Società Nazionale di Scienze Lettere e Arti in Napoli

Rendiconto dell'Accademia delle Scienze Fisiche e Matematiche

Antonio Corbo Esposito, Cristian Tirelli

A review about public cryptography protocols based on RSA or elliptic curves

Rendiconto dell'Accademia delle Scienze Fisiche e Matematiche, Serie 4, Vol. **86** (2019), n.1, p. 123–146.

Società Nazione di Scienze, Lettere e Arti in Napoli; Giannini

<http://www.bdim.eu/item?id=RASFMN_2019_4_86_1_123_0>

Rendiconto dell'Accademia delle Scienze Fisiche e Matematiche, Società Nazione di Scienze, Lettere e Arti in Napoli; Giannini, 2019.

Rend. Acc. Sc. fis. mat. Napoli
Vol. LXXXVI, (2019) pp. 123-146

DOI 10.32092/1021

A review about public cryptography protocols based on RSA or elliptic curves

Nota di Antonio Corbo Esposito[1], Cristian Tirelli[1]

Presentata dal socio Antonio Corbo Esposito
(Adunanza del 15 novembre 2019)

Key words: RSA, Diffie-Hellman, Discrete Logarithm, Elliptic Curve, ECDSA

Abstract – We provide the basic definitions regarding computational complexity theory and review some basic cryptography protocols based on RSA or elliptic curves. These protocols summarize the history of the last fifty years in cryptography and are actually ubiquitous in applications, as for example SSL (secure socket layers), smartcards, creation of a bitcoin wallet etc. Since it is known they are in the polynomial class for the Shor's algorithm, the possible development of quantum computers, needed to run such algorithm, will represent a dramatic shift in cryptography research and in applications.

Riassunto – In questa nota forniamo le definizioni di base relative alla teoria della complessitá computazionale ed esaminiamo alcuni semplici protocolli crittografici basati su RSA e curve ellittiche. Questi protocolli riassumono la storia degli ultimi cinquant'anni della crittografia e sono onnipresenti nelle applicazioni, come ad esempio SSL (secure socket layers), smartcards, creazione di portafogli per bitcoin etc. Poiché é noto che rientrano nella classe di problemi polinomiali per l'algoritmo di Shor, un possibile sviluppo dei computer quantistici, necessari per eseguire questi algoritmi, rappresenterebbe un drammatico cambiamento nella ricerca crittografica e nelle applicazioni.

1 - INTRODUCTION

In this review we summarize the basic concepts about complexity theory and provide an overview of main public cryptography protocols actually used for a widespread range of applications, from the protections of money transactions over internet to the creation of a bitcoin wallet to the digital signature of documents.
While these algorithms are ubiquitous in the present ICT world they all share a common feature: they are all vulnerable to the Shor's algorithm, meaning that if

[1]Dipartimento di Ingegneria Elettrica e dell' Informazione "Maurizio Scarano", Università degli studi di cassino e del Lazio meridionale, Via G. Di Biasio 43, 03043, Cassino, Italia. e-mail: corbo@unicas.it, cristiantirelli@gmail.com

a "quantum computer" is ever realized managing a suitable number (i.e. >128) of qubits, the complexity to solve these problems will fall into the polynomial class.
The only problems present in this review which will resist to "quantum computers" are NP-complete problems, as MQ problem. However MQ problem, as other "post quantum" cryptographic problems has failed so far to provide adequate and practical public cryptography protocols. This review therefore provides an outlook directed to the recent history of cryptography: this history has provided exceptionally beautiful topics in mathematics, as RSA algorithm or the theory of elliptic curves, while the fate of the protocols based upon these algorithms is already sealed.
The world of cryptography is actively preparing for the upcoming paradigm shift. New algorithms are proposed and NIST (National Institute of Standards and Technology) has already started the selection for the next generation of protocols that will resist even attacks running on "quantum computers".
The present situation and perspectives will be addressed in an upcoming report.

2 - COMPUTATIONAL COMPLEXITY THEORY

Computational complexity theory is a tool that help us to classify and analyze computational problem to their inherent difficulty.
Information Theory tells us that every cryptographic algorithm is insecure, but with the study of computational complexity we can tell after how much time this algorithm can be broken.

2.1 - Algorithms complexity

The complexity of an algorithm is defined like the power needed to compute it. Usually it's measured considering two variables: T the time complexity and S the spatial complexity. The complexity of an algorithm is expressed using the Big O notation, that allow us to approximate the result using a superior limit.

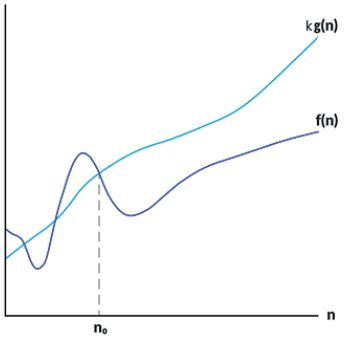

Figure 1: Big O notation

Definition 1. *$f(n) = \mathcal{O}(g(n))$ means that $k \cdot g(n)$ is the superior limit of $f(n)$. So there exists a constant k such that $f(n) \leq k \cdot g(n)$ for values bigger enough of n*

Definition 2. *$f(n) = \Omega(g(n))$ means that $k \cdot g(n)$ is the inferior limit of $f(n)$. So there exists a constant k such that $f(n) \geq k \cdot g(n)$, for values bigger enough of n*

Definition 3. *$f(n) = \Theta(g(n))$ means that $k_1 \cdot g(n)$ is the superior limit of $f(n)$ and $k_2 \cdot g(n)$ is the inferior limit for each $n \geq n_0$. So there exists some constants k_1 and k_2 for which $f(n) \leq k_1 \cdot g(n)$ and $f(n) \geq k_2 \cdot g(n)$ with this we can say $g(n)$ is a good approximation of $f(n)$.*

This notation is usefull to group in the same class of complexity different algorithm. The most common class of complexity are:

- Constant functions: $f(n) = 1$ when we don't have a dependence of n
- Logarithmic functions: $f(n) = \log n$
- Linear functions: $f(n) = n$
- Superlinear functions: $f(n) = n \log n$
- Quadratic functions: $f(n) = n^2$
- Cubic functions: $f(n) = n^3$
- Exponential function: $f(n) = c^n$
- Factorial functions: $f(n) = n!$

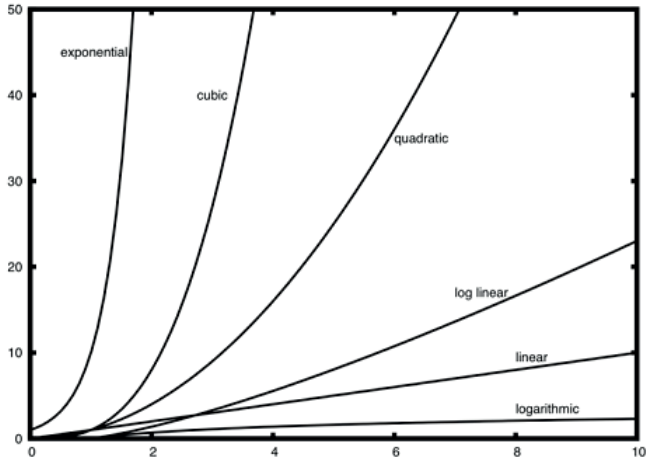

Figure 2: Execution time

To have a better gasp of the running time associate to each complexity function we can read the following table:

Classes	Complexity	#operation for $n = 10^6$	Time
Constant	$\mathcal{O}(1)$	1	1 μsec.
Linear	$\mathcal{O}(n)$	10^6	1 sec.
Quadratic	$\mathcal{O}(n^2)$	10^{12}	11.6 days
Cubic	$\mathcal{O}(n^3)$	10^{18}	32000 yrs.
Exponential	$\mathcal{O}(2^n)$	10^{301030}	10^{301006} y.u.

We can see that bruteforce attack for problems of exponential complexity are not practically possible, since they will require enormous amount of time, in fact this is the strengths of cryptographic algorithms.

2.2 - Complexity class

Cryptography is a science that was developed together with writing, in fact there are numerous example of "secret writing" in the history. In the past all the algorithm used to communicate secretly where based on the secrecy of the algorithm itself, but right know it's the complete opposite. Modern cryptography made possible for two strangers to communicate securely using a channel secured by the application of some mathematical problems and everything is done with public algorithms.

Computational complexity theory classify not only the complexity of algorithms, but also the complexity of problems in general. The objective is to find, using less time and space possible , the solution to a problem on a theoretic computer called Turing machine. Problems are divided in various class of complexity, determinated on the base of their difficulty. In the following image we can see various class and their relation, however mathematical a lot of topics still need to be demonstrated.

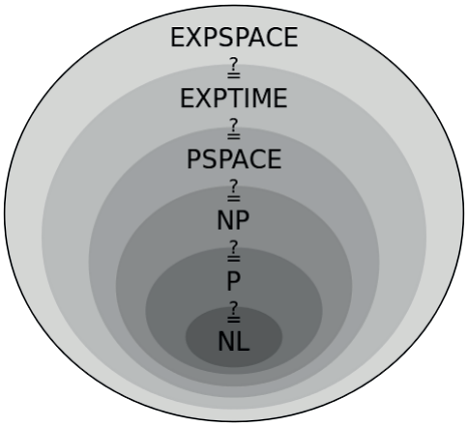

Figure 3: Diagram of the complexity class

2.3 - P vs NP

Definition 4. *The class P contains all the problem that can be solved in a polynomial time.*

Definition 5. *The class NP contains all the problem of which the solution can be verified in a polynomial time.*

Like we can guess from the definitions the membership to a class NP require less then the memebershipt to the class P. The class NP contains P because every problem that can be resolved in polynomial time can also be verified in polynomial time. It seems easy to prove that NP problems are more difficult of P problems, but right now a mathematical demonstration still has to be made. If $NP = P$ will be demonstrated, a lot o cryptographic algorithms could become useless. Some of this NP problems are more difficult than others and are called $NP - complete$.

Definition 6. *A P problem is called $NP - complete$ is every other problem Q, contained in NP, can be reduced to P in a polynomial time*

Example 1. *NP-complete problems:*

- *The knapsack problem. Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible*
- *Graph coloring problem. Given n colors, find a way to color the vertex of a graph such that two adjacent vertex doesn't have the same color.*
- *Travelling salesman problem. A traveler need to visit n different cities following the shortest route possible.*

If at least one of this problems would be verified with an algorithm that run in polynomial time, then we can say that $NP \neq P$. All this problems are easy to verify, but hard to solve. They complexity is exponential, but that doens't mean that every NP problem is exponential and this is what brings us to ask ourselves if $P = NP$. Nowday no mathematical demonstration has succeeded in giving us an answer.

2.4 - Boolean satisfiability problem

Boolean satisfiability problem (**SAT**) is a fundamental problem in the mathematical logic and in the theory of computational complexity. In practice it's a tool used for a variety of problems, for example there are numerous application of it for solving design problem of integrated circuits.
Given a set of n variables: $x_1, \ldots, x_n$, a set of literals (a literals is a variable $Q = x$), a set of distinct rules $C_1, \ldots, C_n$, and each rules is a set of literals bound together by the logic operation or ($\vee$).
The objective of SAT problems is to find the value that associate to the literals, make the following formula true.

$$C_1 \wedge C_2 \wedge \ldots \wedge C_m$$

2.5 - Multivariate quadratic polinomial (MQ) problem

Another importan $NP-Complete$ problem is the Multivariate Quadratic polinomial (MQ) problem that sees its application in the public key cryptography. A system of this kind have a set of quadratic polynomials on a finite field and solving this problems on a finite field is not easy. This kind of problem is considered to be one of the few that could resist to a quantic computer. The public key cryptosystem depends on the existance of a class of function called *trapdoor*; this function are easy to calculate, but very difficult to invert (for example the multiplication between two prime numbers). In the PKV (public key cryptosystem) that uses MQ systems, the trapdoor function is a polynomial equation with more then one boolean variables. Usually the key is generated with a system of quadratic polynomials:

$$\mathscr{P} = (p_1(\omega_1, \ldots, \omega_n), \ldots, p_m(\omega_1, \ldots, \omega_n))$$

where p_i is a non linear quadratic polynomial system over $w = (\omega_1, \ldots, \omega_n)$:

$$p_k(w) := \sum_i P_{ik}\omega_i + \sum_i Q_{ik}\omega_i^2 + \sum_{i>j} R_{ijk}\omega_i\omega_j$$

in which each coefficient and each variable is in $\mathbb{F}_q$.

3 - CRYPTOGRAPHIC PROTOCOLS

Sharing data and information on a secure channel between two endpoint was something needed also during the egyptian times and only with the evolution of society and the improvement in technologies was possible to create more sophisticated methodologies to communicate. The real breakthrough there was in the 1976 with the publication of the **Diffie-Hellman** algorithm which got us a secure way to exchange information on an insecure channel and after a few years, in the 1978 another important algorithm, **RSA**(from it's creators *Rivest, Shamir e Adleman*) was born. The publication of this algorithms helped to develop and research new area of math, contributing with the improvement of the general cryptography technique. We can divide cryptographic protocol in two big branches: **symmetric** and **asymmetric** cryptography. In the symmetric cryptography the key used to encrypt the data is also used to decrypt it, meaning that the two part need to share the same key. However with asymmetric cryptography we don't have this problem since it uses two different keys, one to encrypt and one to decrypt. Usually this is used only to exchange the key and then use symmetric cryptography, since asymmetric cryptography it's not so efficient when dealing with large quantities of data.

3.1 - Prime number generation

Prime number play an important role in cryptography, but generating those number in a deterministic way and in a small amount of time is not possible; what we can to is do generate random number and then check if they are prime with a probability test.

Theorem 1. ***Prime number theorem***

This theorem describe the asymptotic distribution of prime number and gives use an approximation of how many prime exists under a certain integer n.
Let $\pi(x)$ be the prime-counting function then

$$\lim_{x\to\infty} \frac{\pi(x)}{\frac{x}{\log(x)}} = 1$$

That can be approximated like this $\pi(x) \approx \frac{x}{\log(x)}$

Algorithms like **RSA** and **D-H** use prime number of 512 bit that, using the previous theorem, gives us around 10^{150} prime number that we can pick. Generating prime number is not easy, we could pick a random number and then try to factorize it, but with number of 512 bit this is not possible since this operation while require too much time. We need to use some probabilistic algorithms like the following one.

Lehmann Algorithm

1. Pick a random number a less then p
2. Compute $a^{\frac{p-1}{2}} \pmod p$
3. If $a^{\frac{p-1}{2}} \neq \pmod p$ or $a^{\frac{p-1}{2}} \neq -1 \pmod p$ then p is not prime
4. If $a^{\frac{p-1}{2}} \equiv 1 \pmod p$ or $a^{\frac{p-1}{2}} \equiv -1 \pmod p$ then the probability that p is not prime doesn't exceed 50%

Repeating this algorithm decrease the probability of p to not be prime.

Rabin-Miller Algorithm Let's pick a random number p. Compute b that is the number of times that 2 divide $p-1$ e let's compute m so that $p = 1+2^b m$.

1. Pick a random number a less then p
2. $j = 0$ and $z = a^m \pmod p$
3. If $z = 1$ or $z = p-1$ then p is probably prime
4. If $j > 0$ and $z = 1$ then p is not prime
5. $j = j+1$
6. If $j < b$ and $z \neq p-1$ then $z = z^2 \pmod p$ and repeat from the third point. If $z = p-1$ then p is probably prime.
7. If $j = b$ and $z \neq p-1$ then p is not prime.

This is the most used algorithm because the probability of a number to not be prime decrease faster than other algorithms, since it's about $\frac{1}{4^t}$ where t is the number of iteration.

3.2 - RSA

The security of this particular algorithm is based on the difficulty to factorize large number. There are some technique developed trying to break this algorithm, but right now nobody found a solution that can be run on polynomial time.
RSA is not so difficult to understand, there are just three main step:

1. ***Generation of the public and private key***
   - Generate two big number p and q both with the same lenght in bit.
   - Compute $n = pq$ and $\phi = (p-1)(q-1)$
   - Pick a random integer e such that $1 < e < \phi$ and $gcd(e, \phi) = 1$
   - With the extended Euclidean algorithm compute d such that $1 < d < \phi$ and $ed \equiv 1 \pmod{\phi}$
   - The *public key* is given by the pair (n, e), while the *private key* is d
2. ***Data Encryption***
   - The part interested in the communication get the public key (n, e)
   - Represented the message like a big integer $m \in [0, n-1]$
   - Compute $c = m^e \pmod{n}$
   - The encrypted message to send is c
3. ***Data Decryption***
   - The receiver, who provided the public key and who recived the encrypted text c, can get the original text use its private key d to get $m = c^d \pmod{n}$.

Demonstration

Since $ed \equiv 1 \pmod{\phi}$ exist a value k such that $ed = q + k\phi$. If $gcd(m, p) = 1$ then for Fermat's last theorem we have:

$$m^{p-1} \equiv 1 \pmod{p}$$

raising for $k(q-1)$ and multiplicand for m we get:

$$m^{1+k(p-q)(q-1)} \equiv m \pmod{p}$$

Else if $gcd(m, p) = p$ then the last congruency hold since its congruent to 0 modulo p. So we always have that:

$$m^{ed} \equiv m \pmod{p}$$
$$m^{ed} \equiv m \pmod{q}$$
$$m^{ed} \equiv m \pmod{n}$$

So it holds that:

$$c^d \equiv (m^e)^d \equiv m \pmod{n}$$

The encryption phare used in RSA can be speed up if we choose e equal to 4, 16 or 65537 since are more easy to calculate and they choice don't make weak, from a secuirty prospective, the algorithm.

3.3 - Diffie-Hellman

The first public key protocol was published in 1976 by **Whitfield Diffie** e **Martin Hellman** and its the first example of public key exchange. Usually a secure communication required a key exchange on a physical channel, but with the introduction of **D-H** it become possible to exchange a secret key on a public(insecure) channel.
Conceptually the algorithm is represented in the following image

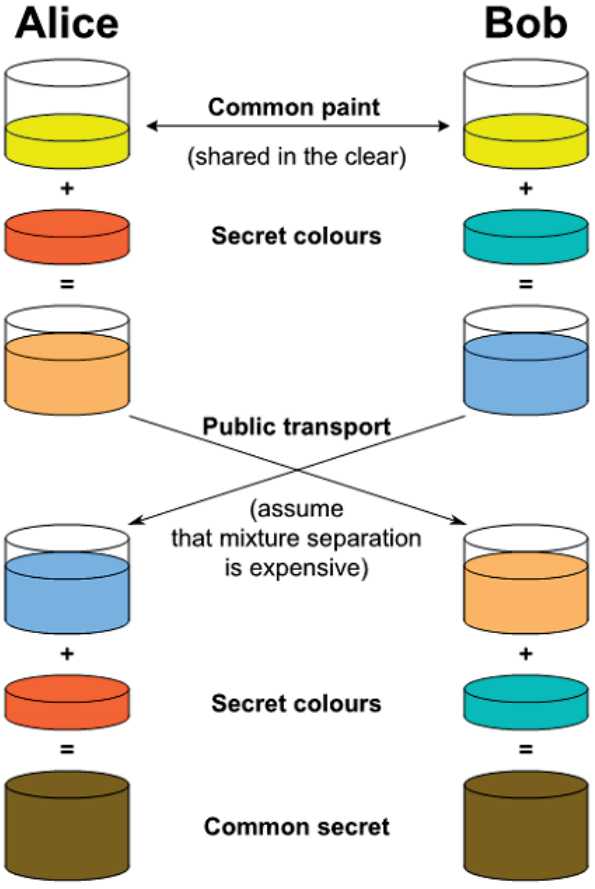

Figure 4: Diffie-Hellman key exchange

Mathematically we define **D-H** algorithm on a multiplicative group of a finete

set of prime number. The two part **Alice** (A) and **Bob**(B) choose a big prime number n and g such that g is a primitive element of G. The values n and g must be kept secret and can be shared on a public channel following this procedure:

- **A** Pick a big integer x and sends to **B** $X = g^x \pmod n$
- **B** Pick a big integer y and sends to **A** $Y = g^y \pmod n$
- **A** Compute $k_0 = Y^x \pmod n$
- **B** Compute $k_1 = X^y \pmod n$

Since $(g^x)^y = (g^y)^x = g^{xy}$ we have that $k = k_0 = k_1$. Knowing only n, g, X and Y is not possible find x or y without first solving a discrete logarithm problem.

3.4 - Man in the middle

One thing that **D-H** cannot do is to protect ourself from MITM attack.

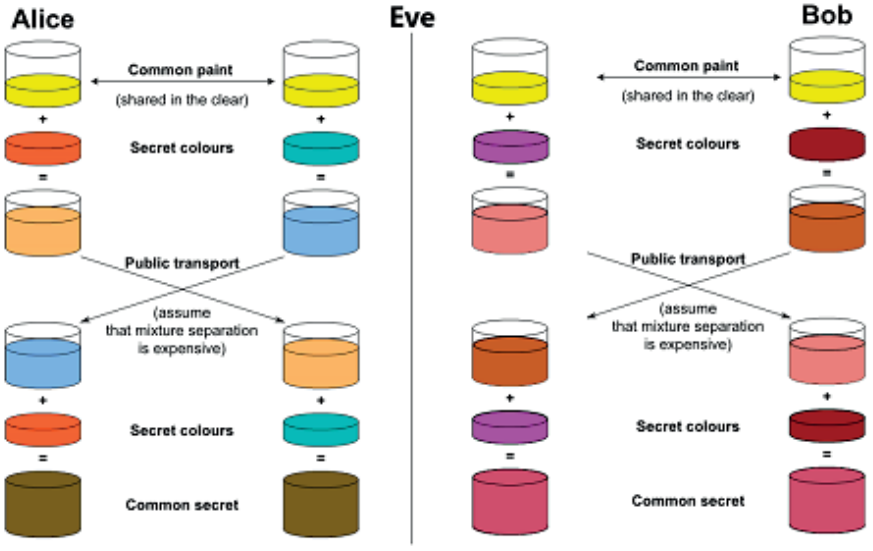

Figure 5: Diffie-Hellman MitM

Looking at the image is easy to notice that Alice cannot know if she is communicating with Eve or Bob, the only way to know the other interlocutor is to meet him physically, same thing for Bob.

3.5 - Discrete logarithm problem

The **D-H** algorithm is the application of a mathematical problem called *discrete logarithm*.

Definition 7. *Let $(G, \circ)$ be a cyclic group of order n and let g be a generator of G. Having $y = g^x = g \circ g ... \circ g$ find x.*

We basically need to calculate $x = \log_g y$ (discrete logarithm). One thing to notice is that the solution to this problem is not unique. If G is a cyclic group with g as generator then $g^x = g^z \Leftrightarrow x \equiv z \pmod{n}$. The difficulty in solving a discrete logarithm problem depends also from the chosen group and from the operation defined in it. In fact solving the **DLP** on additive group modulo n is a lot easier that solving **DLP** on multiplicative group(like in *D-H*).

4 - ELLIPTIC CURVE

A lot of public key algorithms, protocols and cryptocurrency (recently born) are based on elliptic curve that allows to have a security level comparable with the one offered by **RSA** and **D-H**, but with faster encryption and decryption time.

Definition 8. *An elliptic curve E on a field $\mathbb{K}$ written as $E/\mathbb{K}$ is given by the Weierstraβ equation:*

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

with $a_1, a_2, a_3, a_4, a_6 \in \mathbb{K}$ such that for every point (x_1, y_1) with coordinate in $\overline{\mathbb{K}}$ satisfy E and its partial derivatives $2y_1 + a_1x_1 + a_3$ and $3x_1^2 + 2a_2x_1 + a_4 - a_1y_1$ do not become null.
Another condition to satisfy is that of **non-singularity**. A point of a curve is singular if both the partial derivatives are null. More formally we can write:

Definition 9. *Let:*

$$\begin{aligned} b_2 &= a_1^2 + 4a_2 \\ b_4 &= a_1a_3 + 2a_4 \\ b_6 &= a_3^2 + 4a_6 \\ b_8 &= a_1^2a_6 - a_1a_3a_4 + 4a_2a_6 + a_2a_3^2 + a_4^2 \end{aligned}$$

Using the transformation $y \to y - (a_1x + a_3)/2$ we get an isomorphic curve

$$y^2 = x^3 + \frac{b_2}{4}x^2 + \frac{b_4}{2}x + \frac{b_6}{4}$$

whose cubic polynomial have the roots in the closure of $\overline{\mathbb{K}}$ if and only if the discriminant is not null.

The equation is useful to find out if a curve is elliptic or not.

Definition 10. *Let E a curve defined over $\mathbb{K}$. The discriminant of E denoted by Δ satisfies*

$$\Delta = -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6$$

The curve is not singular and so its elliptic if and only if Δ is not null. That assure the existence of one unique tangent for each point on the curve.

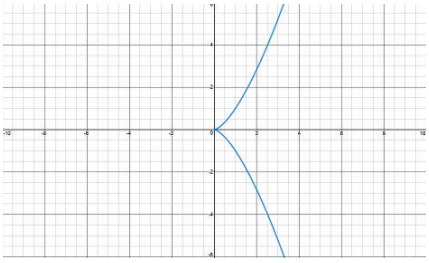

Figure 6: Curve $y^2 = x^3$ with a singular point of cusp $y^2 = x^3 - 3x + 2$

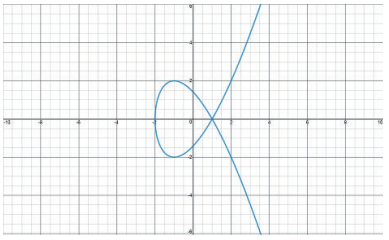

Figure 7: Curve $y^2 = x^3 - 3x + 2$ with a node(singular point)

4.1 - Group law

The set of point described by the curve together with a group operation define a group on $\mathbb{R}$. Since we are in an abelian group we can write $P + Q + R = \Theta$ like $P + Q = -R$. This allows us to derive a geometric way to sum two points $P(x_1, y_1)$ and $Q = (x_2, y_2)$. The result of the operation can be obtained by drawing a straight line that pass on the two points P and Q, the third point R is given by the intersection of the straight line with the curve. The result of the sum is the point R with its y coordinate multiplied by -1.

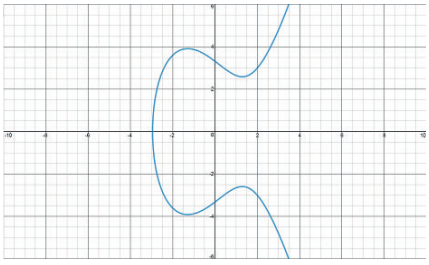

Figure 8: Curve over $\mathbb{R}$

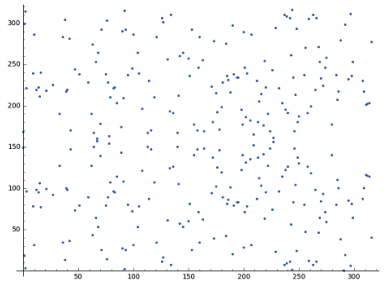

Figure 9: Curve over $\mathbb{F}$

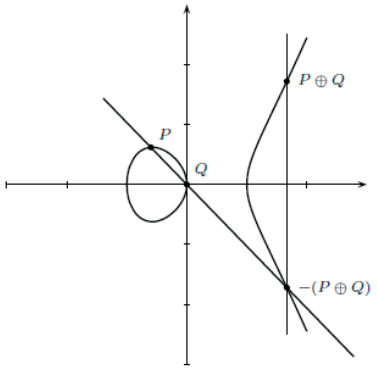

Figure 10: Sum over $\mathbb{R}$

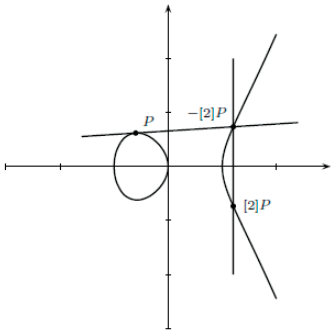

Figure 11: Doubling over $\mathbb{F}$

From the *Weierstraβ* equation we can obtain different kind of curve and each curve have a different group law which depends on the group.

Group law on a field $\mathbb{K} = \mathbb{F}_p$ with p>3

$$y^2 = x^3 + ax + b$$

1. Identity: $P + \Theta = P$ for every P on the curve
2. Opposite: Let $P = (x,y)$ then $(x,y) + (x,-y) = \Theta$. The point $(x,-y)$ is written as $-P$
3. Addition: Let $P = (x_1,y_1)$ and $Q = (x_2,y_2)$ points on the curve with $P \neq \pm Q$. Then $R = P + Q = (x_3,y_3)$ with:

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

4. Doubling: Let $P = (x_1,y_1)$ a point on the curve and $P \neq -P$. Then $R = P + P = (x_3,y_3)$ with:

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

Group law of super singular curve over $\mathbb{K} = \mathbb{F}_{2^m}$

$$y^2 + cy = x^3 + ax + b$$

1. Identity: $P+\Theta=P$ for every P on the curve
2. Opposite: Let $P=(x,y)\in E$, the point $P^{'}=(x,x+c)$ be on the curve and $P+P^{'}=\Theta$. $P^{'}$ is the opposite of P and is written as $-P$
3. Addition: Let $P=(x_1,y_1)$ and $Q=(x_2,y_2)$ points on the curve, $P\neq\pm Q$. Then $R=P+Q=(x_3,y_3)$ where:

$$x_3=\left(\frac{y_2+y_1}{x_2+x_1}\right)^2+x_1+x_2$$
$$y_3=\left(\frac{y_2+y_1}{x_2+x_1}\right)(x_1+x_3)+y_1+c$$

4. Doubling: Let $P=(x_1,y_1)$ a point on the curve and $P\neq -P$. Then $R=P+P=(x_3,y_3)$ with:

$$x_3=\left(\frac{x_1^2+a}{c}\right)^2$$
$$y_3=\left(\frac{x_1^2+a}{c}\right)(x_1+x_3)+y_1+c$$

Group law of non super singular curve over $\mathbb{K}=\mathbb{F}_{2^m}$

$$y^2+cy=x^3+ax+b$$

1. Identity: $P+\Theta=P$ for every P on the curve
2. Opposite: Let $P=(x,y)\in E$ the point $P^{'}=(x,x+c)$ is a point on the curve and $P+P^{'}=\Theta$. $P^{'}$ is the opposite of P witten as $-P$
3. Addition: Let $P=(x_1,y_1)$ and $Q=(x_2,y_2)$ points on the curve, $P\neq\pm Q$. Then $R=P+Q=(x_3,y_3)$ where:

$$x_3=\lambda^2+\lambda+x_1+x_2+a$$
$$y_3=\lambda(x_1+x_3)+x_3+y_1$$

with $\lambda=(y_1+y_2)/(x_1+x_2)$

4. Doubling: Let $P=(x_1,y_1)$ a point on the curve and $P\neq -P$. Then $R=P+P=(x_3,y_3)$ with:

$$x_3=\lambda^2+\lambda+a=x_1^2+\frac{b}{x_1^2}$$
$$y_3=x_1^2+\lambda x_3+x_3$$

where $\lambda=x_1+y_1/x_2$

Observation 1. *From the group law we can obtain another operation i.e. the scalar multiplication:*

$$n \cdot P = \underbrace{P + P + ... + P}_{n \text{ times}}$$

Looking at the formula we can notice that if n have k bit then the sum would spend about $\mathcal{O}(2^k)$, that's practically not feasible. What we can to do speed up this operation is to use a technique similar to the *Square and Multiply*.

Example 2. *Let $n = 151$, in binary becomes 10010111_2 that we can write like:*

$$151 = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

and in our case:

$$151 \cdot P = 2^7 P + 2^4 P + 2^7 P + 2^2 P + 2^1 P + 2^0 P$$

That's a lot easier to calculate and take only $\mathcal{O}(\log n)$ to be computed.

4.2 - Order of a group

In general is not easy to calculate how many points are on a **EC**, what we can to is to write and approximation of it using the following theorem:

Theorem 2. ***Hasse-Wail theorem***
Let E an elliptic curve define over $\mathbb{F}_q$. Then

$$|E(\mathbb{F}_q)| = q + 1 - t$$

with $|t| \leq 2\sqrt{q}$

4.3 - Discrete logarithm problem

Definition 11. *Let $(G, \oplus)$ a cyclic group of order p, with p prime. Let $P, Q \in G$ and let P be a generator of G.*
*The **DL** in G of Q respect to P is $n = \log_P Q$ so that:*

$$Q = n \cdot P = \underbrace{P \oplus P \oplus ... \oplus P}_{n \text{ times}}$$

We need to find n, modulo p, knowing only P and Q.

The security of a system that uses a **DLP** on elliptic curve depends also from the group operation $\oplus$ and from the choice of G.
In our case we have:

$$Q = \underbrace{P + P + ... + P}_{\text{n times}} = n \cdot P$$

4.4 - Elliptic curve over finite fields

We only talked about elliptic curve defined over $\mathbb{R}$ but real application uses only finite fields.

Definition 12. *An elliptic curve on $\mathbb{F}_p$ with $p > 3$ is the set of all the point $(x, y) \in F_p$*

$$y^2 \equiv x^3 + ax + b \pmod{p}$$

together with an imaginary point at infinity and with $a, b \in \mathbb{F}_p$ and $4a^4 + 27b^2 \neq 0 \pmod{p}$

5 - STUDY OF A CURVE

Let's take for example the curve $y^2 = x^2 - 5x + 11$ over $\mathbb{F}_{317}$. Graphically on the set of real number and in the finite set we have:

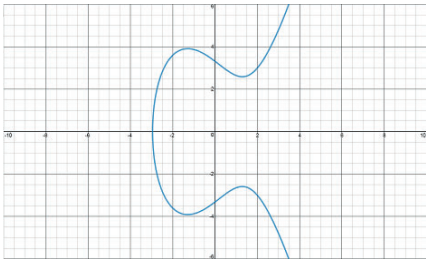

Figure 12: Curve over $\mathbb{R}$

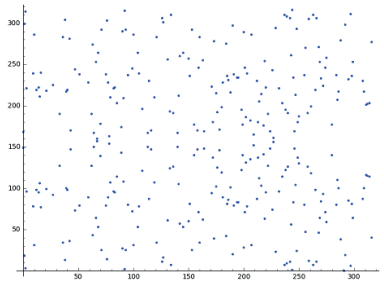

Figure 13: Curve over $\mathbb{F}$

Graphically representing the sum and the doubling we get:

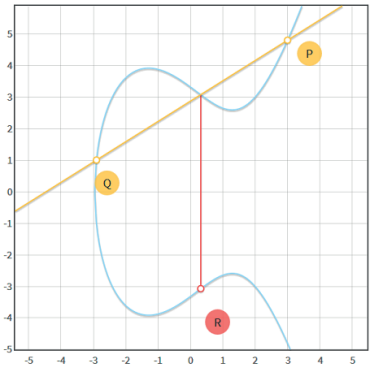

Figure 14: Sum over $\mathbb{R}$

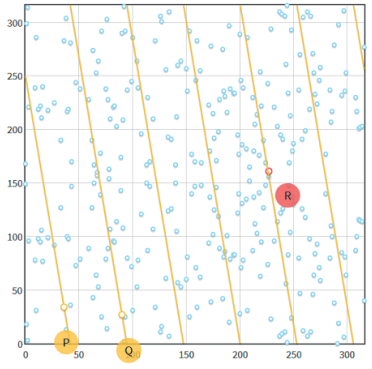

Figure 15: Sum over $\mathbb{F}$

Applying the formulas defined in the group law we get:

- Real field:

$$
\begin{aligned}
&P = (3, 4.7958) \\
&Q = (-2.9054, 1) \\
&R = P + Q = (x_3, y_3) \\
&x_3 = \left(\frac{4.7958 - 1}{2.9054 - 3}\right)^2 - 3 + 2.9054 = 0.3186 \\
&y_3 = \left(\frac{1 - 4.7958}{-2.9054 - 3}\right)(3 - 0.3186) - 4.7958 = -3.0723 \\
&R = (0.3186, -3.0723)
\end{aligned}
$$

- Finite field:

$$
\begin{aligned}
&P = (36, 34) \\
&Q = (90, 27) \\
&R = P + Q = (x_3, y_3) \\
&x_3 = \left(\frac{34 - 27}{90 - 36}\right)^2 - 36 - 90 = 227 \pmod{317} \\
&y_3 = \left(\frac{27 - 34}{90 - 36}\right)(36 - 227) - 34 = 161 \pmod{317} \\
&R = (227, 161)
\end{aligned}
$$

For the doubling:

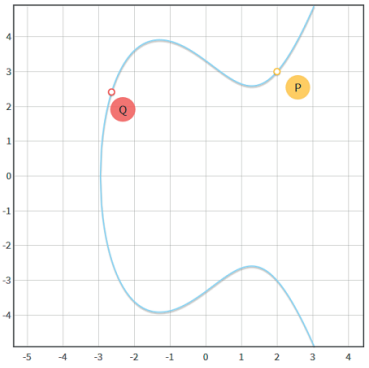

Figure 16: Doubling over $\mathbb{R}$

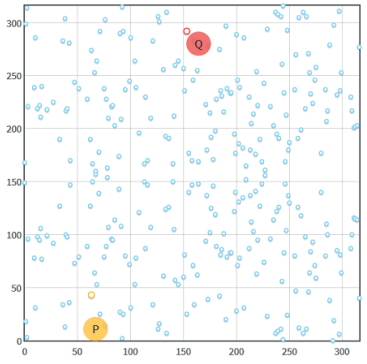

Figure 17: Doubling over $\mathbb{F}$

- Real field:

$$
\begin{aligned}
&P = (2,3)\\
&R = 2 \cdot P = (x_2, y_2)\\
&x_2 = \left(\frac{3 \cdot 2^2 - 5}{2 \cdot 3}\right)^2 - 2 \cdot 2 = -2.6388\\
&y_2 = \left(\frac{3 \cdot 2^2 - 5}{2 \cdot 3}\right)(2 + 2.6388) - 3 = 2.4120\\
&R = (-2.6388, 2.4120)
\end{aligned}
$$

- Finite field:

$$
\begin{aligned}
&P = (63,43)\\
&R = 2 \cdot P = (x_2, y_2)\\
&x_2 = \left(\frac{3 \cdot 63^2 - 5}{2 \cdot 43}\right)^2 - 2 \cdot 63 = 153 \pmod{317}\\
&y_2 = \left(\frac{3 \cdot 63^2 - 5}{2 \cdot 43}\right)(63 - 153) - 43 = 292 \pmod{317}\\
&R = (153, 292)
\end{aligned}
$$

To estimate the order of the curve we use the *Hasse-Weil theorem* and we get that :

$$282 \leq |E(\mathbb{F}_q)| \leq 353$$

Calculating the exact order of a curve is not an easy task, to get a precise result we could calculate and the count every point in the group, but this is not feasible we using group with a big prime number.
In our case since p is very small this approach can be used and after calculating and counting every point we get 312.

6 - ELLIPTIC CURVE APPLICATION IN CRYPTOGRAPHY

A lot of the algorithms defined on finite field can be readapted to be used on group build upon elliptic curve. In the following section we will see two application of it, just to have an idea and to see how much they differ from they corresponding defined only on finite filed.

6.1 - Diffie-Hellman

Compared to the original algorithm, the basic principle is the same we only need to change the group and its operations.
Let's considere the curve $y^2 \equiv x^3 + ax + b \pmod{p}$ with $p > 3$. Alice (**A**) and Bob (**B**) choose a primitive elements P of G then:

- **A** and **B** generate their private key d_A and d_B

- **A** and **B** generate their public key $Q_A = d_A P$ and $Q_B = d_B P$, with $P \in G$ public
- **A** and **B** exchange Q_A and Q_B
- **A** compute $S = n_A Q_B$ and **B** compute $S = n_B Q_A$

Since P is know by multiplying with they private key the message of the other they get a common key to use to encrypt data and after that they can start to communicate.

6.2 - ECDSA

A digital signature is a mathematical scheme used to verify the authenticity of a message. It guarantees to the receiver that the sender is authentic and that the message didn't change on the communication channel.
There are numerous **DSA** (*Digital Signature Algorithm*) based, for example, on elliptic curve, but the overall procedure is the same.

Algorithm

Let's suppose that Alice wants to send a signed message to Bob, the first thing that they have to do is decide the parameters of the curve, (C, G, n) where:

- C is the equation of the elliptic curve and it include the field on which is defined
- G is a generatore of the curve with order n
- n is an integer that's the order of G and that verify: $nG = 0$

Then Alice generate a private key $d_A \in [1, n-1]$ and a public key $Q_A = d_A \cdot G$. Since the parameters Q_A and G are public in theory it's possible to find the secret key d_A and so impersonate Alice.
In practice this is not possible, since finding d_A means solving a *discrete logarithm problem on elliptic curve*. In the following chapter we will see some common algorithm that try to solve the **DLP** and they execution time.

6.3 - Signature

Alice can sign a massage m by following this steps:

1. Compute $e = HASH(m)$ with an hashing function (like SHA)
2. Let z composed by the L_n leftmost bit of e where L_n is the length in bit of the order of the group n.
3. Pick a random integer $k \in [1, n-1]$
4. Compute the point $(x_1, y_1) = k \cdot G$
5. Compute $r = x_1 \pmod n$. If $r = 0$ go back to 3
6. Compute $s = k^{-1}(z + rd_A) \pmod n$. If $s = 0$ go back to 3

7. The digital signature is the pair (r,s)

The parameter k doesn't have to be private, but it's important that it's different every time that we sign a message else we can compute d_A with the information of two signed message.

Verification of the signature
To verify the signature of Alice, Bob must have the point Q_A then checks if the point is on the curve:

- Check that Q_A is not the identity
- Check that Q_A is on the curve
- Check that $n \cdot Q_A = 0$

Next he verify the digital signature.

1. Checks that r and s are both integer in $[1, n-1]$. If this is not the case then the signature is not valid.
2. Compute $e = HASH(m)$ where the hashing function is the same used for the signature
3. Compute $w = s^{-1} \pmod{n}$
4. Compute $u_1 = zw \pmod{n}$ and $u_2 = rw \pmod{n}$
5. Compute a point on the curve $(x_1, y_1) = u_1 \cdot G + u_2 \cdot Q_A$. If $(x_1, y_1) = \Theta$ then the signature is not valid
6. The signature is valid if $r \equiv x_1 \pmod{n}$ else is not valid

It's easy to verify that the algorithm gives the correct result.

$$\begin{aligned} C &= u_1 \cdot G + u_2 \cdot G = (u_1 + u_2 d_A) \cdot G \\ &= \left(zs^{-1} + rd_A s^{-1}\right) \cdot G = (z + rd_A)\, s^{-1} \cdot G \\ &= (z + rd_A)\,(z + rd_A)^{-1} \left(k^{-1}\right)^{-1} \cdot G \\ &= k \cdot G \end{aligned}$$

7 - COMPUTING THE DISCRETE LOGARITHM

The best algorithms to solve discrete logarithm problem on cyclic group have a complexity of $\mathcal{O}(\sqrt{n})$ and they are called *square root algorithms*. In general those algorithm use the birthday paradox as main idea, trying to exploit the collision during the execution. The most famous are: *giant-step, baby-step* and *Pollard-*ρ.

7.1 - Baby-step, giant-step

This algorithm use as main idea the following remark:

Observation 2. *Let x a positive integer. We can write x like :*

$$x = am + b$$

with a, m, b *integer*

We can define the DLP on elliptic curve such as:

$$\begin{aligned} Q &= xP \\ Q &= (am+b)P \\ Q &= amP + bP \\ Q - amP &= bP \end{aligned}$$

The steps that we need to follow to execute our algorithm are:

- Compute $m = \sqrt{n}$
- For each b in $0...m$ compute bP and save the result into an hash table
- For each a in $0, ..., m$ compute:
  - amP
  - $Q - amP$
  - Check if in the hash table exist a point such that $Q - amP = bP$
  - If it exist then we found $x = am + b$

The points bP are the baby step (computed with small increments), while in the second part of the algorithm we calculate the giant step (computed with big increments) A more direct way to explain the overall algorithm is the following. Let's take the equation $Q = amP + bP$ and consider the case:

- When $a = 0$ we check that Q is equal to bP with b in $0, ..., m$. So we are comparing Q with the points ranging from $0P$ to mP
- When $a = 1$ we check that Q is equal to $mP + bP$. So we are comparing Q with all the points ranging from mP to $2mP$
- When $a = 2$ we are comparing Q with all the points ranging from $2mP$ to $3mP$
- ...
- When $a = m - 1$ we are comparing Q with all the points ranging from $(m-1)mP$ to $m^2P = nP$

7.2 - Pollard-ρ

This algorithm use as main idea the birthday paradox.

Definition 13. ***Birthday paradox***
Given a set of n random people we need to find a pair of person with the same birthday. For the pigeonhole principle we have a probability of 100% *if n is 367, but we can get to* 99.9% *only with 70 people.*

This algorithm have the same time complexity of baby-step giant-step, but it doesn't require large amount of space since its spatial complexity is negligible. We divide a group G in three subset S_1, S_2, S_3 of approximately the same order. Let's also suppose that $1 \notin S_2$ and define the sequence $x_0, x_1, ..., x_h, ...$ with $x_0 = 1$ and

$$x_{i+1} = f(x_i) = \begin{cases} \beta \cdot x_i, & \text{if } x_i \in S_1 \\ x_i^2, & \text{if } x_i \in S_2 \\ \alpha \cdot x_i, & \text{if } x_i \in S_3 \end{cases}$$

with $i \geq 0$, $ord(G) = n$, α generator of G and β in G. This sequence, in turn, defines another two sequence $a_0, a_1, \ldots$ and $b_0, b_1, \ldots$ that fulfill $x_i = \alpha^{a_i}\beta^{b_i}$ for $i \geq 0$. Then defining $a_0 = 0, b_0 = 0$ for $i \geq 0$ we get:

$$a_{i+1} = \begin{cases} a_i, & \text{if } x_i \in S_1 \\ 2a_i \pmod n, & \text{if } x_i \in S_2 \\ a_i + 1 \pmod n, & \text{if } x_i \in S_3 \end{cases}$$

$$b_{i+1} = \begin{cases} b_i + 1 \pmod n, & \text{if } x_i \in S_1 \\ 2b_i \pmod n, & \text{if } x_i \in S_2 \\ b_i, & \text{if } x_i \in S_3 \end{cases}$$

We can now use the **Floyd's algorithm** to find two elements on the group, x_i and x_{2i}, such that $x_i = x_{2i}$. So we obtain $\alpha^{a_i}\beta^{b_i} = \alpha^{a_{2i}}\beta^{b_{2i}}$ that we can write as $\beta^{b_i - b_{2i}} = \alpha^{a_2 i - a_i}$. After doing the logarithm in base α on both side, we get:

$$(b_i - b_{2i}) \cdot \log_\alpha \beta \equiv (a_{2i} - a_i) \pmod n$$

Considering $b_i \not\equiv b_{2i} \pmod n$ (the probability that they are equal is very low so we can neglect it).

On elliptic curve the problem is the same. We need to find an x that satisfy $Q = xP$, to do so we use some integer a, b, A, B such that $aP + bQ = AP + BQ$. In short we have:

$$\begin{aligned} aP + bQ &= AP + BQ \\ aP + bxP &= AP + BxP \\ (a - A)P &= (B - b)xP \end{aligned}$$

Then:

$$a - A \equiv (B - b)x \pmod{n}$$
$$x = (a - A)(B - b)^{-1} \pmod{n}$$

where n is the order of the cyclic group G that we are using.
There are different versions of this algorithm but the overall behavior is the same.

7.3 - Baby-step giant-step vs Pollard-ρ vs bruteforce

In this final section we report some execution times of the previous algorithms run on PC for some quite small groups:

Example 3. *Low order curve*
Curve order: 10331
Using bruteforce
Computing all logarithms: 100.00% done
Took 2m 31s (5193 steps on average)
Using babygiantstep
Computing all logarithms: 100.00% done
Took 0m 6s (152 steps on average)
Using pollardsrho
Computing all logarithms: 100.00% done
Took 0m 21s (138 steps on average)

Example 4. *High order curve*
Curve order: 123779
Using bruteforce
Computing all logarithms: 100.00% done
Took 5h 51m 31s (61866 steps on average)
Using babygiantstep
Computing all logarithms: 100.00% done
Took 3m 56s (527 steps on average)
Using pollardsrho
Computing all logarithms: 100.00% done
Took 14m 11s (481 steps on average)

The worst algorithm is the brute force one, like we expected and we can notice that Baby-step Giant-Step is three times faster then Pollard-ρ.
To understand this result we must remind that the advantage of Pollard-ρ over *BS-GS* is the requirement in terms of memory. One require a huge amount of memory(which depends on the order of the group) the other require a negligible amount of memory and so can be used with larger group.
The last thing to notice is the number of steps that each algorithm took to run; we can see that the square root methods needed about half of the step of the brute force, in according with the theory.

8 - REFERENCES

Bernstein D.J., Buchmann J., Dahmen E. (2009) *Post-Quantum Cryptography*. Springer, pp.1-32

Blackburn S. R., Cid C., Mullan C. (2010) *Group Theory in Cryptography*. Department of Mathematics, Royal Holloway, University of London Egham, Surrey TW20 0EX, United Kingdom, pp. 2-6

Cohen H., Frey G., Avanzi R., Doche C., Lange T., Nguyen K., Vercauteren F. (2005) *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman and Hall/CRC, pp. 19-35, 268-285, 591-607

Corbellini A. (2015) Elliptic Curve Cryptography: A gentle introduction *http://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction*

Hankerson D., Menezes A., Vanstone S.A. (2004) *Guide to Elliptic Curve Cryptography*, Springer-Verlag, pp. 76-83

Koblitz N. (1998), *Algebraic Aspects of Cryptography*. Springer(Corrected Second Printing 1999),pp. 18-21, 133-136

McEliece R. J. (1979) *Finite Fields for Computer Scientists and Engineers*. Springer, pp. 3-28

Mermin N. D. (March 28, 2006). *Breaking RSA Encryption with a Quantum Computer: Shorś Factoring Algorithm*, Cornell University, Physics 481-681 Lecture Notes

Schneier B. (1996), *Applied Cryptography*. John Wiley & Sons, pp. 238-241, 233-263

Shor P.W. (1994) *Algorithms for quantum computation: Discrete logarithms and factoring*, Proc. 35th Annu. Symp. Foundations of Computer Science, pp. 124-134.

Silverman J. H. (1986). *The Arithmetic of Elliptic Curves. Graduate Texts in Mathematics*, Springer-Verlag, pp. 137-139