A new probabilistic public key algorithm based on elliptic logarithms

Afonso Comba de Araujo Neto, Raul Fernando Weber

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

afonso@cpd.ufrgs.br, weber@inf.ufrgs.br

Abstract. *This paper introduces a new probabilistic public key algorithm based on elliptic curves and show that it is secure. The security of the scheme is solely based on the difficulty of the* elliptic curve discrete logarithm problem *while at the same time it has a constant message expansion for one encryption of a plaintext of any practical size. In the alternative algorithms, like Cramer-Shoup and PSEC, for a large plaintext, either the message expansion is proportional to its size or an additional security assumption is needed. Although some restrictions are posed on the public part of the key, we show how to easily find the needed parameters, and also suggest ways to make the public key as small as possible.*

1. Introduction

The concept of probabilistic encryption was first introduced in 1984 by Goldwasser and Micali [5]. The main idea behind probabilistic encryption is to obtain *semantic security*. In essence, for a semantically secure encryption, no polynomially bounded adversary can obtain any partial information about the plaintext from the ciphertext.

The fundamental property of a probabilistic algorithm is that, for any given plaintext, there is a huge number of possible ciphertexts. For example, suppose someone wants to send one bit using a deterministic public key algorithm. The problem is that the message space is small enough so that an attacker can encrypt all the possibilities, in this case the bits 1 and 0, and just compare the ciphertext that was sent. Under probabilistic encryption, there are enough different ciphertexts for both 1 an 0, and that makes it unfeasible for an attacker to use that strategy. And even though the algorithm has this property, all ciphertexts are uniquely decipherable. Between probabilistic encryption and deterministic encryption, given the same security and efficiency requisites, the former would be preferable.

It is widely known that one major topic of research on the field of cryptography today is the use of elliptic curves for public key cryptography. The points of a carefully chosen elliptic curve over a finite field provide a group structure where the discrete logarithm over it has no known sub-exponential algorithm. That fact allows much smaller fields and keys than other approaches, such as those based on the difficulty of the discrete logarithm over the multiplicative group of a finite field or those based on the difficulty of factoring, both which have sub-exponential algorithms. The problem of calculating logarithms over elliptic curves is often called Elliptic Curve Discrete Logarithm Problem, ECDLP for short. More information about the basics of elliptic curve applications in cryptography can be found in [1].

Our objective is to propose a new efficient probabilistic public key encryption algorithm over the smaller fields allowed by elliptic logarithms. There exists at least two other probabilistic schemes based on elliptic curves. The first one is named PSEC (Provably Secure Elliptic Curve Encryption Scheme), which was a submission to the standard IEEE P1363 [8], and the second one is the Cramer-Shoup public key scheme [4], which was designed for arbitrary groups and therefore can be used with elliptic curves. Both schemes, however, suffer from a fundamental disadvantage when used over a large plaintext: either the user encodes several small parts of it as points of an elliptic curve and encrypt them separately or he encrypts everything at once using a random secret key and a block cipher and use the scheme to encrypt only the key. The problem is that the first choice incurs in a huge message expansion (and requires a proportional number of truly random bits) and the second one adds another security assumption to the scheme: now both the elliptic curve problem and the block cipher must be unconditionally secure.

The scheme proposed in this paper offers a third choice of encryption which demands only one security assumption and has a constant message expansion for any plaintext. The assumption is that the Elliptic Curve Diffie-Hellman Problem (ECDHP) over group of points of a carefully chosen elliptic curve is hard on average against all polynomially bounded adversaries. The ECDHP is conjectured to be equivalent to the ECDLP, and will be explained in section 7. In the next section we'll introduce briefly the algorithm proposed and explain how the paper is structured.

2. Overview of the algorithm

Much like the Blum-Goldwasser probabilistic scheme [2], which relies on the pseudorandom bit generator Blum-Blum-Schub, our scheme relies on the elliptic curve based pseudorandom bit generator by Burton Kaliski [6], which will be explained in detail later in section 5. Here's a summary of the full algorithm (logarithms are always taken base 2):

1. `Public parameters`. Find a pair of elliptic curves, E and its twist E^t, over a finite field $\mathbb{F}_p$ (with characteristic greater than 3) where the number of points of both curves is $prime$. In section 3 we show how to quickly find such pair, using the theory of Complex Multiplication. Our method has the useful advantage that a single integer with about $\frac{\log p}{4}$ bits is enough to uniquely identify the prime field, the coefficients of both curves (up to $\mathbb{F}_p$-isomorphism), both orders and to easily find base points which are generators of both curves.
2. `Key pair`. Consider n_E and G_E the order and generator of the first curve and n_{E^t} and G_{E^t} the order and generator of the twist. Choose uniformly secret keys $0 < s_E < n_E$ and $0 < s_{E^t} < n_{E^t}$ and calculate the public key using scalar multiplication over the curves, that is the points $P_E = s_E G_E$ and $P_{E^t} = s_{E^t} G_{E^t}$.
3. `Encryption`. Select uniformly a secret, discardable seed $a \in [0, 2p+1]$. As a preliminary step, using the curves, generators, orders and public keys, the algorithm computes 3 points, namely M, T_E and T_{E^t} and also a scalar value $r \in [0, 2p+1]$. Using r as a seed, and the points T_E and T_{E^t} as generators of the curves E and E^t, run the Kaliski pseudorandom bit generator, creating a keystream of the size of the plaintext. The ciphertext is created XOR'ing the keystream and the plaintext. Send the ciphertext along with the point M. Using point compression on M, the size of the complete ciphertext is the size of the plaintext plus $\lceil \log p \rceil + 1$ bits.

4. `Decryption`. Having M, the secret keys s_E and s_{E^t} and the ciphertext, the objective is to reconstruct the keystream and recover the plaintext by XOR'ring the ciphertext with it. The algorithm, therefore, using the point M and the secret keys recalculates the points T_E, T_{E^t} and the value r, allowing the inverse procedure of the encryption.

The fact that the ciphertext depends both on the plaintext and also on the initial seed provided for encryption implies that, for the same plaintext, an exponential number of ciphertexts are possible. That's the property that makes this a probabilistic public key encryption algorithm, instead of a deterministic one.

One interesting advantage comes from the restriction that both curves must have prime orders. As we'll show later, that implies that any point has order of same magnitude as the order of the finite field itself. In the end it means that when you select the bit length of the characteristic of the finite field, you are automatically determining the security level of the elliptic logarithms.

It is important to notice that the necessity of a prime order twisted pair as a public key is a strong restriction. Although it's not that hard to find one curve with a prime number of points, it's not obvious how to efficiently find a prime order elliptic curve such that its twist also has prime order. For the algorithm to be usable, it is important to show that it is easy to find such pairs and that there are enough of them. Our method also allows a considerable compression of the public parameters, which uncompressed are clearly a lot of information. Section 3 explains how to find such pairs and section 4 explains the compression techniques.

The algorithm itself is pretty straightforward. Given a public key and a seed, a point is generated such that, along with the private key, parameters for the Kaliski pseudorandom bit generator are calculated and a keystream is created. Kaliski showed that this pseudorandom bit generator is cryptographically secure under the assumption that the ECDLP is hard. That means that its period is at least superexponential in p (p being the characteristic of the finite field where the curves are defined) and it's forward and backward unpredictable against all polynomially bounded adversaries. The encryption with this keystream is, therefore, equivalent to a One-Time-Pad against all polynomial attacks, if the seed is never disclosed or repeated. Section 5 presents a comprehensive explanation of the Kaliski generator, which will be necessary for understanding the details of the full algorithm described in section 6.

3. Generating prime order twisted pairs

An elliptic curve over a finite field $\mathbb{F}_p$ with characteristic greater than 3 can be defined by an equation with variables x and y, that has the form

$$y^2 = x^3 + ax + b, \tag{1}$$

with $a, b \in \mathbb{F}_p$ and having $4a^3 + 27b^2 \not\equiv 0 \mod p$.

Given any elliptic curve, we call a *quadratic twist* (hereafter only called *twist*) of this curve another curve with coefficients related in the following way. Take any quadratic

non-residue β modulo p. The twist of the curve with coefficients a and b is the curve with coefficients $a^t = a\beta^2$ and $b^t = b\beta^3$. In other words, if equation 2 is an elliptic curve, then,

$$y^2 = x^3 + a\beta^2 x + \beta^3 b, \tag{2}$$

is its twist, for any quadratic non-residue β. Although they have different structure over $\mathbb{F}_p$, an elliptic curve and its twist are isomorphic over the extension field $\mathbb{F}_{p^2}$.

By a famous theorem due to Helmut Hasse, it is known that the number of points $\#E$ of any elliptic curve over a finite field $\mathbb{F}_p$ satisfies the following equation

$$\#E = p + 1 - t, \tag{3}$$

where $|t| \leq 2\sqrt{p}$. The value t is known as the *trace of Frobenius at p* of the curve. An interesting and useful fact is that if the number of points of a curve is $\#E = p + 1 - t$, then the number of points of its twist is exactly $\#E^t = p + 1 + t$. That means that the calculation of the number of points of any curve automatically gives the number of points of the twist.

In order to find the prime order twisted pair, we need to find an elliptic curve over some prime field $\mathbb{F}_p$ with trace of Frobenius t such that $p + 1 + t$ and $p + 1 - t$ are both prime. The problem is that the determination of the trace of an arbitrary curve is a really hard problem. By inspection of equation 3, it becomes clear that the determination of t and the order of the curve are actually the same problem, as they are related by a linear equation. There is a polynomial time algorithm for counting the number of points of a curve, known as Schoof-Elkies-Atkin algorithm. A naive approach would be to randomly choose a prime p and the coefficients a and b, count the number of points of the curve, and, if it's prime, check if $\#E^t = 2p + 2 - \#E$ is also prime. Although that method works, it is in fact too slow for use in practice.

To solve the problem efficiently, we must turn it upside down, using the theory of Complex Multiplication of elliptic curves. Even though the determination of the trace of Frobenius of a given curve is a hard problem, the creation of a curve with a specific trace is a lot easier.

The theory of Complex Multiplication is deep, so we'll only provide the absolute necessary to understand the ideas being presented. We urge the reader to turn to [3] for a comprehensive explanation of the theory. Take $-D$ as a negative integer congruent to 0 or 1 modulo 4 and squarefree, that is, it's not divisible by any squares. Then $-D$ is a fundamental discriminant of an imaginary quadratic field $\mathbb{Q}(\sqrt{-D})$.

We need to solve the following equation:

$$t^2 = 4p - Dy^2 \tag{4}$$

The value p will be the characteristic of a finite field, so it must be prime. Given p, t will be the trace of Frobenius of a curve defined over $\mathbb{F}_p$. The variable y can be any integer. We'll explain how to calculate the curve coefficients later.

Consider D as a constant for the moment. Our approach to solve this equation is to use the idea of families of curves. That means, rearrange the equation so that the right

part of it depends only on one variable, and that for any given integer it generates possible primes and traces with our necessary needs. It essentially becomes a polynomial in one variable. The equation won't produce all possible curves, but will produce a family of related curves.

We choose to fix $y = 1$. That automatically gives one less variable to work with. Second, we need odd traces. By inspection of equation 3 we can see that even traces won't ever produce prime number of points. So replace t by

$$t(x) = 2x^2 - 2x + 1, \tag{5}$$

which is odd for any x. After rearranging the terms and making the substitutions, the equation now becomes

$$\begin{aligned} p(x) &= \frac{(2x^2 - 2x + 1)^2 + D}{4} && (6) \\ &= x^4 - 2x^3 + 2x^2 - x + \frac{1 + D}{4} && (7) \end{aligned}$$

Notice that, after the rearrangements, D becomes positive. For the independent term, we need $\frac{1+D}{4}$ integer. That will happen for all $|D| \equiv 3 \mod 4$. But one more constraint is needed. We want p to be prime, and that will never happen if the constant term of the equation is even. That is solved by having $|D| \equiv 3 \mod 8$.

To actually find the curves, we run a sequential search replacing x by integers, positive or negative, and checking if $p(x)$ is prime using some probabilistic primality algorithm like a Miller-Rabin test. When a p prime is found, we set $t = 2x^2 - 2x + 1$ and check the orders of the curves. Notice that if you want primes with about $\log p$ bits, we must use integers with about $\frac{\log p}{4}$ bits.

On a 1.3GHz Athlon, using $-D = -163$ and looking for primes with 160 bits, this search produces a prime order twisted pair in about fifteen seconds. And more improvements can be made. For example, take $-D \in \{-11, -19, -43, -67, -163\}$ and for each integer calculate a base value $p(x) = x^4 - 2x^3 + 2x^2 - x$. Now, for each base value, we can add five different constants generating five different prime candidates, which will be faster than moving to the next value of x.

Notice that, by equation 3, the order of both curves are of the same magnitude as the finite field, give or take a factor of $2\sqrt{p}$. The fact that both groups have prime orders implies they are cyclic, and all points are generators (a famous group theory theorem by Joseph Lagrange). That means that the security level of the discrete logarithm in that group, measured as bit length, is the same as the order of the finite field. Therefore, choosing the bit length of p is the same as choosing the security level of the system.

3.1. Calculating the curve parameters

At this point we can assume to have a prime p, and a trace t which defines a prime order twisted pair. With these numbers, and using the value of D that allowed to find them,

$-D$	$H_D(x)$
-11	$x + 32768$
-19	$x + 884736$
-43	$x + 884736000$
-67	$x + 147197952000$
-163	$x + 262537412640768000$

Table 1. Hilbert polynomials for some discriminants $-D$.

we can construct the curves. To do that, we should get back to the theory of complex multiplication.

Jumping ahead on the theory, D is a number that defines uniquely what is called a Hilbert polynomial $H_D(x)$. The degree of this polynomial is the *class number* of the imaginary quadratic order with discriminant $-D$. For some values of $-D$ this polynomial has degree one, but for higher values it quickly escalates, not only in degree, but also in the size of its coefficients. For example, the discriminants -11, -19, -43, -67, -163 all have class number one and are more or less simple. On the other hand, -4195587 has class number 328, and the coefficients of $H_{-4195587}(x)$, in ASCII representation, sum up to 130 kilobytes.

The calculation in real time of $H_D(x)$ for a given D is not easy, and probably should be avoided. Therefore we suggest to define in advance the values of D you want to use and precompute these polynomials. In table 1 we present $H_D(x)$ for five values of D having class number one, all with $|D| \equiv 3 \mod 8$, which makes them suitable for our techniques.

Another concept that must be introduced is that of the *j-invariant* of an elliptic curve. The j-invariant of a curve is a number that is equal to all curves that are isomorphic over the algebraic closure $\overline{\mathbb{F}_p}$. So, as twisted pairs are isomorphic over $\mathbb{F}_{p^2}$, they have the same j-invariant. It turns out that the j-invariant of the curve and its twist, with trace of Frobenius respectively t and $-t$, is the root, modulo p, of the Hilbert polynomial $H_D(x)$ of the value of D that gave the respective traces on equation 4. Given a j-invariant j and a quadratic non-residue β, the curves coefficients are computed in two steps.

$$c = \frac{j}{1728 - j} \mod p \tag{8}$$

$$E : y^2 = x^3 + 3cx + 2c \mod p \tag{9}$$

$$E^t : y^2 = x^3 + 3c\beta^2 x + 2c\beta^3 \mod p \tag{10}$$

Finding roots of polynomials can be costly, but for polynomials with degree one it's just one subtraction.

3.2. Full example of the algorithm

For simplicity take $-D = -43$ and we'll restrict our prime search to primes $p \equiv 3 \mod 4$, which gives us some nice properties, for instance, $p-1$ is a quadratic non-residue modulo p, $(p-1)^2 \equiv 1 \mod p$ and $(p-1)^3 \equiv p-1 \mod p$. Also, for any $a \in \mathbb{F}_p$,

$a(p-1) \equiv p - a \mod p$. Half the primes are on that form, so that won't be a real problem.

The trace is always $t(x) = 2x^2 - 2x + 1$. For $|D| = 43$ the equation 7 becomes

$$p(x) = x^4 - 2x^3 + 2x^2 - x + 11 \tag{11}$$

A quick search find $x = 332$ with $p = 12076361567$ prime. The trace is $t = 219785$, which makes $p + 1 + t = 12076581353$ and $p + 1 - t = 12076141783$ both prime. We found our curves. Using the Hilbert polynomial for $-D = -43$ we have $j = p - 884736000 \mod 12076361567$, and so $j = 11191625567$ is the j-invariant. The constant c of equation 8 is

$$c = \frac{11191625567}{1728 - 11191625567} \mod 12076361567 \tag{12}$$

$$c = 6691706436 \tag{13}$$

And here is the prime order twisted pair:

$$E : y^2 = x^3 + 7998757741x + 1307051305 \tag{14}$$

$$E^t : y^2 = x^3 + 7998757741x + 10769310262 \tag{15}$$

On this example, p has 34 bits, which is considered weak. But in the very same way, $x = 1099511695761$ and $-D = -43$ give a prime order twisted pair with p having about 160 bits which is considered secure.

3.3. Brief analysis of the generation process

One could ask about the security implication of three fundamental choices made on the algorithm proposed in the last section. We now consider and discuss each one.

The first thing that could be questioned is about the D value used. We are in fact suggesting the use of D values with class number one, even though nothing rules out using higher class numbers. The fact is that, although there is no proof of that, many researchers think that someday it will be possible to mount attacks over curves defined using D with a low class number.

The real question is, what a D with a high class number is protecting exactly? Clearly, the polynomial $H_D(x)$ is a lot more complex, but it's still easily constructible from any practical value of D, otherwise we couldn't find the curve parameters. Other fact is that this polynomial has a lot of other roots, but these other roots just define isomorphic curves over the field, which are easier to find using other techniques. Another possibility is that it would probably be harder to find the specific D of a given curve, which would allow to find all values of equation 4, so it could be used somehow. But the algorithm uses D as a public parameter anyway (they are pre-computed), so that is given. And probably

no implementation of curve construction using CM today is worried about hiding the D value used. All these arguments carry out to the conclusion that we shouldn't avoid the benefits of using a low class number just because of that general feeling. Any attack that distinguishes CM curves over random curves would probably be polynomially extendable to all class numbers at the limit of the method anyway. In that case, the CM method itself would have to be avoided, not just some curves.

The second aspect one could wonder about the generation process is about fixing the value $y = 1$ in equation 4. By similar arguments to the ones just used, that would be a fundamental security disadvantage if and only if the CM method itself was at risk. What could be argued is that if you use more than one value for y you would get more candidate primes. The problem is that varying y would nullify the property that $|D| \equiv 3 \mod 8$ always gives odd values in equation 7, so that isn't an obvious conclusion. Anyway, it would be interesting to investigate different values for y. What should hold is that y is fixed.

The last point that could be discussed is the choice for the polynomial equation of the trace, as in equation 5. The only real restriction it must obey is to generate odd numbers and have 1 as the independent term. Clearly one could have chose a cubic equation, like $2x^3 - 2x^2 - 2x + 1$ or a linear one like $2x + 1$. In practice, the choice of this polynomial is a compromise between compression (explained in the following section) and the number of curves you skip when increasing the x value of the trial to find the curves. Using the cubic, the parameters can represented by a value with $\frac{\log p}{6}$ bits, but, in return, it would be slower to find a twisted pair (you'll skip more curves). On the other hand, with the linear equation you would need $\frac{\log p}{2}$ bits to represent that value, with the benefit of finding the curves a little faster. As both properties are useful in different stages of the algorithm, we conclude that a compromise of both options would be better, choosing a quadratic polynomial.

4. Public parameters compression

If the process of finding the curves was understood, it's easy to see that the value x used to calculate the prime field in equation 7, along with D, deterministically define a lot of information. Assuming that $p \equiv 3 \mod 4$ (and $p-1$ as quadratic non-residue where necessary), these values uniquely define p, the orders and the coefficients of both curves. One information that they don't define directly is which curve has positive trace and which has negative trace, so to associate each order to the right curve. Fortunately, this determination is easy since all points of each curve are generators and the scalar multiplication of any generator with the order of the curve gives the point at infinity.

What is left to determine are the base point generators. One of them can be found very quickly using a property of twisted pairs. For all $i \in \mathbb{F}_p$, either there is a point with coordinate $x = i$ in one curve, or a point with $x = i\beta$ on the twist, β being the quadratic non-residue used to calculate the twist. This property implies that the independent term b of one of the curves is a quadratic residue modulo p, and the points $(0, \pm\sqrt{b})$ are on that curve. Testing quadratic residuosity and extracting roots modulo a prime are relatively simple algorithms. For more details, refer to [7]. We define the base point generator of this curve by using the positive root of b (the one which is less than $\frac{p-1}{2}$) as y coordinate.

The generator of the other curve is a little trickier, and there's is no known polynomial-

time deterministic algorithm to find a point on that curve. We must define a generic algorithm that, for a given curve, will always find the same point. For x starting from 1 (zero, for sure, is not on that curve) and increasing by 1 each time, calculate the expression of right side of the equation (take as reference the equation 2) and, for the first value that the expression generates a quadratic residue, take that as x coordinate, calculate the square root of the result and take the positive root as y coordinate. That's the base generator of the second curve.

Notice that both procedures combined will always produce the same points, so it is safe to define the public keys based on those points, and omit them whenever it is easier. For the second point, by the distribution of quadratic residues, the expected number of trials you'll have to make is two, so it is fast enough to use in practice.

Summarizing the ideas, let's consider that the public parameters consist of the explicitation of the twisted pair, the orders, the prime number p, the generators and the public points. Using point compression, for a security factor $k = \log p$ these all sums up to $11k + 4$ bits of information. Using the techniques just presented, it gets down to about $2.25k + 2$ bits, with a minimal overhead.

One last remark about the value of D used. In the previous examples we assumed that D has class number one, and that makes the Hilbert polynomial have only one root. If a D with class number two or higher is used, then the polynomials can have more than one root, and the exact same root must be selected every time we reconstruct the curve parameters. There are two ways to solve that: 1) define a rule of what root is used (for example always the one with lower absolute value) or 2) send the root used along in the public key, which will add k bits to it, but it's still a lot less than expliciting all parameters.

5. The Kaliski pseudorandom bit generator

The Kaliski pseudorandom bit generator is based on the very same property we used to find generators of the curves. That is, for all integers $i \in \mathbb{F}_p$, if i is not on the first curve, then $i\beta$ is on the twist, β being the quadratic non-residue used to define the twist. Also, for each i value, there are two points over that curve, with positive and negative y value. That allows one to construct a mapping between the points on both curves and the integer values in the set $[0, 2p + 1]$. The function $\chi : E \cup E^t \rightarrow [0, 2p + 1]$ is defined as the following. A superscripted ^{t} indicates that the point is from the twist, and the *sign(y)* : $\mathbb{F}_p \rightarrow \{0, 1\}$ function returns 0 if the y is less then or equal $\frac{p-1}{2}$ and 1 otherwise.

$$\chi[E, E^t](P) = \begin{cases} 2x + sign(y) & \text{if} \quad P = (x, y), y \neq 0; \\ 2\left(\frac{x}{\beta} \mod p\right) + sign(y) & \text{if} \quad P = (x, y)^t, y \neq 0; \\ 2x & \text{if} \quad P = (x, 0); \\ 2\left(\frac{x}{\beta} \mod p\right) + 1 & \text{if} \quad P = (x, 0)^t; \\ 2p & \text{if} \quad P = \infty; \\ 2p + 1 & \text{if} \quad P = \infty^t. \end{cases} \tag{16}$$

This function is also used in the main algorithm. Although the χ function is really simple, there is one possible improvement. Notice that the multiplicative inverse of $(p - 1)$ is always itself. In case $p \equiv 3 \mod 4$, the division on the χ function turns into a subtraction. For example, $\frac{x}{\beta} \mod p \equiv p - x$.

Given a twisted pair, its orders and generators (namely E, E^t, n_E, n_{E^t}, and G_E, G_{E^t} as seen in the overview), the generator works iteratively. For a given seed $i \in [0, 2p+1]$ it checks if i is less than the order of E, and in that case it calculates the point iG_E using scalar multiplication. In case it is greater or equal, then it calculates $(i - n_E)G_{E^t}$. Notice that if $i > n_E$ then, $i - n_E < n_{E^t}$. Using the result point on the function χ just defined, it generates another number in the same interval and iterates.

At each iteration, the algorithm provides the $\log\log p$ higher bits of the logarithm of the point calculated. These higher bits are defined based on halving intervals as follows. The most significant bit of c with respect to the interval n is $\eta(c, n)$ defined as

$$\eta(c,n) = \begin{cases} 0, \text{ if } c \mod n < \frac{n}{2} \\ 1, \text{ if } c \mod n \geq \frac{n}{2} \end{cases} \tag{17}$$

In other words, the most significant bit $\eta(c, n)$ is 1 if the value is higher than half the interval in which it lies, otherwise it is 0. The *bth* most significant bit is defined recursively as the following

$$\eta_b(c,n) = \eta_{b-1}(2c,n) = \eta_1(2^{b-1}c,n).$$

Therefore, the algorithm return, at each iteration, the bits of $\eta_b(i, n_E)$ if $i < n_E$ and $\eta_b(i - n_E, n_{E^t})$ otherwise, for all $0 \leq b \leq \log\log p$.

6. The main algorithm

By now, it should be clear how to find the public key, the ninetuple

$$< p, E, E^t, n_E, n_{E^t}, G_E, G_{E^t}, P_E, P_{E^t} >$$

that are respectively the prime field, the twisted pair, its orders, base generators and the public points, which are the scalar multiplication of the secret keys s_E, s_{E^t} with the base point of each curve. Also, it should be clear how the Kaliski generator works. What is left to explain in more detail is the full process of encryption and decryption.

6.1. Encryption

As seen in the overview, given a secret discardable seed $a \in [0, 2p+1]$ the algorithm generates three points, M, T_E and T_{E^t} and another scalar value $r \in [0, 2p+1]$. T_E is a point and generator of E and T_{E^t} is a point and generator of E^t. Using these points as base generators of the curves, along with the seed r, the Kaliski pseudorandom bit generator is executed and a keystream with the size of the plaintext is generated. The actual encryption occurs by calculating the exclusive-or of this keystream with the plaintext.

The χ function is once more used extensively, exactly as defined on equation 16. Remember it takes a point on either curve and return a number in the set $[0, 2p+1]$. The three points and the r value are calculated using the following algorithm:

`If` $(a < n_E)$

$$\begin{aligned} M &= aG_E \\ T_E &= aP_E \\ T_{E^t} &= \left(\left\lfloor \frac{\chi(T_E) * n_{E^t}}{2p+1} \right\rfloor \right) P_{E^t} \\ r &= \chi(T_{E^t}) \end{aligned}$$

`else`

$$\begin{aligned} M &= (a - n_E)G_{E^t} \\ T_{E^t} &= (a - n_E)P_{E^t} \\ T_E &= \left(\left\lfloor \frac{\chi(T_{E^t}) * n_E}{2p+1} \right\rfloor\right) P_E \\ r &= \chi(T_E) \end{aligned}$$

`End if.`

The expression used to calculate the point T_{E^t} in the `if` branch generates uniformly all values in the interval $[0, n_{E^t} - 1]$ with high probability, as we'll show later. The same follows for the `else` branch.

6.2. Decryption

The decryption process is quite trivial, once one notices that

$$T_E = aP_E = s_E aG_E = s_E M$$

if M is on E and

$$T_{E^t} = (a - n_E)P_{E^t} = s_{E^t}(a - n_E)G_{E^t} = s_{E^t} M$$

if M is on E^t. Having T_E or T_{E^t}, the decryption is straightforward.

To check whether M lies on E or E^t, it suffices to evaluate the right side of the curve equations for the x value of M. Due to the properties of twisted pairs, either $x^3 + ax + b$ or $x^3 + a\beta^2 x + b\beta^3$, but not both, will be a quadratic residue modulo p. That tells if M is on E or E^t. The idea translates to the following algorithm (QR stands for Quadratic Residue):

`Make` z `equal the` x `coordinate of` M

`If (`$z^3 + az + b$ `is a QR. mod p)`

$$\begin{aligned} T_E &= s_E M \\ T_{E^t} &= \left(\left\lfloor \frac{\chi(T_E) * n_{E^t}}{2p+1} \right\rfloor\right) P_{E^t} \\ r &= \chi(T_{E^t}) \end{aligned}$$

`else`

$$\begin{aligned} T_{E^t} &= s_{E^t} M \\ T_E &= \left(\left\lfloor \frac{\chi(T_{E^t}) * n_E}{2p+1} \right\rfloor\right) P_E \\ r &= \chi(T_E) \end{aligned}$$

`End if.`

After that, the algorithm proceeds exactly like the encryption. Using the calculated points and the value r, the exact same keystream can be generated. By the properties of the exclusive-or operator, operating the keystream with the ciphertext results in the plaintext.

7. Security analysis

In this section, we'll analyze the encryption process and show that it is secure. In order to make the analysis, we initially give a mathematical definition of the ECDLP in our context. Then we define the Elliptic Curve Diffie Hellman Problem.

Definition 1 *Let P and $Q \neq \infty$ be two points of the elliptic curve E with prime order n_E. The ECDLP is to find the unique scalar value $r \leq n_E$ such that, by scalar multiplication, the equation $P = rQ$ is true.*

Definition 2 *Let P, aP and bP points of the elliptic curve E with prime order n_E such that the values a and b are not known. The ECDHP is to calculate the point abP.*

Obviously, given an algorithm for the ECDLP it's easy to solve the ECDHP. Although the inverse has not yet been proved, the following conjecture is widely accepted, and assumed in this paper:

Conjecture 1 *The ECDHP is equivalent to the ECDLP.*

Now we proceed to the analysis. First, we consider that the pseudorandom bit generator is secure with the assumption that the ECDLP is hard as proved by Kaliski in [6]. Therefore, for a polynomially bounded adversary, deriving patterns in the ciphertext is equivalent to a One-Time-Pad against an unbounded adversary. The only advantage one can get is by distinguishing the parameters used to generate the pseudorandom bit stream from a random distribution. We'll see that any advantage over that, with high probability, is equivalent to solve the ECDLP.

By inspecting the decryption process, it becomes clear that, for any encryption, the attacker breaks the system if he obtains the point T_E if $a < n_E$ and the point T_{E^t} otherwise. Let's first understand what happens if $a < n_E$ (the first branch), as clearly the else part is very similar. This first comparison ensures that a is within the order of the curve E. We claim and prove the following lemmas:

Lemma 1 *Let $a \in [0, 2p+1]$ be the encryption seed and $s_E \in [0, n_E - 1]$ be the private key over E. If a and s_E are selected uniformly within its intervals, and $a < n_E$, then the probability that $M = T_E$ is negligible.*

Proof By the algorithm, all points are generators and n_E is the order of any point. $M = aG_E$ and $T_E = aP_E = as_EG_E$. So $M = T_E$ if and only if $a \equiv s_E a \mod n_E$. As both a and s_E are less then n_E, this congruence is true only when s_E is the unit or $a = 0$. So, the probability of this congruence to hold is at most

$$\frac{2}{n_E} = \frac{2}{p + 1 - 2\sqrt{p}} = \Omega(p^{-1})$$

□

Lemma 2 *T_E is any point of the curve E with the same probability for each point.*

Proof If a and s_E are uniformly selected and less then n_E, then the congruence $s_E a \mod n_E$ has the exact same probability of being any value of the interval $[0, n_E - 1]$. As that is the exact value of the index of T_E at base G_E, then its index has the probability of $\frac{1}{n_E}$ of being any value. Relatively to a given generator, each index defines uniquely one point. So T_E is any point of the curve with same probability. □

Lemma 3 *Let $\mathcal{A} = \{\chi(P)|\forall P \in E\}$ be set of values generated by the χ function over all the points of the curve E. Then, the distribution of the values in the set $\mathcal{A}$ over the interval $[0, 2p+1]$ is uniform with high probability.*

Proof The worst case occurs with the curve having positive trace. This curve has at least $p + 1 - 2\sqrt{p}$ points. The first thing to notice is that

$$\frac{p+1-2\sqrt{p}}{2p+2} \cong \frac{1}{2},$$

which gives one point (consequently one value) for each two values of the set. In [9], Peralta shows that half the elements of a finite field are quadratic residues, and they are all uniformly distributed over all the values of the field, with high probability. That implies the values of x which evaluates the expression $x^3 + ax + b \mod p$ to a quadratic residue are also uniform over $[0, p-1]$. The χ function is, on average, two times the x value of the points of the curves. If these values x are uniform over $[0, p-1]$ then, the values $2x$, plus the points at infinity defined as the upper limit of the interval, are uniform over $[0, 2p+1]$. □

Lemma 4 *If the secret key $0 < s_{E^t} < n_{E^t}$ is uniformly selected, then, the point T_{E^t} is uniformly any point of the curve E^t with almost uniform probability.*

Proof By lemma 2, T_E is any point. So the possible values for $\chi(T_E)$ is any such that the points of E can generate. By lemma 3, this set of values is uniform over [0,2p+1]. That means that the expression

$$\left(\left\lfloor \frac{\chi(T_E) * n_{E^t}}{2p+1} \right\rfloor \right)$$

is a uniform reduction from the set of values in the interval [0,2p+1] (the χ function over E) to the set of values $[0, n_E - 1]$ which is almost the same as $[0, p-1]$. Therefore, the expression is diving by two the values over [0,2p+1] and, consequently, generating uniformly all values in the interval $[0, n_{E^t} - 1]$ with high probability. If that's true and if s_{E^t} is uniformly selected, then, by an argument similar to lemma 1 over E^t, T_{E^t} is any point of E^t with almost uniform probability. □

Theorem 1 *If $a \in [0, 2p+1]$, $0 < s_E < n_E$ and $0 < s_{E^t} < n_{E^t}$ are uniformly selected, then, if $a < n_E$, the scheme is as secure as the ECDLP.*

Proof Clearly, M and P_E are uniformly any points of E. As the index of T_E is the product of their indexes, then finding T_E is the ECDHP, which is as hard as the ECDLP by conjecture 1. Also, by lemmas 3 and 4 the distribution probability of $r = \chi(T_{E^t})$ is uniform over $[0, 2p+1]$ and any advantage of finding r implies the same advantage of finding T_{E^t}, as the χ function is easily invertible. The point T_{E^t} is uniformly any point of E^t and its index is the product of the secret key s_{E^t} and the value of the χ function over the point T_E, uniformly reduced to the interval $[0, n_{E^t} - 1]$. As T_E is uniformly any point of E and s_{E^t} is uniformly selected, then calculating T_{E^t} is as difficult as calculating T_E, which was already established to be as hard as the ECDLP. □

All the proofs carry out in the exactly the same way for the case when $a \geq n_E$. In that case, $a - n_E$ is a uniform value in the interval $[0, n_{E^t} - 1]$. That implies that M and

T_{E^t} are uniformly any points of E^t and so on. Finally, as both branches of the encryption algorithm can be proved to be secure, then, all the algorithm is secure.

We also claim that the scheme is semantically secure, hence, probabilistic. Semantic security is achievable by proving that no polynomially bounded adversary can obtain any advantage of distinguishing from a random ensemble the set of possible ciphertexts of any given plaintext.

Theorem 2 *The scheme is semantically secure.*

Proof The sequences generated by any two distinct group generators, by operating them with itself until the result is ∞, are different. So, each triple $\langle T_E, T_{E^t}, r\rangle$ defined by each value a generate a different pseudorandom bit stream on the Kaliski generator. This triple can be reduced to the tuple $\langle T_E, T_{E^t}\rangle$ because $r = \chi(T_E)$ or $r = \chi(T_{E^t})$. For each possible value of a, at least one of the points of the tuple will be different. If a is uniformly selected and the Kaliski generator is cryptographically secure, then, any advantage of distinguishing the ciphertext from a random stream is, at most,

$$\frac{1}{p+1-2\sqrt{p}} \cong p^{-1}.$$

□

References

[1] I. F. Blake, G. Seroussi and N. P. Smart, "Elliptic Curves in Cryptography," *London Mathematical Society. Lecture Notes Series 265. Cambridge University Press.*, 1999;

[2] M. Blum and S. Goldwasser, "An Efficient Probabilistic Public-key Encryption Scheme Which Hides All Partial Information," *Advances in Cryptology - CRYPTO'84*, vol. 196, *Springer Verlag*, pp. 289–302, 1985;

[3] H. Cohen, "A course in computational algebraic number theory," *Graduate Texts in Mathematics 138. Springer-Verlag.*, 1993;

[4] R. Cramer and V. Shoup, "A practical public key crypto system provably secure against adaptive chosen ciphertext attack," *proceedings of Crypto 1998*, LNCS 1462, p.13ff, 1998;

[5] S. Goldwasser and S. Micali, "Probabilistic encryption," *JCSS*, vol. 28, pp. 270–299, April 1948;

[6] B. S. Kaliski, Jr. "Elliptic Curves and Cryptography: A Pseudorandom Bit Generator and Other Tools," *PhD Thesis, MIT EECS Dept.*, January 1988;

[7] A. Menezes, P. van Oorschot, and S. Vanstone, "Handbook of Applied Cryptography", *CRC Press*, 1996;

[8] T. Okamoto, E. Fujisaki and H. Morita, "PSEC: Provably Secure Elliptic Curve Encryption Scheme," *Submission to IEEE P1363a.*, March 1999;

[9] R. Peralta , "On the distribution of quadratic residues and non-residues modulo a prime number", *Mathematics of Computation*, Vol. 58, pp. 433 – 440, 1992.