Entrust®
Securing Digital Identities & Information

EntrustSecurity
@Entrust

Zero to ECC in 30 Minutes

A primer on elliptic curve cryptography

Entrust.com
1-888-690-2424
Entrust@Entrust.com

Entrust®
Securing Digital Identities & Information

Table of Contents

What is Elliptic Curve Cryptography?

Elliptic curve cryptography, or ECC, is one of several public-key cryptosystems that depend, for their security, on the difficulty of the discrete logarithm problem.

Public-key cryptosystems of this type are based upon a one-way function; a function for which the output corresponding to a particular input is easy to calculate, but for which the input corresponding to a particular output is computationally infeasible to calculate.

The required one-way function is created by repeatedly applying an operation to one element in a group of elements. The input to the one-way function is the number of times the operation is to be applied, and this is computationally infeasible to determine by someone who only knows the element to which the operation is applied and the resulting output.

In the earliest form of public-key cryptosystem, the chosen group comprised the set of integers from 1 … p-1, where *p* was a large prime number, and the group operation was multiplication of two integers *modulo p*. In elliptic-curve cryptosystems, the chosen group elements are points on an elliptic curve. And, the group operation is addition of two points.

Elliptic Curves

The most common elliptic curve equation for cryptographic applications takes the following form:

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

But, for those of us whose knowledge of mathematics is a bit rusty, it provides little insight into why this particular form of curve is useful in cryptographic applications.

First, the fact that the term in *y* is squared means that the curve is symmetrical about the *x* axis; i.e., for every positive value of *y*, there is a corresponding negative value with the same *x* coordinate.

Also, the fact that the highest order term in *x* is *x*-cubed means that, for certain values of *a* and *b*, there is exactly one point of inflection in the line on each side of the *x* axis. The inflection points are located where the line crosses the *y* axis.

An inflection point is a point on a curve where the curvature switches sign; the line curves one way on one side of the point and the other way on the other side. In other words, the curve has a wobble in it.

Figure 1 shows an example curve for particular values of *a* and *b* (a = 3, b = 1). The symmetry about the *x* axis is clear to see.

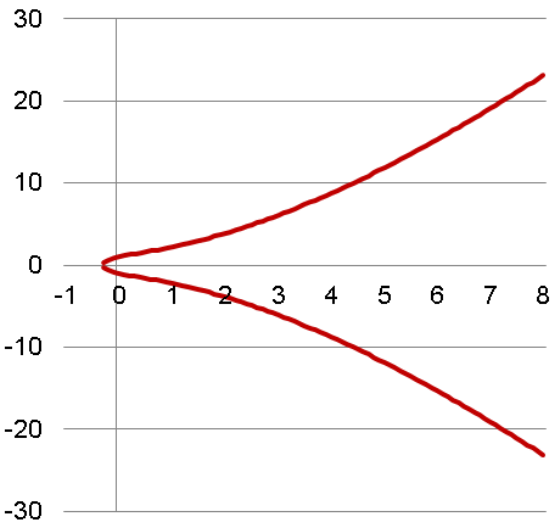

Figure 1: Example elliptic curve.

> *Elliptic curve cryptography, or ECC, is one of several public-key cryptosystems that depend, for their security, on the difficulty of the discrete logarithm problem.*

And, we can also see the points of inflection in Figure 2.

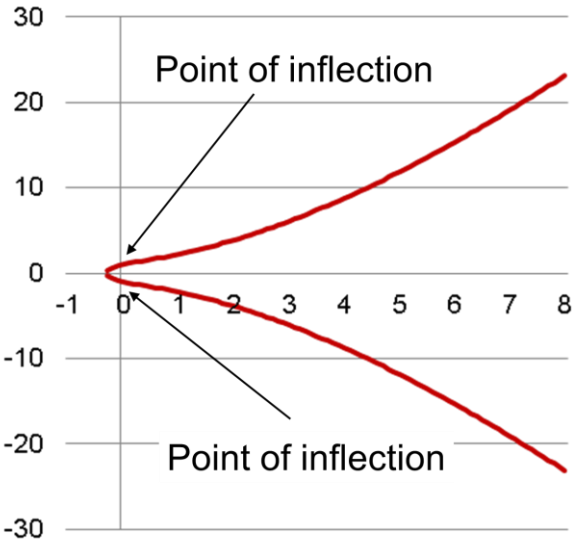

Figure 2:
Points of inflection.

A necessary consequence for curves with these two properties is that a straight line that crosses the curve in two places also crosses the curve in exactly one other place, Figure 3.

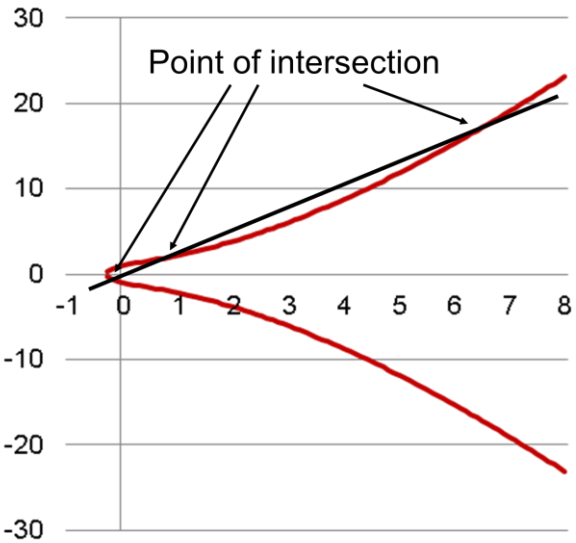

Figure 3:
Straight line intersects the curve in three places.

There is a notable exception to this rule for vertical lines, which either cross the curve in just two places or in one place (in the case where the line is the vertical tangent). We will be coming back to this later on.

Arithmetic on Elliptic Curves

The fact of three intersection points raises the intriguing possibility of doing arithmetic using points on the curve; two points could be combined in an operation that results in a third point.

We call the operation of finding the resulting point "addition" (addition of the two other points), although it bears no resemblance to the familiar operation of addition with numbers.

The operation of adding two points involves finding the third point of intersection between the curve and the straight line passing through the two points that are being added, then reflecting the result in the x axis.

This produces another point on the curve, see Figure 4. The reflection step is included so that a point can be added to itself repeatedly (i.e., multiplied by an integer) without falling into a short repeating cycle.

The mathematics involved in understanding and implementing point addition on an elliptic curve are taught in the high-school curriculum, see Figure 4.

In the diagram, *s* is the slope of the straight line, and (x_3,y_3) are the coordinates of the point that is the sum of the two points (x_1,y_1) and (x_2,y_2). Calculating *s* and y_3 uses elementary results in the geometry of triangles, whereas calculating x_3 depends on finding the roots of a cubic equation. Both of these topics should be familiar to the high-school math student.

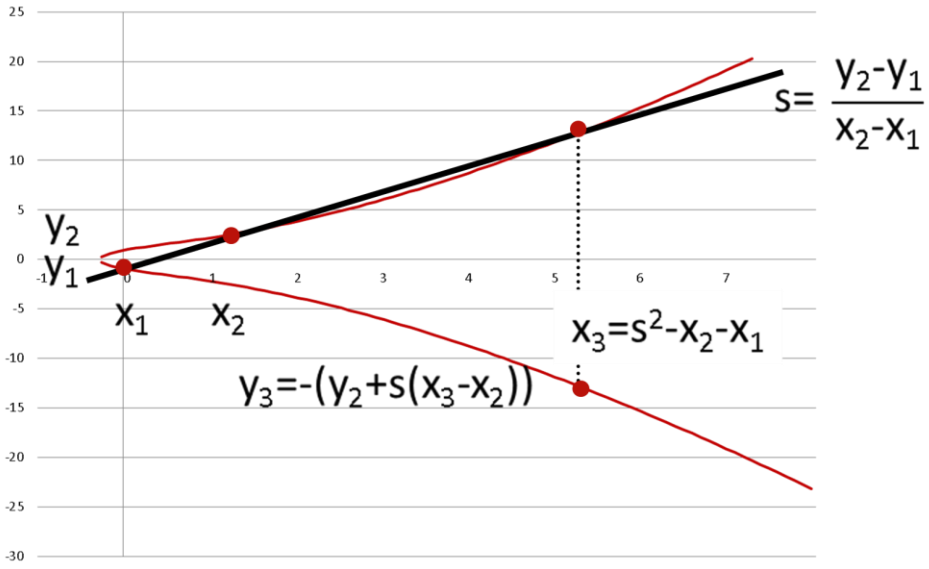

Figure 4: Calculating the result of point addition.

Adding a point to itself is also known as doubling the point, and it involves finding a second point where the tangent at the first point intersects the curve, and then reflecting that point in the x-axis, as in Figure 5.

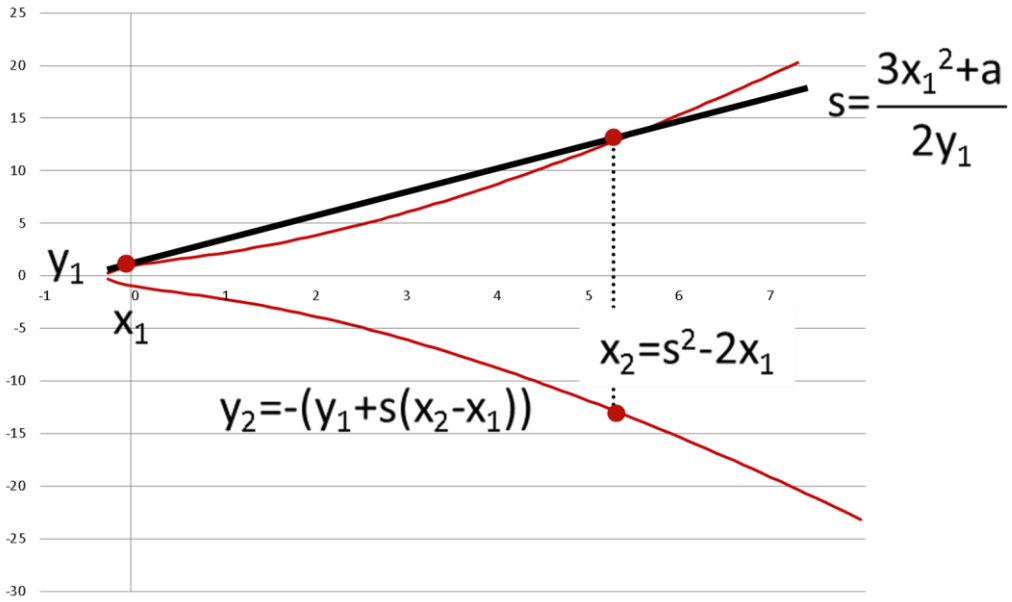

Figure 5: Calculating the result of point doubling.

As above, calculating *s* and y_2 uses elementary results in the geometry of triangles, and calculating x_2 involves solving a cubic equation.

Discrete Points on an Elliptic Curve

Performing mathematics with points whose coordinates are real numbers is of little value in cryptographic applications, because computer systems don't deal predictably with real numbers.

What we need is a large, but finite, set of discrete values, such as the elements of a "field." A field is a set of elements that is "closed" under the operations of addition, subtraction, multiplication and division that we require for the point addition calculations.

Closure means that, after applying any of the field operations to elements of the field, the result is also an element of the field. By choosing to use a field, we preserve the additive property possessed by real points on the elliptic curve, but the field elements are discrete and finite, and therefore they are amenable to manipulation by a computer with repeatable results.

Mathematical identities (i.e., equations) are preserved when field elements and operations are substituted for real numbers and the corresponding real operations.

In this way, calculating the third intersection point of a line, for which the coordinates of the other two intersection points are field elements, results in a third point whose coordinates are also field elements. Therefore, the result of adding discrete points is predictable and precise; just what we need for a cryptographic primitive.

The integers modulo a prime, p, form a suitable field. Polynomials with binary coefficients modulo an irreducible polynomial also form a suitable field. But, we won't discuss that possibility any further here; we'll limit our discussion to the more common case of fields over the integers *modulo p.*

The set of points inherits the properties of the elliptic curve, such as straight lines intersecting in three places. So, in order to add two points, all we need do is perform the geometric calculations required for point addition in the set of integers *mod p*.

Operations *mod p* are similar, but not quite identical to the equivalent and familiar operations over the integers.

Addition, for instance is the same, except that, in the event the result is greater than or equal to *p*, *p* is subtracted, so that the result falls in the range *0 .. p - 1*.

Similarly for subtraction, if the result of subtraction is less than *0*, then *p* is added, so the result, again, falls in the range *0 .. p - 1*.

In the case of multiplication, the two numbers are multiplied and then the remainder is taken after division by *p*. Again, the result falls in the range *0 .. p - 1*.

Division is a bit more complicated. Every number in the range *1 .. p-1* has a “multiplicative inverse.” When a number is multiplied by its multiplicative inverse, the result is *1 mod p*.

For instance, the inverse of *13 mod 23* is *16*, because *13 x 16 is 208*, which is *9 x 23 + 1*. So, instead of directly dividing one number by another, the first number has to be multiplied by the multiplicative inverse of the second number.

For example,

12 / 13 mod 23

= 12 x 16 mod 23

= 8 mod 23.

That is, in the field of integers *mod 23*, *12* divided by *13* is *8*. As with real numbers, division by zero is undefined.

Figure 6 shows all of the points that satisfy the elliptic curve equation:

$$y^2 = x^3 + 3x + 1$$

with coordinates over the integers mod p, where p is 23. To illustrate:

when *x* is 11,

$y^2 = (1331 + 3 \times 11 + 1) \bmod 23 = 8$, and so

$y = \sqrt{8} = 10 \text{ or } 13$

since $10 \times 10 \bmod 23 = 8$,
and $13 \times 13 \bmod 23 = 8$

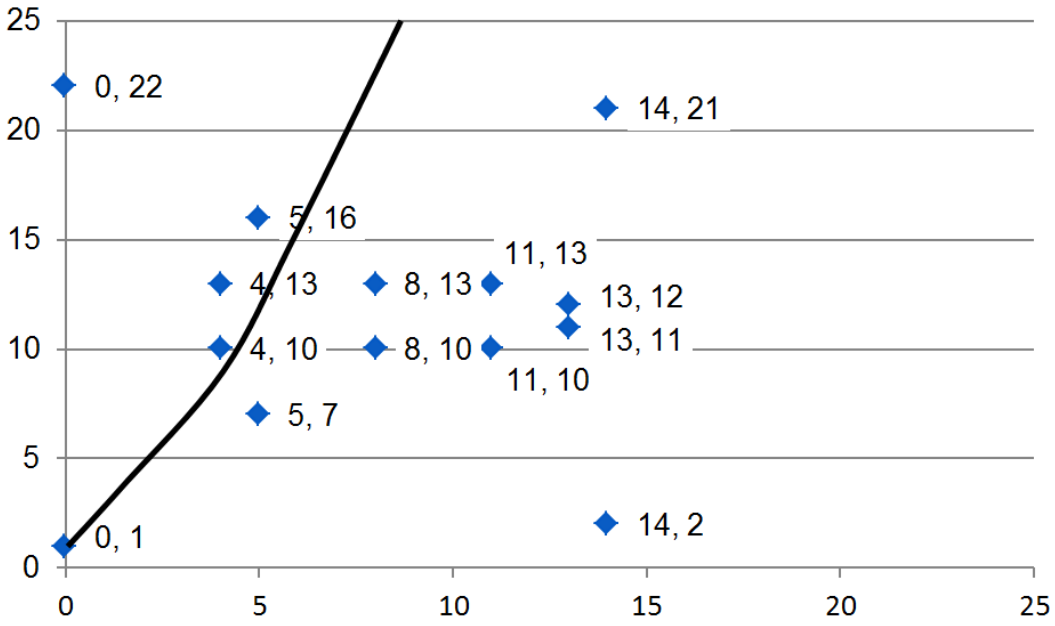

Figure 6: Discrete points on the curve.

The continuous line in Figure 6 is the original curve through this region of the x-y plane. Note that, with the exception of the point *(0,1),* none of the points are congruent with a solution of the curve equation over the real numbers, either in this region or any other region of the curve for that matter.

Also note that not every *x* value has a solution; for these values of *x* the right-hand side of the curve equation is not a perfect square. Further note that the points form pairs arranged symmetrically about the horizontal line, $y = p/2$.

So, there must be an even number of points. This is because, for every *x* that has a solution on the curve, there are two solutions: *y* and *–y*, and taken *mod p*,

$$-y = p - y.$$

In what follows, we will use uppercase bold characters to indicate curve points and lowercase normal characters to indicate integers.

Elliptic Curve Points Form a Group

The curve over the real numbers serves no further purpose at this point, so our discussion will be limited to points with integer coordinates in the range *0 .. p-1*.

With one caveat, these points form a finite 'group.' Like a field, a group contains a set of elements. But, unlike a field, which defines four operations, a group defines just one operation (the group operation). And, under this operation, the group is closed (i.e., when two group elements are combined by the group operation, the result is another group element).

In order for the set of points to form a group, it must have an "identity element." This is also known as the notional "point at infinity," designated by **0**.

Remember that the points come in pairs, arranged symmetrically about the horizontal *y = p/2* line. Each element of the pair is the additive inverse of its partner (i.e., when they are added together, the result is **0**).

When we join a point **P** to its additive inverse (-**P**), we get a vertical line, which also goes through the "point at infinity." By including the "point at infinity" (or identity element) in the set of points, we complete the group with order (i.e., the number of group elements) '*q*.' In our example, *q* has the value fifteen.

$$\mathbf{P} + (\mathbf{-P}) = \mathbf{O} \text{ and}$$
$$\mathbf{P} + \mathbf{O} = \mathbf{P}$$

Because the coordinates of a point are field elements, the operation of adding curve points still works. But, the formulae from **Figure 4** have to be modified as follows in order to calculate the coordinates of the point that is the sum of two points.

$$s = (y_2 - y_1).(x_2 - x_1)^{-1} \bmod p$$
$$x_3 = s^2 - x_2 - x_1 \bmod p$$
$$y_3 = -y_2 + s.(x_2 - x_3) \bmod p$$

Similarly, the formulae of **Figure 5** have to be modified as follows to calculate the coordinates of the point that is the double of another point:

$$s = (3x_1^2 + a).(2y_1)^{-1} \bmod p$$
$$x_2 = s^2 - 2x_1 \bmod p$$
$$y_2 = -y_1 + s.(x_1 - x_2) \bmod p$$

No matter which two points are chosen as input, the result is always another point.

Note that *b* does not appear in these formulae, and a only appears in the formulae for point doubling.

Certain points are called "generators," because their multiples generate every point in the group (see **Figure 7**).

Points in the group are generated by successively applying the group operation (point addition) to a point called a generator (alternatively called a base point). For instance, if we choose the point (11,13), its multiples generate fifteen points.

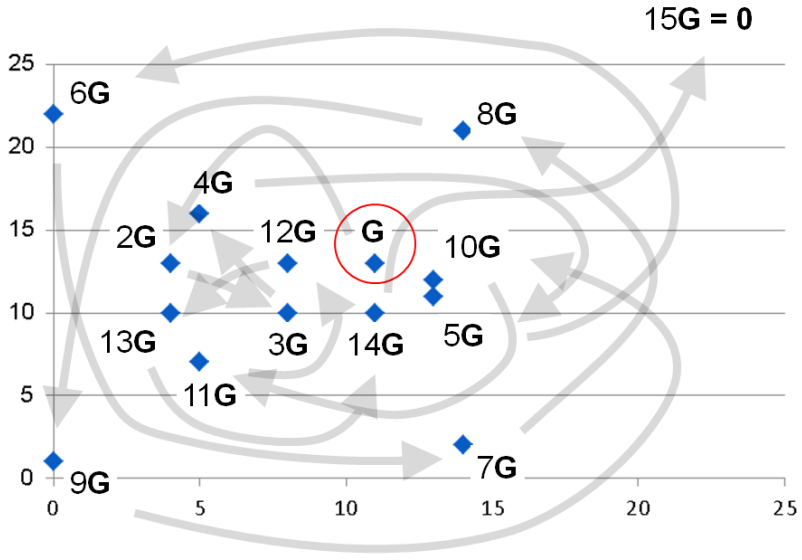

Figure 7: Points are multiples of a "generator."

Beyond 15, the cycle of points repeats such that 16**G** = **G**, 17**G** = 2**G**, etc.

Not all points are generators, however (see **Figure 8**). If we choose the point (0,1), for instance, its multiples generate five points, so the order of the sub-group generated by the point (0,1) is only five.

Even if *p* is large, it is not guaranteed that a particular sub-group will be large too.

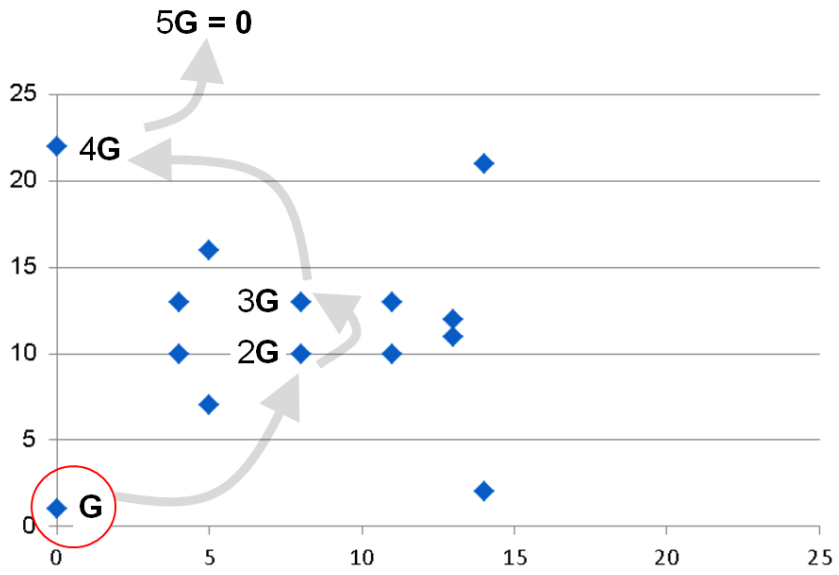

Figure 8: Not all points are generators.

But, if the order of the group is a prime number, then every point is a multiple of every other point except the point at infinity.

As an example, in **Figure 9** this curve has fifteen pairs of points, plus the point at infinity, making a total of 31, which is a prime number.

Hence, all points, with the exception of the point at infinity, are generators; generating a group of order 31, and there are no smaller sub-groups.

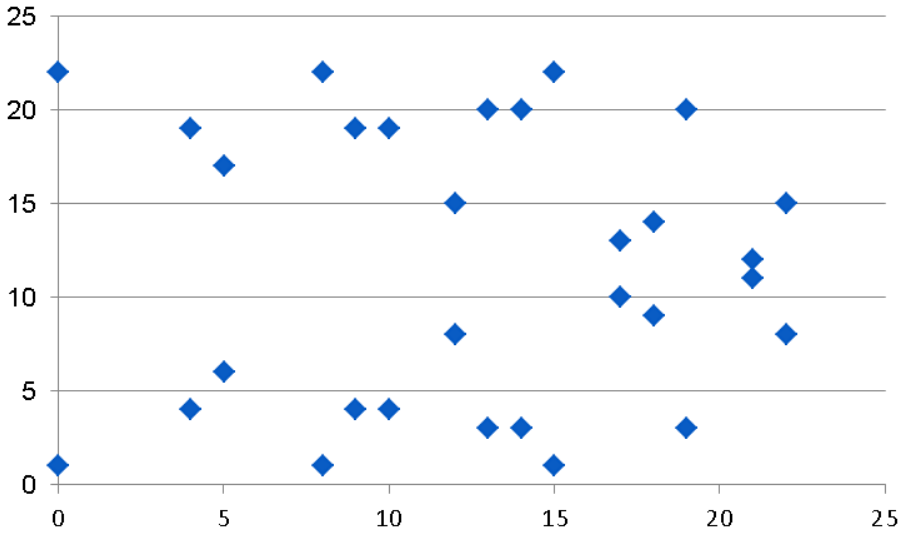

Figure 9: Points of the curve: $y^2 = x^3 + 5x + 1 \bmod 23$.

The Group Operation

When points are expressed as multiples of a generator, they can be added by adding the multiples modulo *q*. So, going back to our example from **Figure 6**, for instance:

11**G** + 3**G** = 14**G**

13**G** + 12**G** = 10**G**

See **Figure 10** and **Figure 11**.

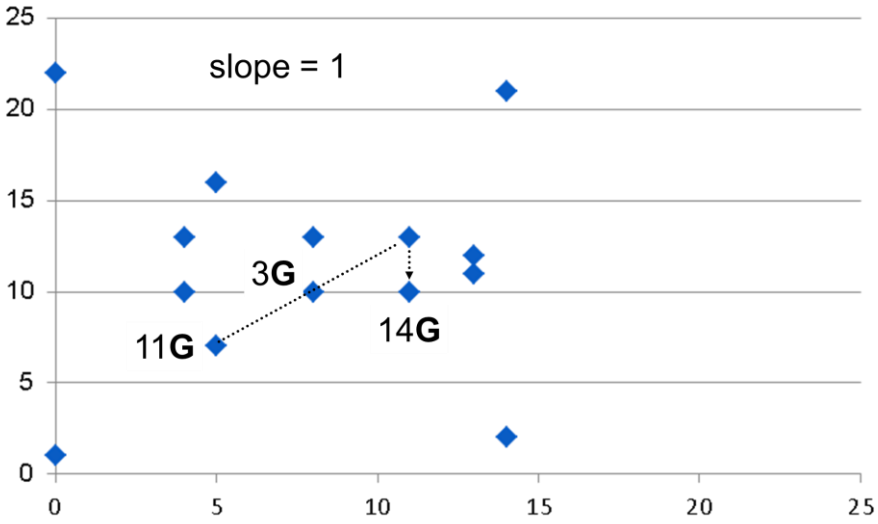

Figure 10: Point addition: 11**G** + 3**G** = 14**G.**

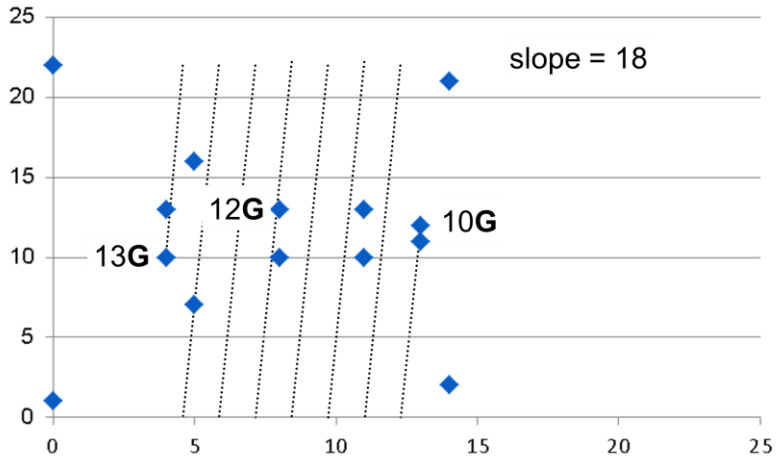

Figure 11: Point addition: 13**G** + 12**G** = 10**G.**

From these examples, we can see that point addition displays the 'distributive' property, in that

i**G** + j**G** = (i + j)**G**

The order of the group, *q*, is determined by the parameters of the curve and the field modulus, and it is the group order that determines the security of any cryptographic schemes based on the curve.

In our simple example, it would be possible to hold all of the points in computer memory, and perform operations on the points directly.

But, in real-world cryptographic applications, there are so many points that this isn't possible. Instead, the points have to be calculated as they are needed.

And because there are *q*-squared ways of choosing pairs of points, and only *q* possible results, many combinations produce the same result. For instance:

$8\mathbf{G} + 10\mathbf{G} = 3\mathbf{G}$, and
$6\mathbf{G} + 12\mathbf{G} = 3\mathbf{G}$

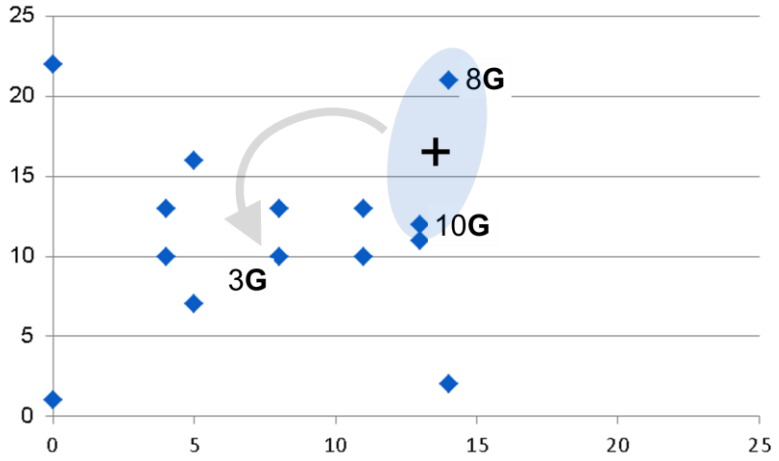

Figure 12: Point addition: 8**G** + 10**G** = 3**G.**

A point can also be doubled by adding it to itself:

$2 \times 9\mathbf{G} = 9\mathbf{G} + 9\mathbf{G} = 3\mathbf{G}$

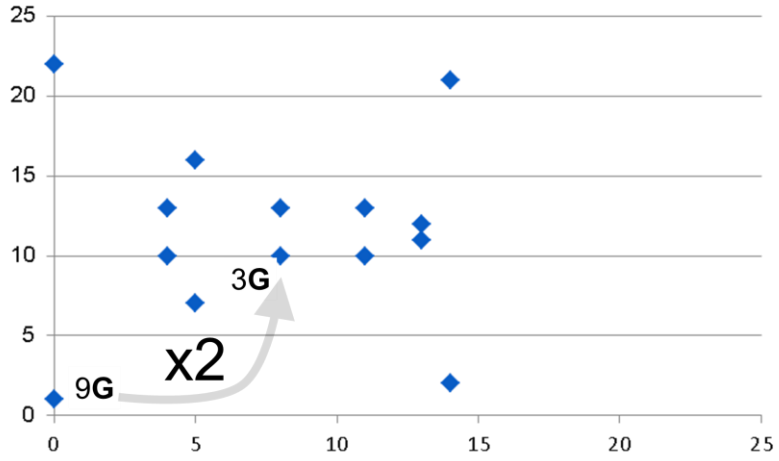

Figure 13: Point doubling: 2 x 9**G** = 3**G.**

Integer Multiplication of Elliptic Curve Points

The primitive operation used in all elliptic-curve public-key cryptographic applications is multiplication of a point by an integer.

This involves repeated application of the point-doubling and point-addition operations; the number of operations being determined by the number of bits in the binary representation of the integer multiplier. Each bit of the multiplier is processed separately and sequentially.

The outcome of processing each bit of the multiplier results from a doubling of the outcome of processing the previous bit and an optional addition of the base point, depending on the value of the current bit of the multiplier.

Representing *n* in binary form with *m+1* bits, then:

$$\mathbf{n = n_0 + n_1.2 + n_2.2^2 + ... + n_m.2^m}$$

$$\mathbf{where\ [n_0\ ..\ n_m]\ are\ bits\ \{0,1\}}$$

The algorithm for calculating **P** = n**G** is:

```
P := 0
for i from m to 0 do
   P := 2P
   if n_i == 1 then P := P + G
return P
```

If, for example, *n* were a 256-bit number, then this algorithm would require a maximum of 512 elliptic-curve operations, whereas the naïve approach of adding **G** to itself *n-1* times would require (at least) an impossible 2^{255}-1 operations.

Because the "if" statement depends on the value of one bit from *n*, the uncertainty in the path leading to the final result is 2^m. For *m* of sufficient size, it becomes impossible in practice to figure out the value of *n* required to multiply **G** in order to obtain **P**. Attempting to calculate *n* from **P** and **G** is known as the elliptic-curve discrete logarithm problem.

Real-World Elliptic-Curve Cryptography

Of course, the example curve we have been using, with $a = 3$ and $b = 1$, is too trivial for use in real cryptographic applications.

For comparison, here is the parameter set for a real-world cryptographic elliptic-curve: the P-256 curve specified by NIST.

a = -3
b = 5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b
p = 115792089210356248762697446949407573530086143415290314195533631308867097853951

The *x* and *y* coordinates of the base point are:

gx = 6b17d1f2 e12c4247 f8bce6e5 77037d81 2deb33a0 f4a13945 d898c296
gy = 4fe342e2 fe1a7f9b 8ee7eb4a 2bce3357 6b315ece cbb64068 37bf51f5

p is expressed as a decimal number, whereas *a*, *b* and **G** are expressed in hexadecimal.

In order to ensure that the designers have not deliberately specified a curve with a hard-to-detect weakness, a known one-way function was used in the calculation of the curve parameter *b*, and the input to the function has been published as part of the design documentation.

In this way, the designers were not able to fix the parameter directly in order to incorporate a weakness.

One-Way Function Based on Elliptic Curves

From the foregoing we see how points on an elliptic-curve with coordinates over a field modulo a prime, *p*, can form a group, and how a group can be used to construct a one-way function suitable for use in cryptographic applications.

Graphically, we can depict the one way function as shown in **Figure 17**. Here, the thick lines represent curve points, and thin lines represent integers.

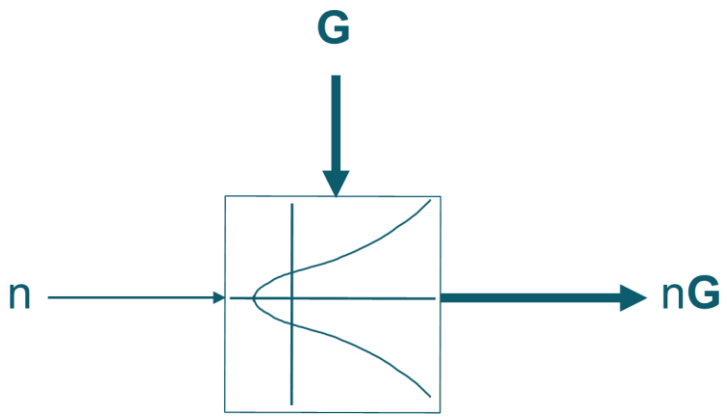

Figure 14: One-way function

While it is straightforward to calculate n**G**, given *n* and **G**, it is computationally infeasible to find *n*, given **G** and n**G**. These are just the properties we need in a practical cryptographic primitive with which we can build strong cryptographic mechanisms.

Entrust & You

More than ever, Entrust understands your organization's security pain points. Whether it's the protection of information, securing online customers, regulatory compliance or large-scale government projects,

Entrust provides identity-based security solutions that are not only proven in real-world environments, but cost-effective in today's uncertain economic climate.

Entrust offers software authentication platforms that strengthen security in a wide range of identity and transaction ecosystems. Government agencies, financial institutions and other enterprises rely on Entrust solutions to strengthen trust and reduce complexity for consumers, citizens and employees.

Now, as part of Datacard Group, Entrust offers an expanded portfolio of solutions across more than 150 countries. Together, Datacard Group and Entrust issue more than 10 million secure identities every day, manage billions of secure transactions annually and issue a majority of the world's financial cards.

For more information about Entrust products and services, call **888-690-2424**, email **entrust@entrust.com** or visit **entrust.com**.

Company Facts

Website: www.entrust.com
Employees: 359
Customers: 5,000
Offices: 10 Globally

Headquarters

Three Lincoln Centre
5430 LBJ Freeway, Suite 1250
Dallas, Texas 75240

Sales

North America: 1-888-690-2424
EMEA: +44 (0) 118 953 3000
Email: entrust@entrust.com