An introduction to elliptic curve cryptography

Johan Dams - October 12, 2012

Editor's note: See the original article on PurpleAlientPlanet.

Some of my research is focused on the implementation issues of elliptic curve cryptography on embedded systems. Since I often have to explain what elliptic curve cryptography exactly is, I decided to write this little introduction on the matter. Maybe this will get the attention of some of my students and can perhaps get them more interested in the mathematical branch of finite fields in algebra.

Introduction
Elliptic curve cryptography (ECC) is a public key cryptography method, which evolved form Diffie Hellman. To understanding how ECC works, lets start by understanding how Diffie Hellman works.

The Diffie Hellman key exchange protocol, and the Digital Signature Algorithm (DSA) which is based on it, is an asymmetric cryptographic systems in general use today. It was discovered by Whitfield Diffie and Martin Hellman in 1976, and uses a problem known as the Discrete Logarithm Problem (DLP) as its asymmetric operation. The DLP concerns finding a logarithm of a number within a finite field arithmetic system.

Prime fields are fields whose sets are prime. In other words, they have a prime number of members. Prime fields turn out to be of great use in asymmetric cryptography since exponentiation over a prime field is relatively easy, while its inverse, computing the logarithm, is difficult. The "Diffie-Hellman Method for Key Agreement" allows two hosts to create and share a secret key. This is done by the following method:

1. First the hosts must get the "Diffie-Hellman parameters": a prime number p (larger than 2) and "base", g, an integer that is smaller than p. They can be hard coded or fetched from a server, depending on the implementation.

2. The hosts each generate a secret private number called x, which is less than $p - 1$.

3. Next, the hosts generate the public keys, y. They are created with the function:

$$y = g^x \bmod p$$

4. The two host then exchange the public keys (y) and these exchanged numbers are then converted into a secret key, z as follows:

$$z = y^x \bmod p$$

The secret key z can at this point be used as the key for a standard encryption method, used to transfer the information between the hosts. Mathematically, the two hosts have generated the same value for the secret

key x since:

$$x = (g^x \bmod p)^{x'} \bmod p = (g^{x'} \bmod p)^x \bmod p$$

Using the values in the equation above, finding the discrete logarithm problem is finding x when only y, g and p are known. As an example, take the situation in which someone has multiplied g by itself x times, and reduced the result into the field (by performing the modulo operation) as often as needed to keep the result smaller than p. In this case, when knowing y, g and p, the problem is trying to find what what value of x was used. This turns out to be extraordinarily difficult to do for large enough values of p, where p is prime. It is in fact so much more difficult to do than just finding y from g, x and p that, even using the world's fastest supercomputer, it would be unfeasible to attempt within a reasonable amount of time.

Mathematically, a proof to this effect is neither known nor thought to be forthcoming. Before wide-scale implementation, it is thus of the utmost importance that an extensive investigation of the true complexity of the problem is done in order to obtain the highest degree of confidence in the security of discrete logarithm based cryptographic systems. Such an investigation is in progress by various researchers around the world.

Elliptic curves

Since the discovery of RSA (and El-Gamal) their ability to withstand attacks has meant that these two cryptographic systems have become widespread in use. They are being used every day both for authentication purposes as well as encryption/decryption. Both systems cover the current security standards--so why invent a new system? Even though ECC is relatively new, the use of elliptic curves as a base for a cryptographic system was independently proposed by By Victor Miller and Neil Koblitz. What makes it stand apart from RSA and El-Gamal is its ability to be more efficient that those two. The reason why this is important are the developments in information technology--most importantly hand held, mobile devices, sensor networks, etc. Somehow, there must be a way to secure communications generated by these devices, however their computing power and memory are not nearly as abundant as on their desktop and laptop counterparts. A contemporary desktop or laptop system has no problems working with 2048 bit keys and higher, but these small embedded devices do since we do not want to spend a lot of their resources and bandwidth securing traffic.

The operations on which RSA are founded are modular exponentiation in integer rings. The security of RSA depends on the difficulty of factoring large integers which can be done in sub-exponential times. For the ECDLP however, only exponential algorithms are known which means we can use shorter keys for security levels where RSA and El-Gamal would need much bigger keys. For example, a 160 bit ECC key and a 1024 bit RSA key offer a similar level of security. To reach the same level of security than a 15360 bit RSA key, one only needs 512 bit ECC key.

Operations on elliptic curves

The security of ECC depends on the difficulty of the Elliptic Curve Discrete Logarithm Problem. This problem is defined as follows: let P and Q be two points on an elliptic curve such that $kP = Q$, where k is a scalar. Given P and Q, it is computationally unfeasible to obtain k, if k is sufficiently large. Hence, k is the discrete logarithm of Q to P. We can see that the main operation involved in ECC is point multiplication, namely, multiplication of a scalar k with any point P on the curve to obtain another point Q on the curve.

This is also the reason a ECC key of 160 bits provides the equivalent protection of a symmetric key of 80 bits, namely because of the methods used to crack $kP = Q$. If one knows P and Q, one must guess at least the square root of the number of points on average to find k. So if the field size is 2^m, one must guess $2^{(m/2)}$ points. With a 80 bit symmetric key, it takes 2^{79} guesses to crack it on average. The table below gives a comparison of equivalent key sizes.

Bits of Security	Symmetric Algorithm	RSA	ECC
80	2TDEA	$k = 1024$	$f = 160 - 223$
112	3TDEA	$k = 2048$	$f = 224 - 255$
128	AES-128	$k = 3072$	$f = 256 - 383$
192	AES-192	$k = 7680$	$f = 384 - 511$
256	AES-256	$k = 15360$	$f = 512+$

Each curve has a specially designated point G called the base point chosen such that a large fraction of the elliptic curve points are multiples of it. To generate a key pair, one selects a random integer k which serves as the private key, and computes kG which serves as the corresponding public key. For cryptographic application the order of G, that is the smallest non-negative number n such that $nG = O$, with O the point at infinity, must be prime.

Point multiplication

In point multiplication a point P on the elliptic curve is multiplied with a scalar k using elliptic curve equation to obtain another point Q on the same elliptic curve, giving $kP = Q$. Point multiplication can be achieved by two basic elliptic curve operations, namely point addition and point doubling. Point addition is defined as adding two points P and Q to obtain another point R written as $R = P + Q$. Point doubling is defined as adding a point P to itself to obtain another point Q so that $Q = 2P$.

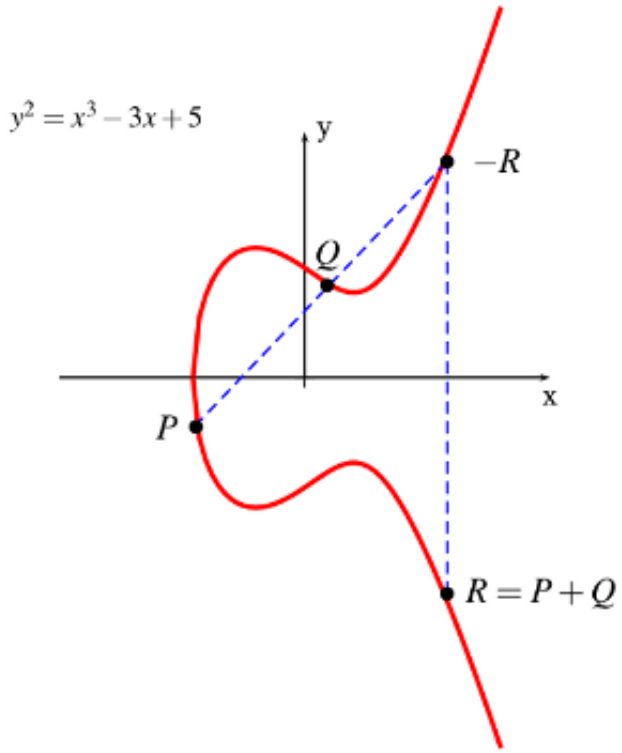
Point multiplication is hence achieved as follows: let P be a point on an elliptic curve. Let k be a scalar that is multiplied with the point P to obtain another point Q on the curve so that $Q = kP$. If $k = 23$ then $kP = 23P = 2(2(2(2P) + P) + P) + P$.

Thus point multiplication uses point addition and point doubling repeatedly to find the result. The above method is called the 'double and add' method for point multiplication. There are other, more efficient methods for point multiplication.

Point addition

Point addition is the addition of two points P and Q on an elliptic curve to obtain another point R on the same elliptic curve. This is demonstrated geometrically in the figure below for the condition that **Q \neq -P**

.



Analytically, we can perform a point addition as follows.

Consider two distinct points P and Q so that $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$.

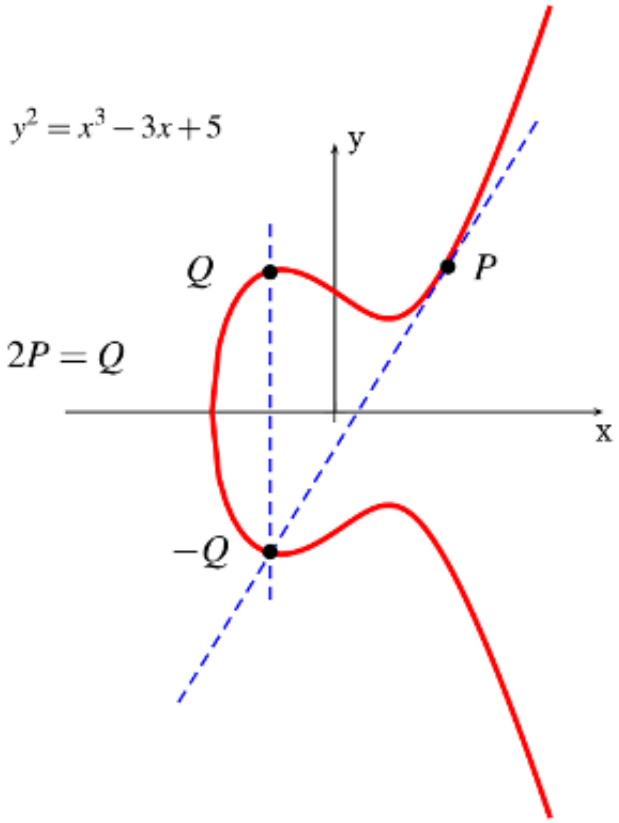
Let $R = P + Q$ where $R = (x_R, y_R)$. Then

$x_R = s^2 - x_P - x_Q$
$y_R = -y_a + s(x_P - x_Q)$
$s = (y_P - y_Q)/(x_P - x_Q)$, thus s is the slope of the line through P and Q.

If $Q = -P$ i.e. $Q = (x_P, -y_P)$ then $P + Q = O$ where O is the point at infinity.
If $P = Q$ then $P + Q = 2P$ then point doubling equations are used.

Also note that the addition is commutative, thus $P + Q = Q + P$.

Point doubling
Point doubling is the addition of a point P on the elliptic curve to itself to obtain another point Q on the same elliptic curve. To double a point P to get Q, i.e. to find $Q = 2P$, consider a point P on an elliptic curve as shown in the figure below. If the y coordinate of the point P is not zero then the tangent line at P will intersect the elliptic curve at exactly one more point $-Q$. The reflection of the point $-Q$ with respect to x-axis gives the point Q, which is the result of doubling the point P.



Analytically, we can again write this as follows.

Consider a point P such that $P = (x_P, y_P)$, where $y_P \neq 0$.

Let $Q = 2P$ where $Q = (x_Q, y_Q)$. Then

x_Q = s^2 2x_P
$y_Q = -y_P + s(x_P - x_Q)$
$s = (3x_P^2 + a)/(2y_P)$, where s is the tangent at point P and a is one of the parameters chosen with the elliptic curve.

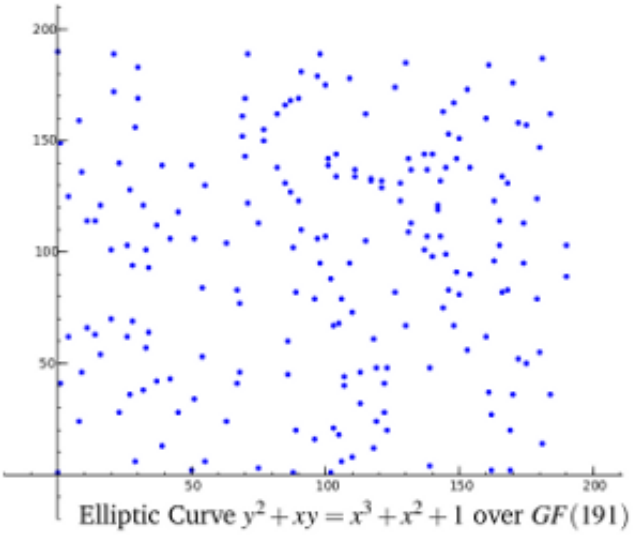
If **y_P = 0** then $2P = O$, where O is the point at infinity.

Finite fields
The elliptic curve operations defined in the previous section are on real numbers. Operations over the real numbers are slow and inaccurate due to rounding errors. Cryptographic operations have to be fast and accurate. To make operations on elliptic curve accurate and more efficient, the elliptic curve cryptography is defined over finite fields, also called Galois fields in honor of the founder of finite field theory, Évariste Galois. For example:

- Prime field $GF(p)$
- Binary field $GF(2^m)$

The field is chosen with finitely large number of points suited for cryptographic operations. Even though

the curve would no longer a gently flowing graph, as shown in the figure below, the algebraic equations for point addition and doubling still apply.



Elliptic Curve $y^2+xy=x^3+x^2+1$ over $GF(191)$

Operations over Prime Field F_p

Let F_p be a prime finite field so that p is an odd prime number, and let $a, b \in F_p$ satisfy $4a^3+27b^2 \pmod p \neq 0$. Then an elliptic curve $E(F_p)$ over F_p defined by the parameters $a, b \in F_p$ consists of the set of solutions or points $P=(x,y)$ for $x, y \in F_p$ to the equation:

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

Together with the extra point O at infinity. The equation $y^2 \equiv x^3+ax+b \pmod p$ is called the defining equation of $E(F_p)$. For a given point $P=(x_P, y_P)$, x_P is called the x-coordinate of P, and y_P is called the y-coordinate of P.
The prime number p is chosen such that there is a finitely large number of points on the elliptic curve to make the cryptosystem secure, usually between 112 and 521 bits.

The number of points on $E(F_p)$ is denoted by $\#E(F_p)$. Hasse's Theorem on elliptic curves states that:

$$p+1-2\sqrt{p} \leq E(F_p) \leq p+1+2\sqrt{p}$$

It is then possible to define an addition rule to add points on E. The addition rule is specified as follows:

Rule to add the point at infinity to itself:

$$O+O=O$$

Rule to add the point at infinity to any other point:

$$(x,y)+O=O+(x,y)=(x,y) \forall (x,y) \in E(F_p)$$

Rule to add two points with the same x-coordinates when the points are either distinct or have y-coordinate 0:

$(x, y) + (x, -y) = O \, \forall (x, y) \in E(F_p)$
This also means that the negative of the point (x, y) is $-(x, y) = (x, -y)$

Rule to add two points with different x-coordinates: Let $(x_1, y_1) \in E(F_p)$ and **(x_2, y_2) \in E(F_p)** be two points such that $x_1 \neq x_2$. Then $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$, where:

$$x_3 \equiv \lambda^2 - x_1 - x_2 \pmod{p}$$
$$y_3 \equiv \lambda(x_1 - x_3) - y_1 \pmod{p}$$
$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} \pmod{p}$$

Rule to add a point to itself (double a point): Let $(x_1, y_1) \in E(F_p)$ be a point with $y_1 \neq 0$. Then $(x_1, y_1) + (x_1, y_1) = (x_3, y_3)$, where:

$$x_3 \equiv \lambda^2 - 2x_1 \pmod{p}$$
$$y_3 \equiv \lambda(x_1 - x_3) - y_1 \pmod{p}$$
$$\lambda = \frac{3x_1^2 + a}{2y_1} \pmod{p}$$

The set of points on $E(F_p)$ forms a group under this addition rule. Furthermore the group is Abelian, meaning that $P_1 + P_2 = P_2 + P_1$ for all points $P_1, P_2 \in E(F_p)$. Notice that the addition rule can always be computed efficiently using simple field arithmetic.

What about scalar multiplication of elliptic curve points? These can be computed efficiently using the addition rule together with the double-and-add algorithm or one of its variants.

Operations on Binary Field F_{2^m}

Let F_{2^m} be a characteristic 2 finite field, and let $a, b \in F_{2^m}$ satisfy $b \neq 0$ in F_{2^m}. Then a (non-supersingular) elliptic curve $E(F_{2^m})$ over F_{2^m} defined by the parameters $a, b \in F_{2^m}$ consists of the set of solutions or points $P = (x, y)$ for $x, y \in F_{2^m}$ to the equation:

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

Where $b \neq 0$ together with an extra point O at infinity, with the only elliptic curves over F_{2^m} of interest being non-supersingular elliptic curves. Here the elements of the finite field are integers of length at most m bits. These numbers can be considered as a binary polynomial of degree $m - 1$. In this binary polynomial the coefficients can only be 0 or 1. All the operation such as addition, subtraction, division, multiplication involves polynomials of degree $m - 1$ or less.

The number of points on $E(F_{2^m})$ is denoted by $\#E(F_{2^m})$. The Hasse Theorem states that:

$$2^m + 1 - 2\sqrt{2^m} \leq E(F_{2^m}) \leq 2^m + 1 + 2\sqrt{2^m}$$

It is again possible to define an addition rule to add points on E as it was done for $E(F_p)$. The addition rule is specified as follows:

Rule to add the point at infinity to itself:

$$O + O = O$$

Rule to add the point at infinity to any other point:

$$(x, y) + O = O + (x, y) = (x, y) \forall (x, y) \in E(F_{2^m})$$

Rule to add two points with the same x-coordinates when the points are either distinct or have x-coordinate 0:

$(x, y) + (x, x + y) = O \forall (x, y) \in E(F_{2^m})$
This also means that the negative of the point (x, y) is $-(x, y) = (x, x + y)$

Rule to add two points with different x-coordinates: Let $(x_1, y_1) \in E(F_{2^m})$ and **(x_2, y_2) \in E(F_{2^m})** be two points such that $x_1 \neq x_2$. Then $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$, where:

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \ \ in \ \ F_{2^m}$$
$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \ \ in \ \ F_{2^m}$$
$$\lambda \equiv \frac{y_1 + y_2}{x_1 + x_2} \ \ in \ \ F_{2^m}$$

Rule to add a point to itself (double a point): Let $(x_1, y_1) \in E(F_{2^m})$ be a point with $x_1 \neq 0$. Then $(x_1, y_1) + (x_1, y_1) = (x_3, y_3)$, where:

$$x_3 = \lambda^2 + \lambda + a \ \ in \ \ F_{2^m}$$
$$y_3 = (x_1)^2 + (\lambda + 1)x_3 \ \ in \ \ F_{2^m}$$
$$\lambda \equiv x_1 + \frac{y_1}{x_1} \ \ in \ \ F_{2^m}$$

The set of points on $E(F_{2^m})$ forms an Abelian group under this addition rule. Notice that the addition rule can always be computed efficiently using simple field arithmetic. As before, scalar multiplication is the process of adding P to itself k times. The result of this scalar multiplication is denoted kP and can be computed efficiently using the addition rule together with the double-and-add algorithm or one of its variants.

Besides the curve parameters a and b, there are other parameters that must be agreed upon by both parties involved. These are called the domain parameters. The domain parameters for both prime fields and binary fields are described below.

Domain parameters for F_p

The domain parameters for elliptic curves over F_p are p, a, b, G, n and h. Here p is the prime number defined for the finite field F_p while a and b are the parameters defining the curve $y^2 \pmod p = x^3 + ax + b \pmod p$. G is the generator point (x_G, y_G), a point on the elliptic curve chosen for cryptographic operations and n is the order of the elliptic curve. The scalar for point multiplication is chosen as a number between 0 and $n - 1$. h is the cofactor where $h = \frac{\#E(F_p)}{n}$.

Domain Parameters for F_{2^m}

The domain parameters for elliptic curve over F_{2^m} are m, $f(x)$, a, b, G, n and h. m is an integer defined for the finite field F_{2^m}. The elements of the finite field F_{2^m} are integers of length at most m bits. $f(x)$ is the irreducible polynomial, known as the reduction polynomial, of degree m used for elliptic curve operations while a and b are the parameters defining the curve $y^2 + xy = x^3 + ax^2 + b$. G is again the generator point (x_G, y_G) and a point on the elliptic curve chosen for cryptographic operations while n is the order of the elliptic curve. The scalar for point multiplication is chosen as a number between 0 and $n - 1$. h is the cofactor where $h = \frac{\#E(F_{2^m})}{n}$.

Choosing the field

Should we choose a curve over the prime field F_p or the binary field F_{2^m}? This decision has to be made based on the way the system is going to be implemented. While curves over the prime field can be more efficient to implement in software, there are optimisations possible that puts the efficiency of curves over a binary field on par with those over a prime field.

*During his professional career, **Johan Dams** has been involved in many projects around the world in the fields of embedded systems, software engineering and systems and security among others. He is a published researcher working in the fields of cryptography and robotics and has over ten years of experience running software teams. He is a lecturer at the Vaasa University of Applied Sciences in Finland and has been invited to give guest lectures at Universities all over the world on topics such as robotics, cryptography, machine vision and learning, software development, software team management and embedded systems and its applications.*