sqrt{q}\right]$. If there exists a *unique* m for which $mP = O$, for what we have just said above, it definitely has to be the cardinality of $E(\mathbb{F}_q)$. This procedure has computational cost $O(\sqrt{q})$, since all values in the interval have to be plumbed and its length is $4\sqrt{q}$. Despite this, some tricks can be exploited in order to reduce the cost to $O(\sqrt[4]{q})$. Indeed, let's look at the following algorithm:

1. compute $Q = (q + 1)P$;

2. choose an integer $m > \sqrt[4]{q}$ (for convenience one can choose $m = \lceil \sqrt[4]{q} \rceil$). Compute and store the points jP for $j = 1, \ldots, m$;

3. compute the points $Q + k(2mP)$ for $k = -m, -(m-1), \ldots, m$ until there is a match $Q + k(2mP) = \pm jP$ with a point among the ones stored at the previous step;

4. conclude that $(q + 1 + 2mk \mp j)P = O$. Let $M = q + 1 + 2mk \mp j$;

5. factor M. Let $p_1, \ldots, p_r$ be the distinct prime factors of M;

6. compute $(M/p_i)P$ for $i = 1, \ldots, r$. If $(M/p_i)P = O$ for some i, replace M with M/p_i and go back to step (5). If $(M/p_i)P \neq O \ \forall i$ then M is the order of the point P;

7. repeat steps (1) - (6) with randomly chosen points in $E(\mathbb{F}_q)$ until the least common multiple of the orders divides only one integer N within $\left[q + 1 - 2\sqrt{q},\ q + 1 + 2\sqrt{q}\right]$. Then $N = |E(\mathbb{F}_q)|$.

Some considerations and explanations are due. First of all, who ensure us that there is a match like the one claimed at step (3)? For this purpose, let's introduce a theoretical result.

Proposition 1. *Let a, m be integers with $|a| \leq 2m^2$. Then there exist integers a_0, a_1 such that $a = a_0 + 2ma_1$ with $-m < a_0 \leq m$ and $-m \leq a_1 \leq m$.*

An immediate consequence of this proposition it that a match always exists. In fact: as an easy effect of the Hasse's Bound, set $N = |E(\mathbb{F}_q)|$, we know that

$$|q + 1 - N| \leq 2\sqrt{q} = 2(\sqrt[4]{q})^2 \leq 2(\lceil \sqrt[4]{q} \rceil)^2 = 2m^2$$

Hence, named $a = q + 1 - N$, we have $|a| \leq 2m^2$ and we are in the hypothesis of the previous proposition. So there exist a_0, a_1 with $-m < a_0 \leq m$ and $-m \leq a_1 \leq m$ such that $a = a_0 + 2ma_1$. But then $Q + k(2mP)$, for $k = -a_1$, is equal to

$$\begin{aligned} Q + k(2mP) &= (q+1)P + k(2mP) = (q + 1 - 2ma_1)P = \\ &= [q + 1 - (a - a_0)]P = (q + 1 - a)P + a_0P = \\ &= NP + a_0P = O + a_0P = a_0P \end{aligned}$$

and the match is thus found because a_0 is one of the valid j and $-a_1$ is one of the valid k which sooner or later will be browsed in our attempts (and this derives from the constraints on a_0 and a_1).

The second justification concerns step (6). Why should that procedure yield the order of P? Here it is another proposition:

Proposition 2. *Let G be an additive group (with neutral element 0) and let $g \in G$. Suppose $Mg = 0$ for some positive integer M. Let $p_1, \ldots, p_r$ be all the primes appearing in the factorization of M. If $(M/p_i)g = 0$ for all i, then M is the order of g.*

The two proofs are simple and are omitted. For completeness, see [1].

The effectively working algorithm we have just described is known as the *Baby Step - Giant Step* algorithm and it takes its name from the fact that there are two computational phases: the "Baby Step" is the one in which we compute the points jP for $j = 1, \ldots, m$ (since we take a "small" step of $1P$), instead the "Giant Step" is the one in which we compute $Q + k(2mP)$ (since we take a "bigger" step of $2mP$ for $k = -m, -(m-1), \ldots, m$). Our implementation can be find at `BGStep.txt` among the attached files.

It is now easy to understand why its cost is $O(\sqrt[4]{q})$. Indeed, the "expensive" operations are those which are aimed at computing points, and for a generic non-neutral point P on E we have a calculation in correspondence of every $j \in \{1, \ldots, m\}$ (hence $\lceil \sqrt[4]{q} \rceil$ calculations) and then, at worst, also in correspondence of every $k \in \{-m, -(m-1), \ldots, 0, 1, \ldots, m\}$ (hence about $2\lceil \sqrt[4]{q} \rceil$ calculations)), for a total of approximately $3\sqrt[4]{q}$ steps. This procedure might require to be applied more than once but this fact does not affect the complexity since we are only contemplating some constant additional steps (turning the cost into $3k\sqrt[4]{q}$). Some other tricks can be exploited in order to make the algorithm work faster (e.g. quick point addition by storing Q and $2mP$ once and for all and smart pairing) but this is not our intent (for completeness, see again [1]). Also in this case, an example is shown.

Example 2 (Baby Step - Giant Step)**.** *Let E be the elliptic curve defined by $y^2 = x^3 - 10x + 21$ over $\mathbb{F}_{557}$. and let $P = (2, 3) \in E$. We follow the procedure above:*

1. *we compute $Q = (557 + 1)P = 558P = (418, 33)$;*
2. *let $m = \lceil \sqrt[4]{557} \rceil = 5$. The sequence of jP for $j \in \{1, \ldots, 5\}$ is*

$$(2, 3), (58, 164), (44, 294), (56, 339), (132, 364);$$

3. *we compute $Q + k(2mP)$ and we see that for $k = 1$ we have the match*

$$Q + 2mP = (2, 3)$$

   and this last point is exactly the first of our previous sequence (for $j = 1$);

4. *let $M := q + 1 + 2mk - j = 557 + 1 + 2 \cdot 5 \cdot 1 - 1 = 567$ and we take $MP = 567P = O$;*
5. *we factor $567 = 3^4 \cdot 7$;*

6. *we compute* $(567/3)P = 189P$ *and we discover that* $189P = O$*, hence we redefine* $M := 189 =$ *whose factorization is clearly equal to* $3^3 \cdot 7$ *and we similarly compute* $(189/3)P = 63P = (38, 535) \neq O$ *and* $(189/7)P = 24P = (136, 360) \neq O$*. Therefore* P *has order* 189*;*

7. *Hasse's Theorem implies that* $511 \leq |E(\mathbb{F}_q)| \leq 605$*. The only multiple of* 189 *in this range is* $3 \cdot 189 = 567$*. Therefore* $|E(\mathbb{F}_q)| = 567$*.*

Notice that the algorithm would have failed if there were more than one multiple of 189 *in* $[511, 605]$*. In that case it would be sufficient to try to perform the algorithm with another random point* P *and use the lcm way. But what if for every point* P *on the curve there are two values in* $(q+1-2\sqrt{q},\ q+1+2\sqrt{q})$ *that annihilates* P*? This case happens almost never, but when it does, the cardinality of the curve is very very small. Thus we are not dealing with an issue.*

As one can ascertain, the two algorithms presented in this section are very tedious and "likely-to-fail" when q is a huge prime, as well as quite inefficient. In the next section we are going to describe the best deterministic algorithm so far for counting point on an elliptic curve. With respect to the dimension of the input such as $\log q$, Schoof's algorithm has been the first polynomial algorithm accomplishing the "counting" task; instead the two presented above have, from this perspective, both an exponential running time. These considerations are very important for our cryptographic purposes (as already mentioned in the first section), and Schoof's breakthrough offers a very powerful tool never seen before.

3 Schoof's Algoritm

In 1985 Renè Schoof published an article containing his algorithm to solve the cardinality problem. As we are going to see, it has computational cost $O(\log^8 q)$ and it is *asymptotically fast*, in the sense that it performs optimally for $q \to +\infty$ and for such large q no other algorithm can remotely compete with it. Nevertheless, the fact that Schoof's algorithm is powerful at the asymptote means that for an initial (and also quite large) range of values of q it performs worse that the two other algorithms we have seen before (as we are going to point out in the "test and comparisons" section).

3.1 The algorithm

Schoof's algorithm makes use of some very interesting theoretical results like the Chinese Reminder Theorem alongside the now standard Hasse's Bound and division polynomials.

Let E be an elliptic curve over a finite field $\mathbb{F}_q$. A reformulation of Hasse's Bound, as we have seen in the first chapter, states that $|E(\mathbb{F}_q)| = q + 1 - t$ where t is an integer such that $t \leq |2\sqrt{q}|$. Recalling that for every point

$P = (x, y) \in E(\overline{\mathbb{F}_q})$ holds that $(x^{q^2}, y^{q^2}) + q(x, y) = t(x^q, y^q)$, one might be tempted to symbolically compute every single point appearing in the previous formula as a couple of functions in the quotient ring $\mathbb{F}_q[x, y]/(y^2 - x^3 - Ax - B)$ and try to express t in terms of them so as to obtain the appropriate t satisfying the Characteristic Equation. Despite this, this way is unfeasible because of the too large degrees of the polynomials representing the coordinates. But there is a clever way to proceed.

As we have already pointed out in the first section (in particular in the "Hasse's Bound" subsection), it is sufficient to determine $t \bmod N$ where $N > 4\sqrt{q}$ to univocally determine t. To achieve this goal the idea is to ulteriorly simplify the problem and determine $t \bmod l$ for l a small prime, repeating this operation for many primes. The precise procedure goes as follows. Let $S = \{l_1, l_2, \ldots, l_r\}$ such that

$$\prod_{i=1}^{r} l_i =: N > 4\sqrt{q}$$

and we want compute $t \bmod l_i$ for every $i \in \{1, \ldots, r\}$. Clearly, if one knew all these congruences, he could rapidly get $t \bmod N = \prod_{i=1}^{r} l_i$ (and hence t) by solving the system given by the mod-equations through the Chinese Reminder Theorem. Let's then see how a congruence like the ones above can be obtained.

We reduce ourselves to the case in which our small prime satisfies $l \neq p$, where p is the characteristic of the field $\mathbb{F}_q$, and also $l \neq 2$, because this last case is very simple and will be treated separately. The idea is to do what we were trying to do previously (i.e. solving the Characteristic Equation for t) but restricting the computation to the l-torsion subgroup $E[l]$. Given a point $P = (x, y) \in E[l]$, by definition we have that $lP = O$. Hence $qP = \bar{q}P$, where $\bar{q}$ is the unique integer such that $q \equiv \bar{q} \pmod{l}$ and $|\bar{q}| < \frac{l}{2}$. Moreover, since $(x^q, y^q) = \phi(P)$ is exactly the image of P under the Frobenius Endomorphism, it has its same order (as we already pointed out some pages ago); hence it follows that $l(x^q, y^q) = O$ and thus $t(x^q, y^q) = t_l(x^q, y^q)$ with $t_l := t \bmod l$. For convenience we define, alike ahead, $\bar{t}$ as the unique integer that satisfies $t \equiv t_l \equiv \bar{t} \pmod{l}$ and $|\bar{t}| < \frac{l}{2}$. What we finally get is that it holds

$$(x^{q^2}, y^{q^2}) + \bar{q}(x, y) = \bar{t}(x^q, y^q)$$

for every point $P \in E[l]$ and where $\bar{t}$ and $\bar{q}$ can only assume the integer values in $\left\{-\frac{l-1}{2}, \ldots, 0, \ldots, \frac{l-1}{2}\right\}$. It is easy to agree that the problem has been radically smoothed. At this point one can compute the leftwing member of this equation (for which there is nothing unknown) and search for the "good" $\bar{t}$ among $\left\{-\frac{l-1}{2}, \ldots, 0, \ldots, \frac{l-1}{2}\right\}$ which makes the equation true. As for its existence, we are sure that such a $\bar{t}$ exists trivially because t exists, and we are sure such a $\bar{t}$ is unique because of the uniqueness of t given by *Theorem 1* in the first section.

Observe that the restriction of the computation of the sum $(x^q, y^q) + \bar{q}(x, y)$ to the l-torsion points requires to compute these expressions as functions in the polynomial ring

$$\mathbb{F}_q[x, y]/(y^2 - x^3 - Ax - B, \psi_l)$$

which is the coordinate quotient ring of E modulo the l-th division polynomial. The reason behind this is that we are looking for points on E (hence points whose coordinates live in $\mathbb{F}_q[x, y]/(y^2 - x^3 - Ax - B)$) which additionally belong to $E[l]$, hence such that their x coordinate is a root of ψ_l (recalling the first section), hence such that their x coordinate annihilates ψ_l. As a matter of fact, all these conditions are actually equivalent, i.e.

$$P = (x, y) \in E[l] \iff \psi_l(x, y) = \psi_l(x) = 0$$

and this justifies the quotient over ψ_l and consequent the computation modulo ψ_l. This means, in particular, that the degree of X and Y defined via

$$(X(x, y), Y(x, y)) := (x^{q^2}, y^{q^2}) + \bar{q}(x, y)$$

and representing the letfwing member of the re-written Characteristic Equation is at most 1 in y (because we transform every y^2 in $x^3 + Ax + B$) and at most $(l^2 - 3)/2$ in x (because we can divide by ψ_l every expression in the single x variable and since $\deg(\psi_l) = \frac{1}{2}(l^2 - 1)$ we always obtain a polynomial whose degree is $\frac{1}{2}(l^2 - 1) - 1 = \frac{1}{2}(l^2 - 3)$ or less).

For each l, we split the problem into two cases: the first one, in which $(x^{q^2}, y^{q^2}) \neq \pm\bar{q}(x, y)$ and the second one, in which $(x^{q^2}, y^{q^2}) = \pm\bar{q}(x, y)$. Clearly, for every l, we keep track of t_l so that at the end the Chinese Reminder Theorem machinery can be triggered.

- ◇ **1st case:** $(x^{q^2}, y^{q^2}) \neq \pm\bar{q}(x, y)$

  Let's set $\overline{q}(x, y) = (x_{\overline{q}}, y_{\overline{q}})$. By the addition formula on the group $E(\overline{\mathbb{F}}_q)$ (see first section) we obtain:

$$X(x, y) = \left(\frac{y^{q^2} - y_{\bar{q}}}{x^{q^2} - x_{\bar{q}}}\right)^2 - x^{q^2} - x_{\bar{q}}$$

$$Y(x, y) = \left(\frac{y^{q^2} - y_{\bar{q}}}{x^{q^2} - x_{\bar{q}}}\right)(x^{q^2} - X) - y^{q^2}$$

  and hence the re-written Characteristic Equation becomes

$$(X(x, y), Y(x, y)) = \bar{t}(x^q, y^q).$$

  Let's also set $(x_{\bar{t}}^q, y_{\bar{t}}^q) = \bar{t}(x^q, y^q)$. Cycling all over the possible values of $\bar{t}$ among the set $\left\{-\frac{l-1}{2}, \ldots, 0, \ldots, \frac{l-1}{2}\right\}$ and checking for which $\bar{t}$ the equality $X(x, y) = x_{\bar{t}}^q$ holds, leads us to restrict the choice of $\bar{t}$ to two

possibilities, namely the positive and negative one. Notice that we only exploited the x coordinate so far. By the comparison of the y coordinate we are then able to unequivocally determine which of the two is the right case. This returns the value of $\bar{t}$ that is t mod l.

- ◇ **2nd case:** $(x^{q^2}, y^{q^2}) = \pm\bar{q}(x, y)$

  Within this case, we distinguish two other subcases:

  - ○ $(x^{q^2}, y^{q^2}) = \bar{q}(x, y)$. In this case the re-written Characteristic Equation becomes

    $$(2\bar{q})(x, y) = \bar{t}\varphi(x, y)$$

    because we are actually summing the same point (hence computing its double) and

    $$(2\bar{q})^2(x, y) = \bar{t}^2\varphi^2(x, y) = \bar{t}^2\bar{q}(x, y).$$

    Therefore, $\bar{t}^2\bar{q} \equiv 4\bar{q}^2 \pmod{l}$, and this means that $\bar{q}$ is necessarily a square modulo l. Suppose for instance $\bar{q} \equiv w^2 \pmod{l}$. Then

    $$\begin{aligned}(x^{q^2}, y^{q^2}) = \bar{q}(x, y) &\iff (\varphi^2 - \bar{q})(x, y) = O \\ &\iff (\varphi + w)(\varphi - w)(x, y) = O.\end{aligned}$$

    It follows immediately that, $\forall P \in E[l]$ one of the following holds: either $(\varphi - w)(x, y) = O$ or $(\varphi + w)((\varphi - w)(x, y)) = O$. But then, in each case, there exists a point $P \in E[l]$ such that $\varphi(P) = \pm wP$. Again, let's split the cases. If $\varphi(P) = wP$, then

    $$\begin{aligned}O &= (\varphi^2 - \bar{t}\varphi + \bar{q})P = \\ &= (\bar{q} - \bar{t}w + \bar{q})P = \\ &= (2\bar{q} - \bar{t}w)P\end{aligned}$$

    so that $\bar{t}w \equiv 2\bar{q} \equiv 2w^2 \pmod{l}$ which clearly implies $\bar{t} \equiv 2w \pmod{l}$. In a similar way one proves that if $\varphi(P) = -wP$ then it happens that $\bar{t} \equiv -2w \pmod{l}$. Notice that computing w as a square root of $\bar{q}$ modulo l is a quite easy task.

  - ○ $(x^{q^2}, y^{q^2}) = -\bar{q}(x, y)$. In this case the re-written Characteristic Equation becomes

    $$-\bar{q}(x, y) + \bar{q}(x, y) = \bar{t}(x^q, y^q) \iff O = \bar{t}(x^q, y^q)$$

    and we immediately obtain $\bar{t} = 0$.

An additional case is, as already mentioned, when $l = 2$. If $x^3 + Ax + B$ has a root $e \in \mathbb{F}_q$, then $(e, 0) \in E[2]$ and $(e, 0) \in E(\mathbb{F}_q)$, so $E(\mathbb{F}_q)$ has even order. Hence in this case $\#E(\mathbb{F}_q) = q + 1 - t \equiv t \equiv 0 \pmod{l}$, so t is even and we

get the congruence $t_2 \equiv 0 \pmod 2$. If $x^3 + Ax + B$ has no roots in $\mathbb{F}_q$, then $E(\mathbb{F}_q)$ has trivially no points of order 2 and t is odd. A smart way to determine whether $x^3 + Ax + B$ has a root in $\mathbb{F}_q$ is to compute the greatest common divisor between $x^3 + Ax + B$ and $x^q - x$. In fact, we recall that the roots of $x^q - x$ are exactly the elements of $\mathbb{F}_q$. Therefore, $x^3 + Ax + B$ has a root in $\mathbb{F}_q$ if and only if it has a root in common with $x^q - x$, hence if and only if

$$gcd(x^q - x, x^3 + Ax + B) \neq 1$$

If it turns out to be 1, then $x^3 + Ax + B$ has no roots in $\mathbb{F}_q$ and $t_2 \equiv 1 \pmod 2$. Notice that and efficient way to compute the *gcd* is to exploit the Euclidean Algorithm. This finishes the case $l = 2$.
The following pseudocode summarizes Schoof's algorithm:

Algorithm 1 : Schoof's Algorithm

function SCHOOF(*an elliptic curve* E, *an integer* $q := p^b$)
  Choose a set of odd primes S *not containing* p *s.t.* $N := \prod_{l \in S} l > 4\sqrt{q}$
  if $gcd(x^q - x, x^3 + Ax + B) \neq 1$ **then** $t_2 \leftarrow 0$
  else $t_2 \leftarrow 1$
  for $l \in S$ **do**
    Compute the division polynomial ψ_l
    (*Computation performed in the ring* $\mathbb{F}_q[x, y]/(y^2 - x^3 - Ax - B, \psi_l)$)
    Let $\overline{q}$ *be the unique integer such that* $q \equiv \overline{q} \pmod l$ *and* $|q| < \frac{l}{2}$
    Compute $(x^q, y^q), (x^{q^2}, y^{q^2})$ *and* $(x_{\overline{q}}, y_{\overline{q}})$
    if $x^{q^2} \neq x_{\overline{q}}$ **then**
      Compute X, Y
      for $\overline{t} = 1, \ldots, (l-1)/2$ **do**
        if $X = x_{\overline{t}}^q$ **then**
          if $Y = y_{\overline{t}}^q$ **then** $t_l \leftarrow \overline{t}$
          else $t_l \leftarrow -\overline{t}$
    else if q *is a square modulo* l **then**
      Compute w *with* $q \equiv w^2 \bmod l$
      Compute $w(x^q, y^q)$
      if $w(x^q, y^q) = (x^{q^2}, y^{q^2})$ **then**
        $t_l \leftarrow 2w$
      else if $w(x^q, y^q) = (x^{q^2}, -y^{q^2})$ **then**
        $t_l \leftarrow -2w$
    else
      $t_l \leftarrow 0$
  Use the Chinese Remainder Theorem to compute t *modulo* N *solving the congruence system* $t \equiv t_l \ mod \ l$
  Deduce t *from* $t \bmod N$
  return $q + 1 - t$

The core of all our work is our MAGMA implementation of this algorithm, which is available at `Schoof.txt` among the annexes. By the way, it can be instructive to show a concrete example of how Schoof's algorithm works.

3.2 Example

Example 3 (Schoof)**.** *Let $E/\mathbb{F}_{19}$ be the elliptic curve of equation $y^2 = x^3 + 2x + 1$. Then, according to Hasse's theorem*

$$\#E(\mathbb{F}_{19}) = 19 + 1 - t$$

where $|t| \leq 2\sqrt{19}$. We want to determine t. Following the algorithm we have that $S := \{2, 3, 5\}$ is such that $19 \notin S$ and

$$\prod_{l \in S} l = 2 \cdot 3 \cdot 5 = 30 > 4\sqrt{19} \sim 17$$

We consider each single case separately:

- ◇ *case $l = 2$*

  This is the easy case. It is sufficient to compute

  $$gcd(x^{19} - 2, x^3 + 2x + 1) = 1$$

  The result state that $x^3 + 2x + 1$ has no roots in $\mathbb{F}_{19}$ hence E does not have any 2-torsion point. This implies $t \equiv 1 \pmod 2$.

- ◇ *case $l = 3$*

  First of all, we compute the third division polynomial:

  $$\psi_3 = 3x^4 + 12x^2 + 12x - 4$$

  Next, $q_3 := q \pmod l = 19 \pmod 3 = 1$. It follows immediately that in this case $\bar{q} = 1$. We now have that

  $$19(x, y) = (x, y) \quad \forall (x, y) \in E[3].$$

  Since $x^{19^2} = x^{361} \neq x$ in the quotient ring, we follow the algorithm by computing X and Y as coordinates of the point $(x^{361}, y^{361}) + (x, y)$. We have that

  $$X(x, y) = \left(\frac{y^{361} - y}{x^{361} - x}\right)^2 - x^{361} - x$$

  and after some reductions we discover that this quantity is equal to the x coordinate of the leftwing member of the Characteristic Equation for $\bar{t} = 1$. This means that the equation holds either for $\bar{t} = 1$ or $\bar{t} = -1$. Now we compute the y coordinate of $((x^{361}, y^{361}) + (x, y))$, which turns out to be

  $$Y(x, y) = y(14x^9 + 8x^7 + 2x^6 + 16x^5 + 8x^4 + 17x^3 + 8x^2 + 9)$$

  which is different (in the ring $\mathbb{F}_{19}[x, y]/(y^2 - x^3 - 2x - 1, \psi_3)$) from y, i.e.

the second component of $1(x,y) = (x,y)$. *Hence* $\bar{t} = -1$ *and* $t \equiv -1 \equiv 2 \pmod 3$.

- ◇ *case* $l = 5$

  As before, we compute the fifth division polynomial, which is

  $$\psi_5 = 5x^{12} + 10x^{10} + 17x^8 + 5x^7 + x^6 + 9x^5 + 12x^4 + 2x^3 + 5x^2 + 8x + 8$$

  In this case, $\bar{q}$ *such that* $\bar{q} \equiv q \pmod 5$ *and* $|\bar{q}| < \frac{5}{2}$ *is* $\bar{q} = -1$. *Note that* $19 \equiv 4 \equiv -1 \pmod 5$, *so* $\bar{q} = -1$ *and*

  $$19(x,y) = -(x,y) = (x,-y) \quad \forall (x,y) \in E[5].$$

  Again, $x^{361} \neq x$, *so we proceed by computing* X *and* Y *as coordinates of the point* $(x^{361}, y^{361}) + (x,-y)$. *The well known formula for the addition yields*

  $$X = \left(\frac{y^{361} + y}{x^{361} - x}\right)^2 - x^{361} - x.$$

  In this case we have a match between $X(x,y)$ *and* $x^{19}_{\bar{t}}$ *for* $\bar{t} = 2$, *therefore* $\bar{t} \equiv \pm 2 \pmod 5$. *To determine the sign, we look at the* y *coordinate, which for the point* $(x^{361}, y^{361}) + (x,-y)$ *is*

  $$y(9x^{11} + 13x^{10} + 15x^9 + 15x^7 + 18x^6 + 17x^5 + 8x^4 + 12x^3 + 8x + 6)$$

  instead for $2(x,y)$ *it is*

  $$y(13x^{10} + 15x^9 + 16x^8 + 13x^7 + 8x^6 + 6x^5 + 17x^4 + 18x^3 + 8x + 18)$$

  After some computations we discover that these y *components opposite in the ring* $\mathbb{F}_q[x,y]/(y^2 - x^3 - Ax - B, \psi_l)$, *hence* $\bar{t} = -2$ *and* $t \equiv -2 \pmod 5$.

At the end of the day we obtain the following system:

$$t \equiv \begin{cases} 1 \pmod 2 \\ 2 \pmod 3 \\ 3 \pmod 5 \end{cases}$$

The Chinese Remainder Theorem returns us that $t \equiv 23 \pmod{30}$, *but since* $|\, t \,| < 2\sqrt{19} < 9$, *reducing the result within our acceptance bounds, we conclude that* $t = -7$ *and we can finally end with* $\#E(\mathbb{F}_{19}) = 19 + 1 - (-7) = 27$.

3.3 Correctness demonstration

This subsection is dedicated to convince the reader of the correctness of the algorithm and of our implementation.

From a *theoretical* point of view, it is clear that the Schoof's machinery *must work*. This statement owes its veracity to the remark that every single step has

been formally demonstrated as theorem or as consequence of a theorem, that classic rules of inferential logic has been applied to move from every stage to the next one, that no unproven suppositions have been formulated and exploited. To mention an example, we emphasize how, at line 12 of the pseudocode on page 16, that for-loop *has to* produce the right and unique $\bar{t}$ because of the existence and uniqueness result given by Theorem 1 on page 7 (and hence it is not a lucky coincidence that a unique $\bar{t}$ is always found). A further emblematic example is the usage of the *law of the excluded middle* (also known as *tertium non datur*) from line 10 to line 24 in the same pseudocode. This way to proceed allows to covers all possible cases and guarantees that no other possible case has been neglected.

Instead, from an *implementative* point of view, we have to draw up a different discourse. Even though our code straightly retraces the Schoof's procedure step by step, we cannot be certain that a bug did not occur. For the most important and complex functions we wrote, we did some debugging labour in order to make sure that those functions were producing the right output. For the smaller functions and for the MAGMA primitives (which are by the way supposed to be correct) we did not run any superfluous test. Indeed, we directly tested the main function (the `Schoof()` one) and checked it it outputted the right result.

As typically happens in such situations, the idea to test the correctness of our code is to take a preexisting (obviously exact) list of solutions, given by pairs of the form "*input* $\rightarrow$ *correct output*" and to compare the outputs of our function with those on the list, when tested on the corresponding inputs. Since no real list like that effectively exists, our idea has been to exploit the "`#`" MAGMA native function, which is clearly designed to be infallible. Our test was designed to stop and launch an alarm message if a discrepancy between our output (`Schoof(E)`) and the standard output (`#E`) was found. Many tests have been conducted: in the first instance the program was tested for all non-singular elliptic curves definable over a finite field with cardinality between 5 and 100 and something like 10000 curves were tested. Then we tried to see if the algorithm presented an anomalous behavior when testes on particular curves defined over larger fields and to this purpose the program was randomly tested on curves defined over fields with cardinality up to 10^6. We never found an irregularity. The code employed to the test part can be found in the form of comment at the end of all the attached files containing an algorithm.

Not without obstacles we carried on our work. At a certain moment we found ourselves with a nearly correct program, which however returned, albeit infrequently, the wrong output. As provocatively asserted in the previous lines, every MAGMA native command or internal operation *should* be correct, and this is why we analyzed the structure of the algorithm until exhaustion and focused only on our edited functions and examined them plenty of times. We did not find

any errors, and a doubt was increasingly arising in our consciences: "*what if it is* MAGMA *that is failing?*" After some debugging procedures, we discovered that the problem was in the computation of $(X(x,y), Y(x,y)) := (x^{q^2}, y^{q^2}) + \bar{q}(x,y)$ in the quotient ring

$$\mathbb{F}_q[x,y]/(y^2 - x^3 - Ax - B, \psi_l)$$

In particular, we discovered that when the prime l divides the order $|E(\mathbb{F}_q)|$ of the curve, it happens that the polynomial that appears as denominator of the two components of the sum point has a factor in common with the division polynomial ψ_l (hence $gcd(denominator, \psi_l) \neq 1$), and it is consequently non-invertible in the quotient ring above. Although this should not theoretically be a problem since the rational function given by $X(x,y)$ *must* symbolically represent a value in $\mathbb{F}_q$ and it is hence impossible that it turns out to be uncomputable because, indeed, a simplification between numerator and denominator *must* always be possible in this case, MAGMA is instead not able to finalize the simplification. Even worse. It performs a wrong computation in a way we still don't know, ignoring the fact that a quantity cannot be inverted, and doesn't send any error message. How it is easy to understand, we wasted a lot of time trying to firstly detect and secondly solve this issue. When we understood that the problem was about computing in the quotient ring, we forced MAGMA to ask itself if the two quantity, which on paper should be in principle equal, were actually equal. It happened that MAGMA definitively sent a long and complicated error, which MAGMA seemed to recognized as its own "internal error" (the exact term appearing on the console). We proceeded by reporting all what we discovered to the MAGMA company. We hope to have contributed in this way to the improvement of the programming language and to have helped the developers to discover further mistakes. How did we solve our problem, in the end? Since we had to verify for what value of $\bar{t}$ a certain equality held, we got rid of the denominators by computing the crossed products (hence by moving a denominator to the numerator on the other side of the equal sign), computing them in the quotient ring (now an easy task) and checking the equality under this form.

This anecdote made me wonder about the discrepancy between understanding an algorithm and implementing it. In fact, the pseudocode on which we based our implementation did not clearly take this issue into account, since all steps are described on a more abstract level than the one which the practice requires.

3.4 Implementation choices

A dissertation on the ideas and the decisions behind our work follows. In primis, a discourse about the choice of the adopted programming language is due.

The reasons which led us to the use of MAGMA are multiple, but the most important one is that it is THE programming language for a mathematician. MAGMA is built in such a way that it is able to perform almost every mathematical operation from all mathematical branches (and in particular Algebra), e.g. it has a command to generate a Free Lie Algebra, to compute the eigenvalues of a given matrix, to calculate the automorphisms group of a certain polytope. It is basically equipped with every useful function inside the math-world, and for what concerns our needs, everything of the series finite fields, polynomial and quotient rings, elliptic curves is available. Below the list of all MAGMA primitives we exploited, whose names are quite self-explanatory. How it is easily understandable, this programming language represent the most suitable option for our purposes, since a very dense theoretical framework is required by Schoof's algorithm. We used:

- ◇ `GF(q)` (creates the Galois Field (Finite Field) with q elements);
- ◇ `PolynomialRing(R,n)` (polynomial ring in n variables with coefficients in the ring R);
- ◇ `quo<R|J>` (quotient ring of R through the relations J);
- ◇ `ChineseRemainderTheorem(X, N)`;
- ◇ `IntegerRing(n)` (returns $\mathbb{Z}/n\mathbb{Z}$);
- ◇ `Lcm(S)`;
- ◇ `LegendreSymbol(q,l)` (computes the Legendre Symbol of q, hence verifies if q is a quadratic residue modulo l);
- ◇ `PrimesUpTo(m)`.

Although some the following functions already exist as MAGMA primitives, we decided to override them for both a personal challenge and a didactic exercise. In fact

- ◇ `choosePrimes(p,q)` (returns the list of primes used in Schoof's algorithm, satisfying some already met conditions);
- ◇ `DivPol(S,p,t)` (returns the t-th division polynomial belonging to the polynomial ring S relative to a certain fixed elliptic curve E, exploiting the array p of the previous division polynomials already computed and stored);
- ◇ `ScalarPol(Q,p,k)` (returns a couple of polynomials belonging to Q which respectively represent the x and the y coordinate of the point $k(x, y)$, i.e. the k-scalar multiple of a generic point $P = (x, y)$ of a certain fixed elliptic curve E).

Despite the fact that MAGMA offers a wide range of comforts, we did not want to rest on our laurels and decided thus to implement some auxiliary function not only to challenge ourselves and to immerse ourselves deeper in the problem, but also to keep us trained and fresh with regard to this programming language. In fact, this latter is the most popular and classic language within the cryptographic and coding sphere.

A last remark concludes: there is a big trade-off between staying close to the machine language and gaining in efficiency: the more a programming language artificial, the slower its performances, although the more comfortable. We decided to use MAGMA for the reasons we explained above, but this has unavoidably affected the efficiency of our program, as we are going to see in the last subsection of this chapter. In this sense, a C implementation would have surely been more high-performing. We also remember that the primitives of a language are designed to be optimized at best, hence another cause of execution velocity loss lies in our choice to implement from scratch some functions which already existed as MAGMA primitives.

3.5 A significant improvement

A particular case happens when we are given an elliptic curve $E/\mathbb{F}_q$ and we want to compute the cardinality of $E(\mathbb{F}_{q^n})$, hence of the extension of E to points whose coordinates live in some superfield of $\mathbb{F}_q$, for some $n \geq 1$. Thanks to the below stated result, it is possible to compute $|E(\mathbb{F}_{q^n})|$ by simply computing $|E(\mathbb{F}_q)|$ (notice that we would still be able to solve this task using the standard Schoof's algorithm).

Theorem 2. *Consider an elliptic curve $E/\mathbb{F}_q$. Let $|E(\mathbb{F}_q)| = q + 1 - t$. Write $X^2 - tX + q = (x - \alpha)(X - \beta)$. Then for all $n \geq 1$ the following holds:*

$$|E(\mathbb{F}_{q^n})| = q^n + 1 - \alpha^n - \beta^n$$

Bear in mind that to apply this result it is necessary to be dealing with a curve defined over $\mathbb{F}_q$, hence such that its coefficients belong to $\mathbb{F}_q$. This is a very strict condition and it is hardly ever known, but one can always check if the coefficients A and B are actually elements of some subfield of the given one (even though some computations are needed). Sometimes it is convenient to apply this argument because we know "a priori" that the elliptic curve we are dealing with admits coefficients in a field smaller than the one in which we want to count points. That's maybe because we were initially studying the curve in the base field and then decided to move in a bigger field.

We implemented our version of this mathematical trick in an isolated file, named `Extender.txt`. This choice is due to the fact that we preferred to keep Schoof's algorithm unaffected by further variations, so that a both theoretical and computational integrity was achievable. Indeed, an adaptation of our code to include this case was possible, however such a change would have altered

the study of the complexity and of the execution times of our implementation, because of the different treatment that certain cases would have received.

However, in the next section we are going to show, among other results, the efficiency impact of our extension program, when tested in comparison to the standard Schoof's algorithm.

3.6 Execution time, tests and comparisons, results

This part collects all the analysis of the execution times of the programs we implemented as well as some efficiency comparisons. All computations have been performed on a 8 GB RAM Dell XPS 13 laptop with an Intel Core i5-6200U CPU @ 2.30GHz $\times$ 4 equipped with Ubuntu 19.10 (64 bit) as operating system. Each algorithm has been tested with primes of different ranges. The execution speed has been measured via the MAGMA primitive function `Cputime()`, which returns the time spent expressed in seconds. Notice that since the Baby-Step Giant-Step algorithm is not a deterministic algorithm, and therefore it may not end the execution, we have chosen to test it only for those values of q that guarantee the correct execution of the program, so that it was possible to compare it with the other deterministic algorithms.

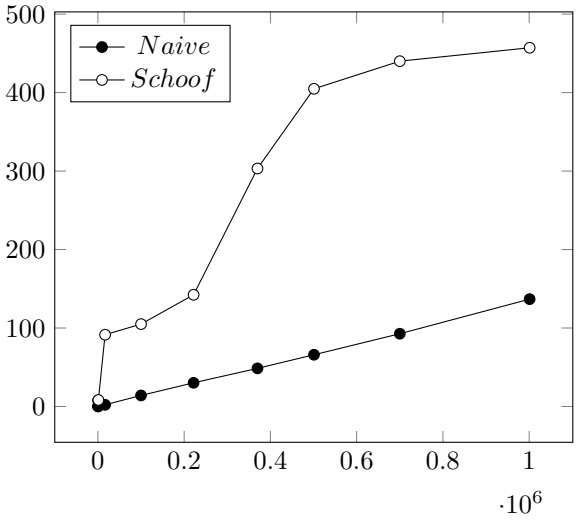
To conduct our test, we proceeded in the following way: for every manageable order of size, we chose a random prime q belonging to it and we tested each algorithm on 100 non-singular curves randomly defined over $\mathbb{F}_q$, and then we took the average value of the results. Please notice that these data are very sensible with respect the laptop itself and to the background processes filling the RAM. The following table summarizes all the results we got from these multiple tests:

PRIME q	NAIVE	B-S G-S	SCHOOF
431	0.080 s	0.050 s	7.620 s
1103	0.160 s	0.060 s	8.320 s
16651	2.140 s	0.060 s	91.440 s
100003	14.080 s	0.080 s	104.980 s
221831	30.160 s	0.120 s	142.360 s
370003	48.580 s	0.140 s	303.260 s
501013	65.920 s	0.160 s	404.820 s
700027	92.720 s	0.180 s	440.040 s
1001069	136.780 s	0.220 s	457.120 s

In the first column the prime q, starting from which the finite field $\mathbb{F}_q$ was built, appears; the other three columns report the execution times on one hundred curves of the three algorithms we implemented. As we have already stated, these data are the fruit of an average procedure: this is because we observed

that, while repeatedly testing on the same q, the outcome underwent some (albeit slight) variation, because of the random factor in the construction of the elliptic curves. Some cases probably allowed the algorithms we implemented to work faster, instead other cases heavily challenged our functions, which had to work with particularly complex curves.

If we represent these data on a Cartesian graph (with the exception of the B-S G-S column, which wouldn't be much meaningful), and connect the points with the "trend" line, we get the following representation, which manifestly shows the evolution of the execution times of the naive and Schoof's algorithms for q verging to infinity:



On the x axis the previously tested primes up to 10^6 are shown; likewise, on the y axis, the programs execution times, expressed in seconds. Notice that we chose our random primes q among some "ad hoc" ranges which have allowed us to obtain a clear picture like the one above. Unfortunately, the increasing computational cost require by our implementation has forced us to test our algorithms only for q up to the order of 10^6. For bigger q, some hours would have been necessary. From the information we collected we can immediately make some comments:

- ◇ the Baby-Step Giant-Step algorithm returns excellent performances, which have to be imputed to its streamlined implementation exploiting some MAGMA primitives and to its *probabilistic* and not *deterministic* nature, which legitimates its incredibly quick behavior, especially for curves defined on relative small fields;
- ◇ the behavior of the Naive algorithm fits the theory, in the sense that it grows linearly, as its computational cost ($O(q)$) envisages;

- ◇ also the behavior of Schoof's algorithm is in accordance with the theoretical results: indeed, thanks to the plot, we can observe its logarithmic evolution for $q \to +\infty$ (notice that Schoof's plot looks actually like the graph of the function $y = \log^8 x$ for the first 10^6 values. The reader can try to use the "Desmos" software to convince himself).

Unluckily, because of the strict constraints given by the programming language we used, it was not possible to study the behavior of the algorithms for some bigger q. However, ideally extending the plotted functions, it is not difficult to understand that the Naive cost function will *definitively* (which is the technical mathematical term) jump over the Schoof one.

At this point, one could bring forward some criticism of the type: why should Schoof's algorithm be considered superior to the other ones? Moreover: the B-S G-S seems to be the fastest one, doesn't it? The answer to these questions is that it would be pretty pointless to carry out such a comparison at this level. For instance, using the same reasoning as before (hence looking at the asymptotical behavior of all the execution times), it would result that the B-S G-S scales as $O(\sqrt[4]{q})$. Also in this case, as we known from a basic Analysis 1 course, it will happen that $\log^8 q$ will definitively drop under $\sqrt[4]{q}$. The point is that this happens for a q that is too large to be appreciated, i.e. for $q \approx 10^{70}$ (as one can verify by solving, with the help of a software, the transcendental equation $\log^8 x - \sqrt[4]{x} = 0$). Obviously, this order of size is out of our possibilities, as well as of out intents. Indeed, our code hasn't been thought for a distribution purpose, rather for an academic one. With a more efficiency-oriented programming language (like C) we would have been able to drastically reduce the computation costs of *all* our algorithms, so that more advanced computation would have been possible. Thanks to a scale reduction of this kind, the true nature of these algorithms would have emerged. A last observation which one should take into account while answering to the previous wonders is that the B-S G-S is a probabilistic algorithm. This means that for huge q it wouldn't even handle the competition with a well implemented Schoof, because it would almost surely be condemned to fail.

For what concerns the extension program, we compared it with the standard Schoof's algorithm in the way we are going to explain. Firstly we fixed $q = 13$, then considered the elliptic curve $E/\mathbb{F}_{13}$ defined by $y^2 = x^3 + 3x + 2$ and tried to compute $E(\mathbb{F}_{13^n})$ suitable n in both manners (notice that, since the curve E is originally defined over $\mathbb{F}_{13}$ and hence its coefficients belong to this latter field, we are actually in the hypothesis of Theorem 2). The results of our tests are summarized in the following table.

n	SCHOOF ON $E(\mathbb{F}_{13^n})$	SCHOOF ON $E(\mathbb{F}_{13})$ + EXTENDER
1	0.010 s	0.010 s
2	0.060 s	0.010 s
3	0.120 s	0.010 s
4	0.460 s	0.010 s
5	12.700 s	0.010 s
6	34.740 s	0.010 s

Such data are very self-explanatory. In fact, it is not surprising that the last column of the table, corresponding to the execution times of the extension program, contains all constant values. Furthermore, if we look at it more carefully, we notice that these timings are all equal to the execution time of the pure Schoof's algorithm in the case $n = 1$. This is because in the usage of the `Extender.txt` file, the bulk of the whole computation lies only in the computation of $E(\mathbb{F}_{13})$ (which explains the equality between the data in the first row and in the third column) and the remaining operations are computationally negligible. This experiment didn't yield anything we didn't expect.

3.7 Post-Schoof Algorithm: SEA (brief sketch)

Here we describe an algoritmh that has been designed after Schoof's original work and that deeply sinks its roots in it. Its name is SEA, and it comes from the names of the mathematicians who figured it, such as Schoof (who did not work with the following two, but who the whole main structure was taken from), Elkies and Atkin. A rapid overview is shown for only a purpose of expositive completeness, even because its implementation goes far beyond the intents of our work. We inform the reader that the MAGMA primitive function `#()` to compute the cardinality of a given elliptic curve is based on SEA, together with a further upgrade.

In the nineties Noam David Elkies and Arthur Oliver Lonsdale Atkin proposed an improved version of the Schoof's algorithm, since they observed that for a very large $q \sim 10^{200}$ the algorithm has to deal with primes $l > 250$. This fact represents a weak spot for Schoof's machinery, because of the huge degrees of the division polynomials involved, that we recall to be $\frac{l^2-1}{2}$ (hence about 30000 for a prime near to 250). For such an l, the representation of one element in the ring $\mathbb{F}_q[x, y]/(y^2 - x^3 - Ax - B, \psi_l)$ requires an enormous quantity of memory, and another crazy amount is requested to operate within that ring. It it then clear that the goal to pursue is to decrease the degree of polynomials ψ_l appearing as quotient relations.

Their idea was to distinct the primes involved in the computation on the basis of a certain good property, which now we explain. We call an *Elkies prime* a prime l such that the Characteristic Equation $\varphi^2 - t\varphi + q = 0$ splits over $\mathbb{F}_l$.

Vice versa, we define a prime l to be an *Atkin prime* if it is not an Elkies prime. In general terms, these definitions posed, one can use an Elkies prime to avoid using the l-th division polynomial, which would lead to annoyingly high degrees, and simply work with particular polynomials, known as modular polynomials, whose degree is at most $\frac{l-1}{2}$. With this mathematical trick Elkies and Atkin were able to reduce the computational cost of the algorithm from $O(\log^8 q)$ to $O(\log^4 q)$.

4 Cryptographic application

We have so far understood that counting point is important for some cryptographic applications. But how can such an information be that useful? In practice, there are several criteria that provide *necessary* (but not *sufficient*, as we are going to discuss) conditions for a curve to be considered "secure" (whatever that means) which are based right on the nature of $|E(\mathbb{F}_q)|$. Keep reading below for specific details.

4.1 Security of a curve: theoretical criteria

Suppose that Alice and Bob want to exchange a secret key using the Diffie-Helmann protocol, and suppose they want to rely on ECDLP. Obviously, they want to use a curve sucht that the ECDLP on that particular curve is as difficult as possible, hence a curve that could be define as a "secure" curve. One could here wonder about the meaning of this adjective. From a formal point of view, the word "secure" belongs to the natural language and is more a colloquial term rather than a mathematical term, and has indeed no mathematical definition. An elliptic curve is said to be secure when no attacks aimed to break it are known. The security of a curve is, intuitively, its strength to endure in the course of time against all attempts of violation. This does not clearly mean that the curve is immune from every kind of attack, because many other attacks are maybe possible, although currenlty still unknown. In this sense we speak about necessary but not sufficient requirements. By the way, the NIST (which is the National Institute of Standards and Technology) provides some curves which are considered to be secure (e.g. P-256, P-384 and P-521), together with some standard criteria which should be verified by an elliptic curve to be a valid candidate for the role of "secure curve". The following conditions are necessary to avoid all the so far known attacks:

- ◇ the most intuitive condition concerns the dimension of $|E(\mathbb{F}_q)|$, which we would like to be pretty much large, or alternatively a "small" multiple of a "reasonably large" prime, hence

  $$|E(\mathbb{F}_q)| = c \cdot l$$

  where c is an integer and l is prime. From Cauchy's Theorem (Group Theory) follows that the curve contains points with order l and hence, if we

smarlty choose our starting point P, the corresponding discrete logarithm problem is not trivial to solve;

- ◇ we don't want the curve to be *anomalous*, i.e. we require that

$$|E(\mathbb{F}_q)| \neq q$$

In fact, in 1999, N.P. Smart showed that there exists a way to solve in linear time the discrete logarithm problem on this particular class of elliptic curves. We refer to [4] for further details.

- ◇ consider an elliptic curve $E/\mathbb{F}_q$ and suppose it has passed the first test here above, hence $|E(\mathbb{F}_q)| = a \cdot l$ where l is a great prime. From the Group Theory we know that there exists a unique subgroup of $E(\mathbb{F}_q)$ with order l. For what little we need, we can define the *embedding degree* k of such a curve to be the smallest integer $k \geq 1$ such that $l \mid (q^k - 1)$. For security purposes, we require k to be ≥ 100. In fact, there exist specific attacks like the MOV (Menezes-Okamotoand-Vanstone) and the FR (Frey-Rück), which are reduction-attacks. This means that they act by transforming the ECDLP in a classic DLP defined in finite field of size q^k, for which sub-exponential algorithms are known. The condition $k \geq 100$ can nevertheless kill any possibility of success with a reduction of this kind;

- ◇ we don't want the curve to be *supersingular*, i.e. we require that

$$|E(\mathbb{F}_q)| \not\equiv 1 \pmod{p}$$

This requirement is due to the fact that again Menezes, Okamotoand and Vanstone proved that for a supersingular curve it holds that the embedding degree k defined some lines above satisfies $k \leq 6$, which originates a problem, since, if k is small, a reduction like the one explained before is possible and hence the discrete logarithm problem has an easy solution.

The reader will find a file named `SecurityTest.txt` among the attached. With that script we tried to implement a little program which returns "true" if the input elliptic curve is such that $|E(\mathbb{F}_q)|$ verifies the up here conditions, and "false" is even just one condition is falsified. Notice that, unlike the last three conditions of the bullet list above, the first one has a quite lax formalization, in the sense that it is not clear what it is meant with "small" and "large". Because of the ambiguity of the constraint, we decided to introduce an auxiliary parameter, called `MAX_MULTIPLE`, which works as follows: we remove from the order of the curve, if they exist, small prime factors dividing it until their product is less than the auxiliary constant. When this process finishes, we will have factored $E = c \cdot l$, where c is the product of the small primes just found. Then we look at l and check if it is prime or not. The default parameter is set to 10000, but it can be varied according to our needs. The smaller this parameter, the smaller the probability of l to be prime.

To compute the cardinality of the input elliptic curve we decided to use the native MAGMA function `#E()` and not our `Schoof()` function. This choice is due to the fact that the first command guarantees a fast execution time and consequently satisfactory results even in the case of a very large q. A short example follows.

4.2 Example

Example 4 (Security test)**.** *It is given the elliptic curve*

$$y^2 = x^3 + 202042051720180045605\,x + 285741207313617940766$$

defined on $\mathbb{F}_q$ *with* $q = 602666154775839791171$ *prime. To start we compute the cardinality of the curve, which is*

$$|E(\mathbb{F}_q)| = 602666154757320329832 = 2^3 \cdot 3 \cdot 13 \cdot 1931622290888847211.$$

Note that the last factor is a prime number. The protocol then proceeds to consider the other conditions: clearly $|E(\mathbb{F}_q)| \neq q$ *hence the curve is not anomalous, furthermore*

$$|E(\mathbb{F}_q)| \equiv 602666154757320329832 \equiv -18519461339 \not\equiv 1 \pmod{q}$$

and hence the curve is also not supersingular. The embedding degree of the curve is then computed to be $386324458177769442 \gg 100$ *and we can finally conclude that the given curve is potentially eligible for the role of "secure curve".*

References

[1] Lawrence C. Washington, *Elliptic Curves, Number Theory and Cryptography*, Chapman and Hall/CRC, Boca Raton, 2008.

[2] Renè Schoof, *Elliptic Curves over Finite Fields and the Computation of Square Roots mod p*, Mathematics of Computation, vol. 44 no. 170, April 1985, 483-494.

[3] Renè Schoof, *Counting Points on Elliptic Curves over Finite Fields*, Journal de Thèorie des Nombres de Bordeaux 7, 1995.

[4] Nigel P. Smart, *The Discrete Logarithm Problem on Elliptic Curves of Trace One*, HP Laboratories Bristol, October, 1997.

[5] The Magma Handbook, available at:
`https://magma.maths.usyd.edu.au/magma/handbook/`