Lecture 09 - ECDSA - Elliptic curve public key encryption

Just one set of lectures this week - 5 parts.

Videos (part 1 .. 5)

Some history.

The biggest problem in cryptography before 1977 was key distribution.

In 1977 public key cryptography was invented. IN 1983 it was patented. In 1988 an adequate key was 100 decimal digits. Computers were a touch slower - today you need 8k to 16k digits. That is slow.

The system really only is used to solve the key distribution problem - not actually to encrypt content.

This is the simple form: A and B both have Public/Private key pairs. B wants to communicate with A. B uses A's public key to encrypt a message 32 bytes long to send to A. The message is a random 32 byte AES encryption key. A uses the "private" part of the key to decrypt the message and get the key. A uses B's private key to send back a new message with 32 bytes of data. The 32 bytes are a random AES key. B uses her own private key to decrypt the message. Now both sizes have AES keys. They use AES, 1000 times faster, to send back and forth encrypted data.

A real key exchange is more complex than this - but this is the basic idea.

There are also key exchanges where the key is never transmitted. That is better for key exchange. It requires the use of a zero knowledge proof combined with a bunch of algebra and no key transmission is done at all. Both sides end up with the same a 32 byte AES key.

Also to make this secure you have to add a bunch of wrappers around the transmission like a Nonce and signature of validity like CBC.

The idea is the same. All the real data is encrypted using AES, the key exchange is done using the public/private key pairs.

This is the basic idea of TLS 1.3, the underlying system for HTTPS. TLS adds the additional validation that the keys are checked in a hierarchy with a signature where the signature can be validated to be from a certain source. This is the “hierarchy of Trust” and why we trust the TLS to be from the original domain.

RSA is easy to understand.

RSA even has articles on it in the Atlantic Magazine.

You can get it down to a simple set of explanations with just some math.

The security is based on factoring of large numbers. *Mathematicians consider factoring to be a difficult* ` *problem*. I find the general understatement of ‘difficult to be very funny.

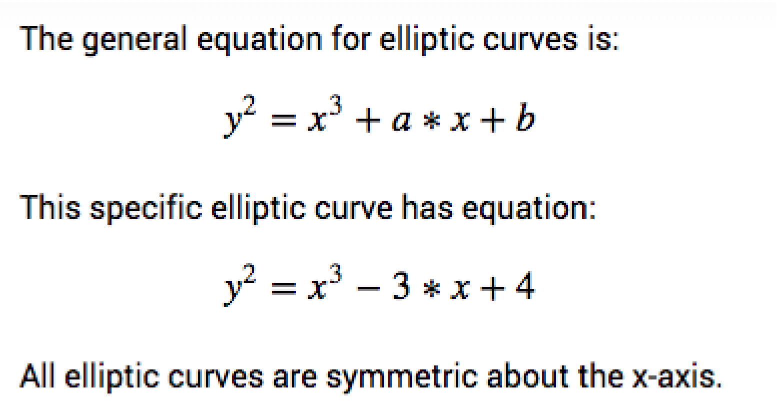
Another good source [is Cloudflare’s primer on EC]https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/) - and I have taken the 2 animations from them.

EC is used as the system in Bitcoin, Ethereum and most other blockchains.

EC and ECDSA Encryption

How ECDSA works under the covers

Eliptic Curve Function



The general equation for elliptic curves is:

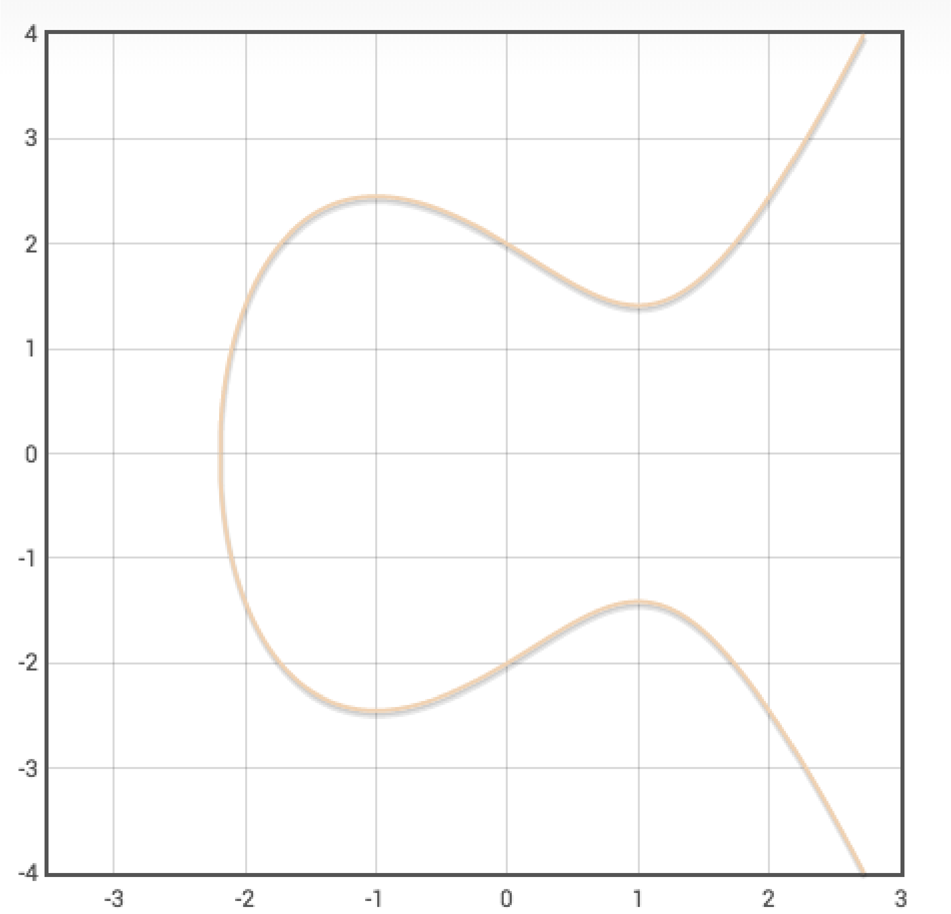
$$y^2 = x^3 + a * x + b$$

This specific elliptic curve has equation:

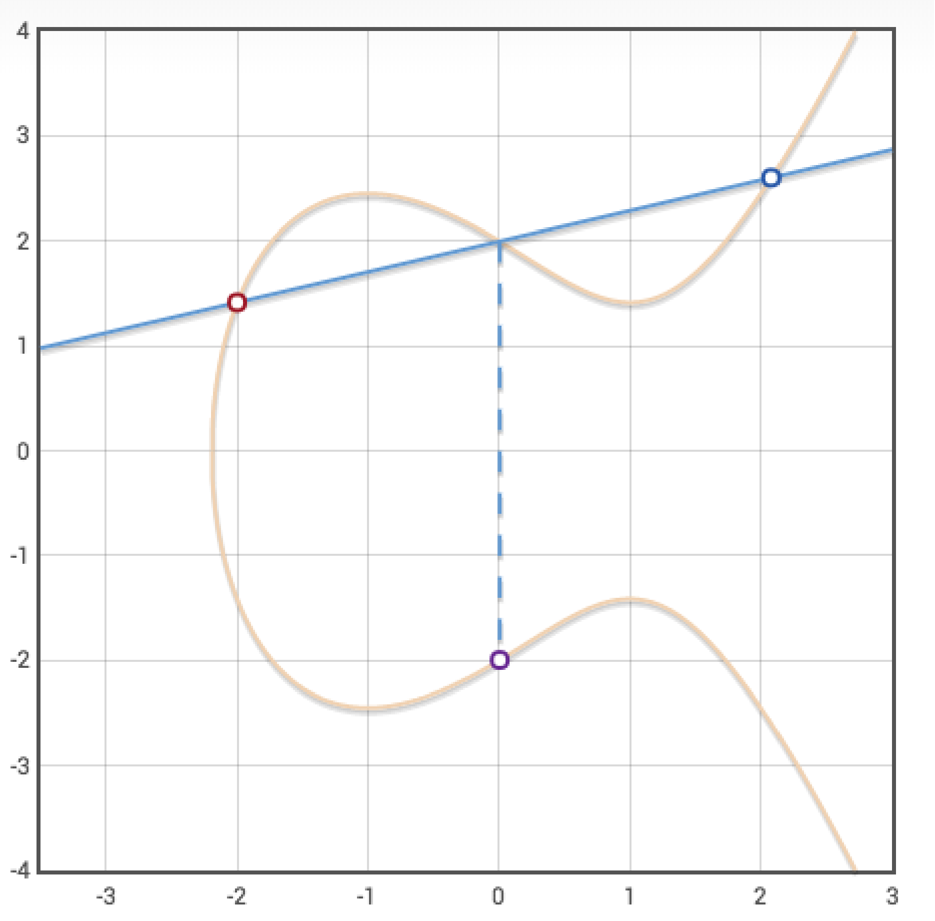
$$y^2 = x^3 - 3 * x + 4$$

All elliptic curves are symmetric about the x-axis.

Graphed



Good EC, bad EC, addition.

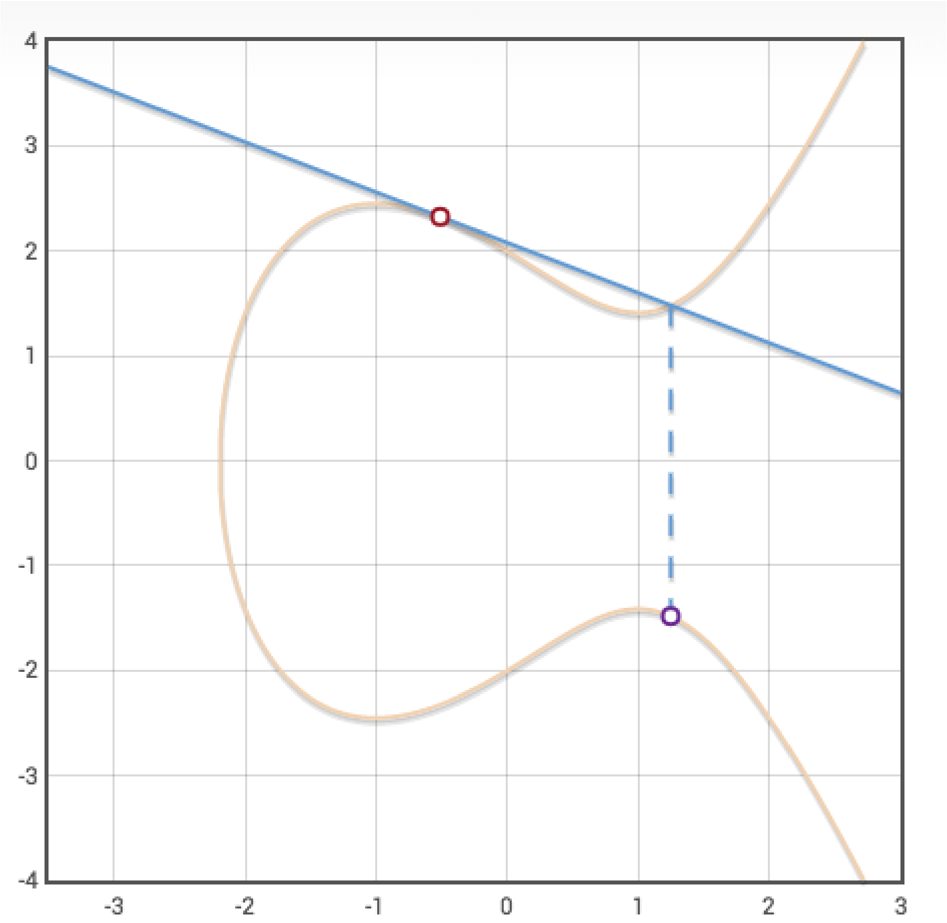


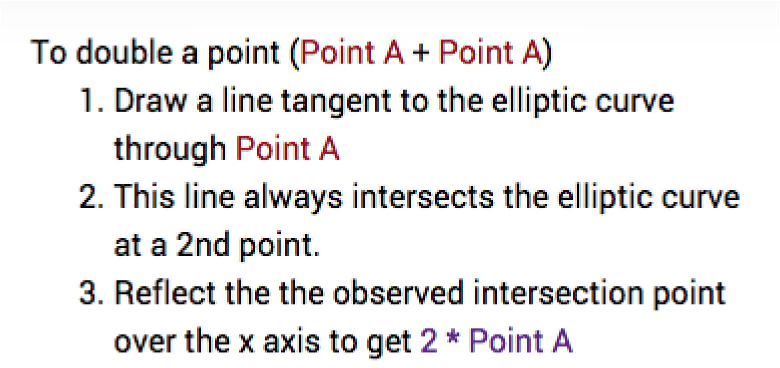
To add 2 points (Point A, and Point B)

1. Draw a line between Point A and Point B
2. This line always intersects the elliptic curve at a 3rd point.
3. Reflect the the observed intersection point over the x axis to get the sum of Point A and Point B

$$(-2.0, 1.4) + (2.0, 2.5) = (0.0, -1.9)$$

Doubleling of a value.

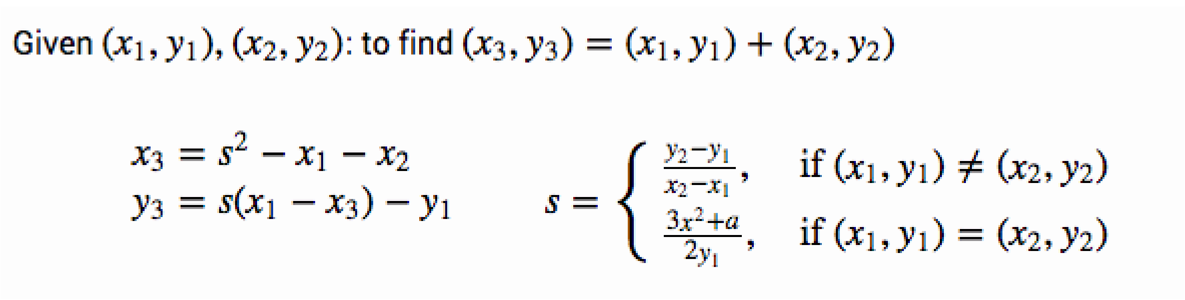




To double a point (Point A + Point A)

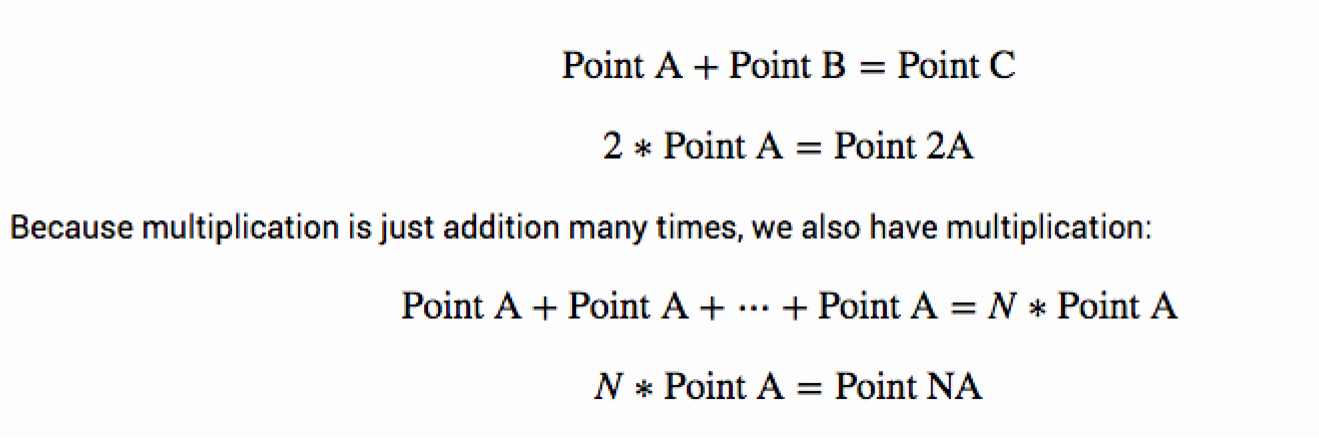
1. Draw a line tangent to the elliptic curve through Point A
2. This line always intersects the elliptic curve at a 2nd point.
3. Reflect the the observed intersection point over the x axis to get 2 * Point A

2 * (-0.5, 2.3) = (1.2, -1.4)



Given $(x_1, y_1), (x_2, y_2)$: to find $(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$

$$\begin{aligned} x_3 &= s^2 - x_1 - x_2 \\ y_3 &= s(x_1 - x_3) - y_1 \end{aligned} \qquad s = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1}, & \text{if } (x_1, y_1) \neq (x_2, y_2) \\ \frac{3x^2 + a}{2y_1}, & \text{if } (x_1, y_1) = (x_2, y_2) \end{cases}$$



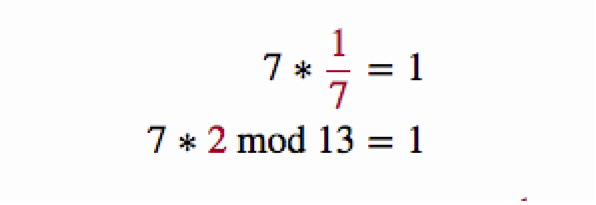
$$\text{Point A} + \text{Point B} = \text{Point C}$$

$$2 * \text{Point A} = \text{Point 2A}$$

Because multiplication is just addition many times, we also have multiplication:

$$\text{Point A} + \text{Point A} + \cdots + \text{Point A} = N * \text{Point A}$$

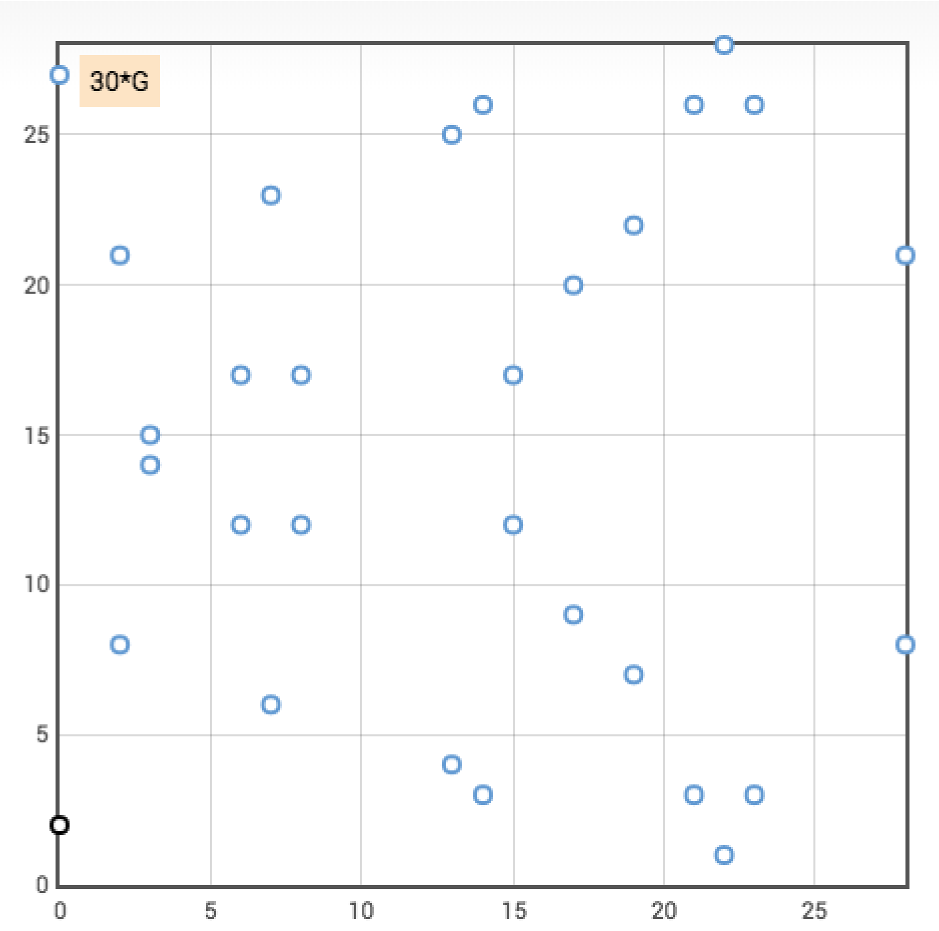
$$N * \text{Point A} = \text{Point NA}$$



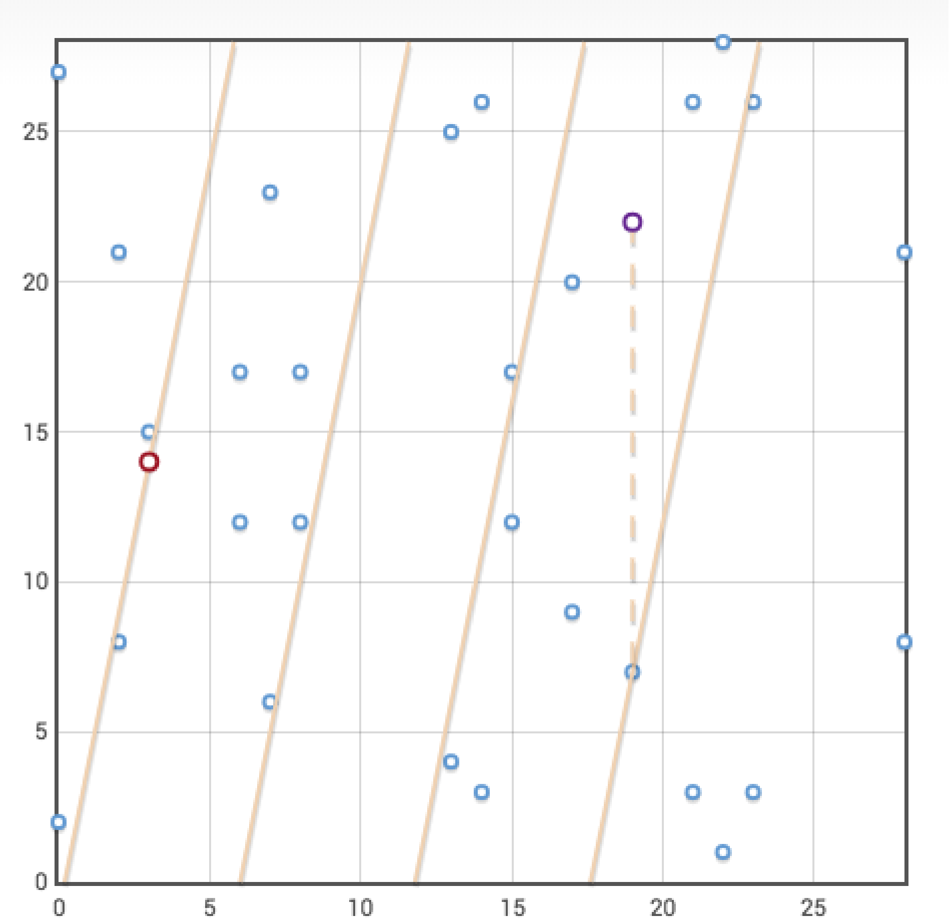
$$7 * \frac{1}{7} = 1$$

$$7 * 2 \bmod 13 = 1$$

As integers we get:



Now we can use a modulo system for this:

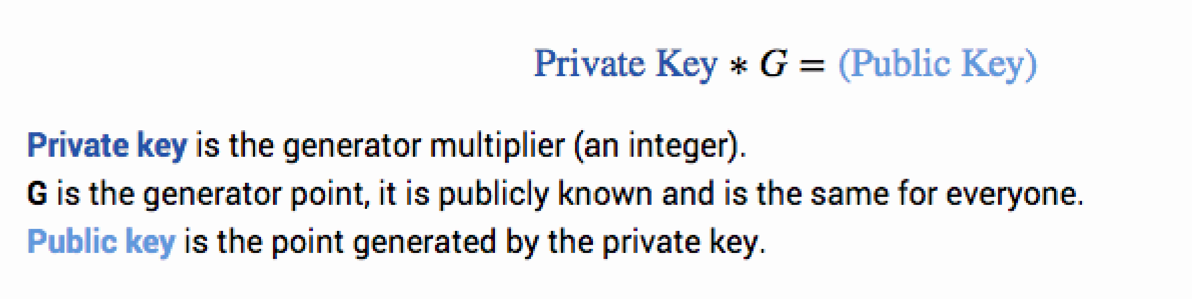


$$2 * (3, 14) = (19, 22)$$

$$(3, 14) = 21 * \mathrm{G}$$

$$(19, 22) = 11 * \mathrm{G}$$

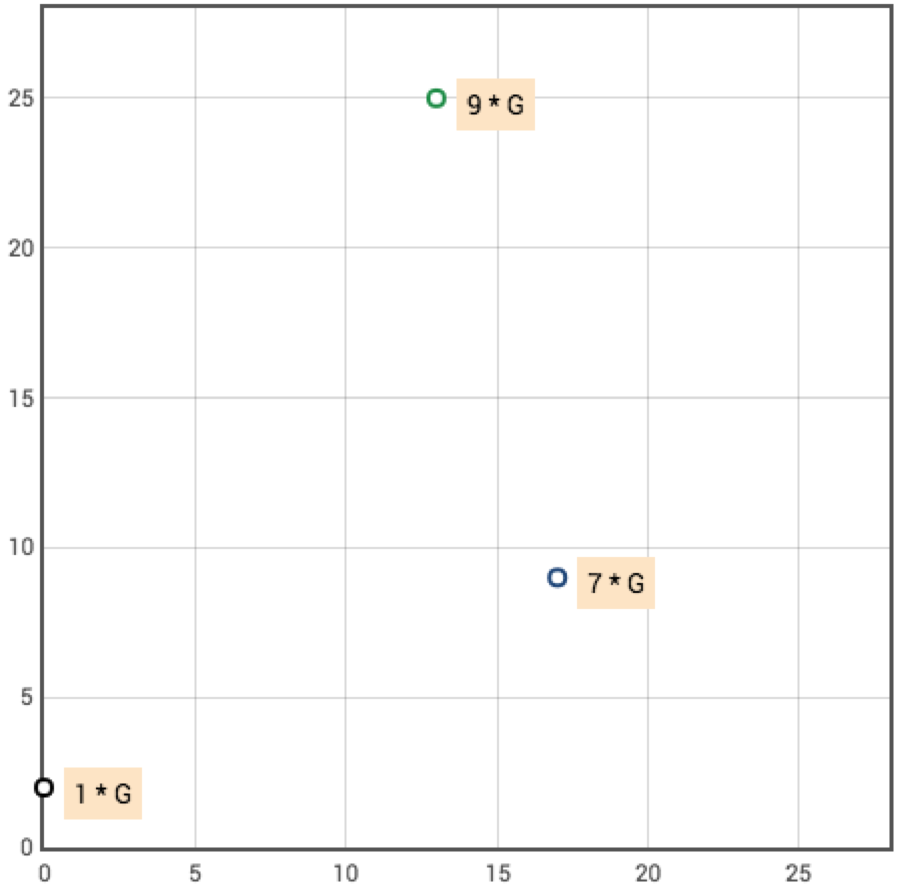
$$11 = 2 * 21 \bmod 31$$



$$\text{Private Key} * G = (\text{Public Key})$$

Private key is the generator multiplier (an integer).
G is the generator point, it is publicly known and is the same for everyone.
Public key is the point generated by the private key.

The Signer



The signer knows:

Generator:	1 * G = (0, 2)
Private Key:	7 * G = (17, 9)
Random Point:	9 * G = (13, 25)
Message Hash:	14

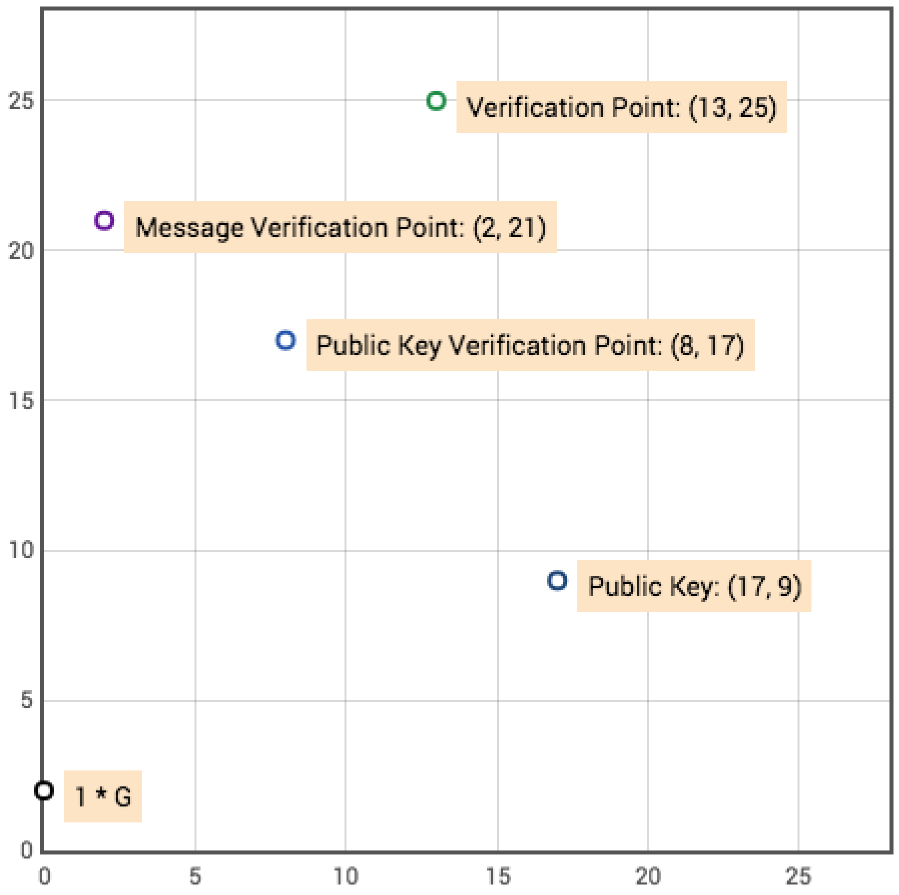
Signature Factor:

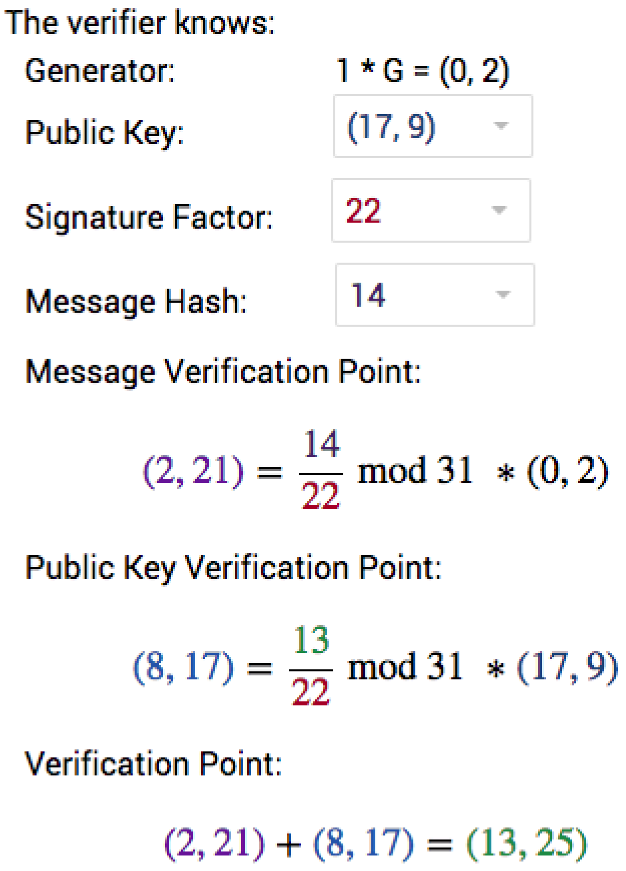
$$22 = \frac{14 + 13 * 7}{9} \bmod 31$$

The Signature

22, 13

The Verifier





The verifier knows:

Generator: 1 * G = (0, 2)

Public Key: (17, 9)

Signature Factor: 22

Message Hash: 14

Message Verification Point:

$$(2, 21) = \frac{14}{22} \bmod 31 \ * (0, 2)$$

Public Key Verification Point:

$$(8, 17) = \frac{13}{22} \bmod 31 \ * (17, 9)$$

Verification Point:

$$(2, 21) + (8, 17) = (13, 25)$$