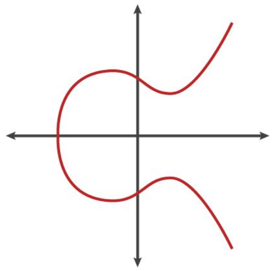
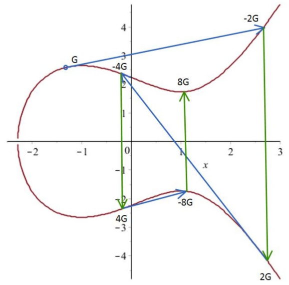
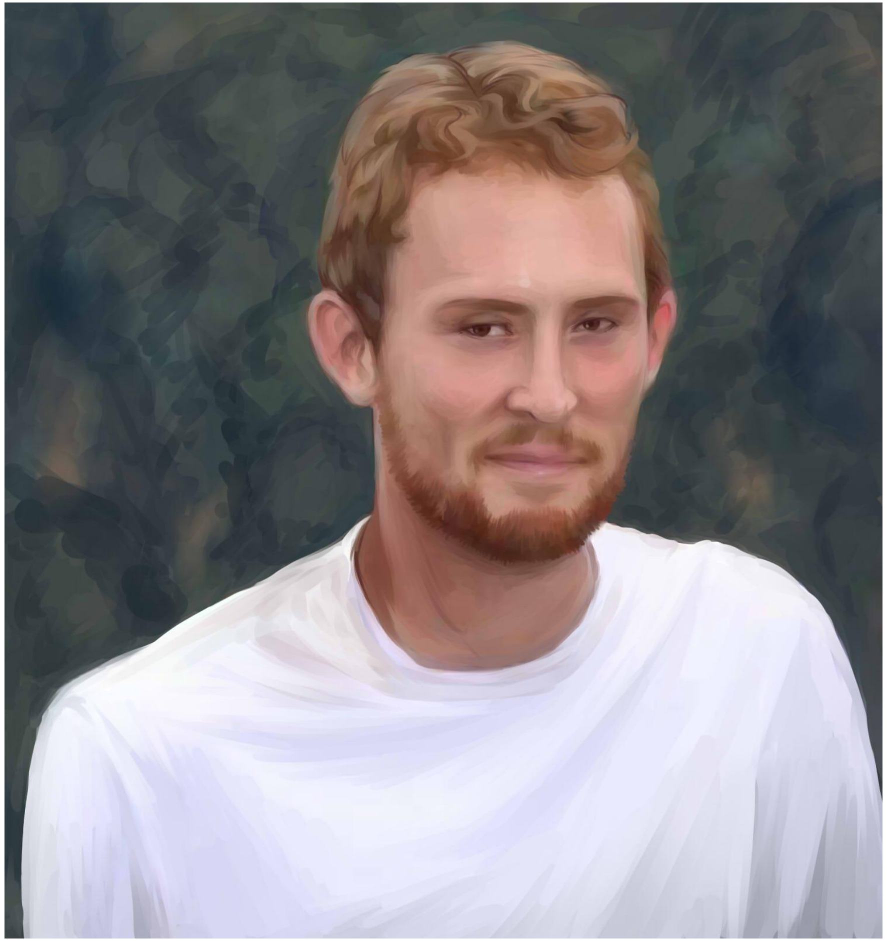
Elliptic Curve Cryptography and Bitcoin



Alex Melville



5F11 78CD D43A 49E9 10D6 D27C 773A E36E 3704 569C



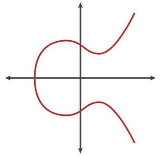
Alex Melville

Software Engineer
@BitGo

World Traveler

github.com/Melvillian

Why I’m talking about ECC

- First attempt at explaining it failed
- Bitcoin applications
- Pictures! 
- Crypto is amazingly powerful*

* … when implemented and used correctly!

The Plan

1. **Crypto Basics**
    a. Public/Private Key Digital Signatures
    b. “Hard” problems and “Easy” problems
    c. Discrete Logarithm Problem

The Plan

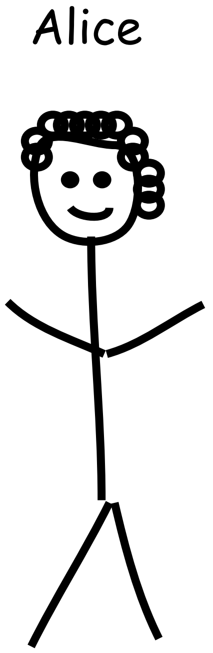
1. **Crypto Basics**
    a. Public/Private Key Digital Signatures
    b. “Hard” problems and “Easy” problems
    c. Discrete Logarithm Problem
2. **Elliptic Curves**
    a. EC Points
    b. Point Addition
    c. Finite Fields

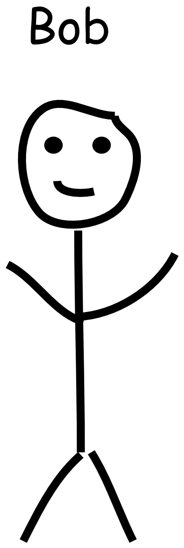
The Plan

1. **Crypto Basics**
   a. Public/Private Key Digital Signatures
   b. “Hard” problems and “Easy” problems
   c. Discrete Logarithm Problem
2. **Elliptic Curves**
   a. EC Points
   b. Point Addition
   c. Finite Fields
3. **ECDSA in Bitcoin**
   a. ECDSA signature
   b. 2013 Android Bitcoin Wallet screwup
   c. Segwit (Fixing Transaction Malleability)

See references at the end for more!

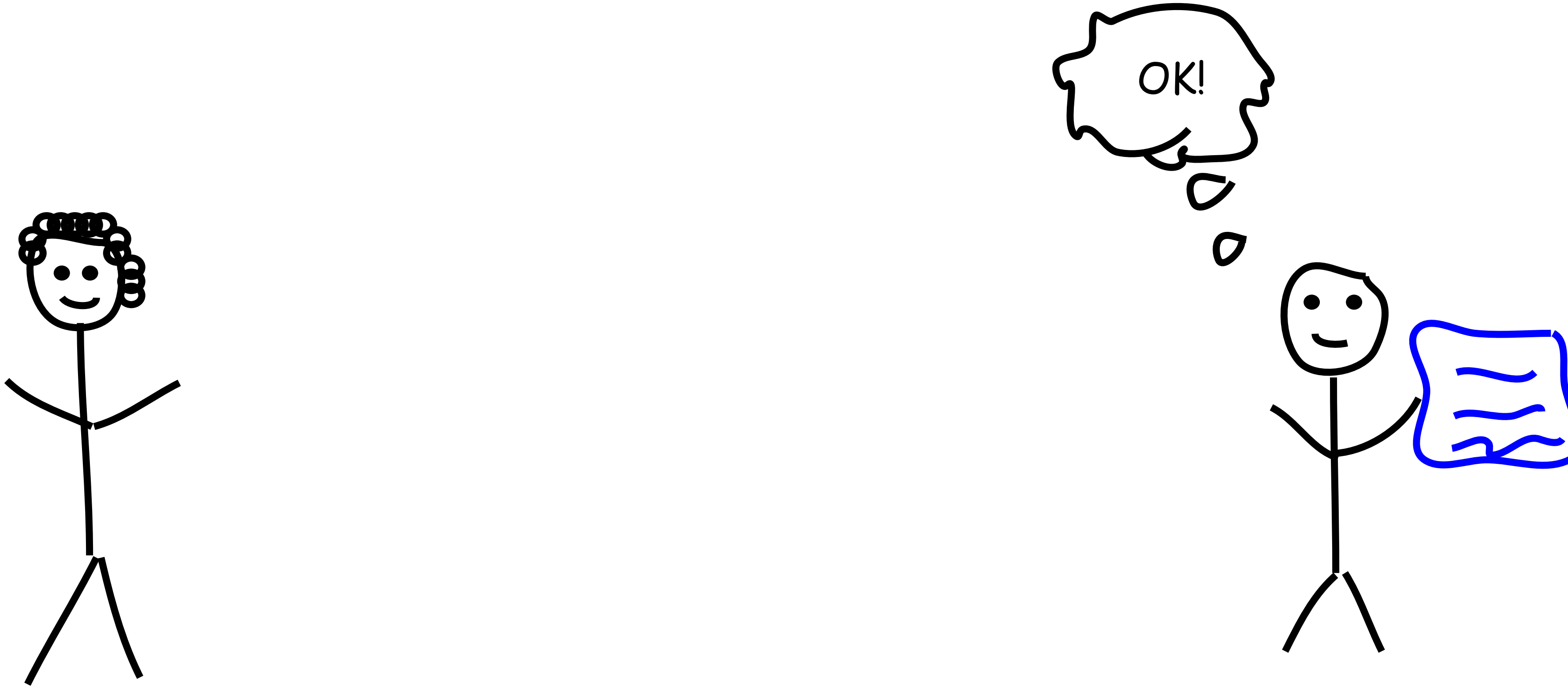
Alice & Bob



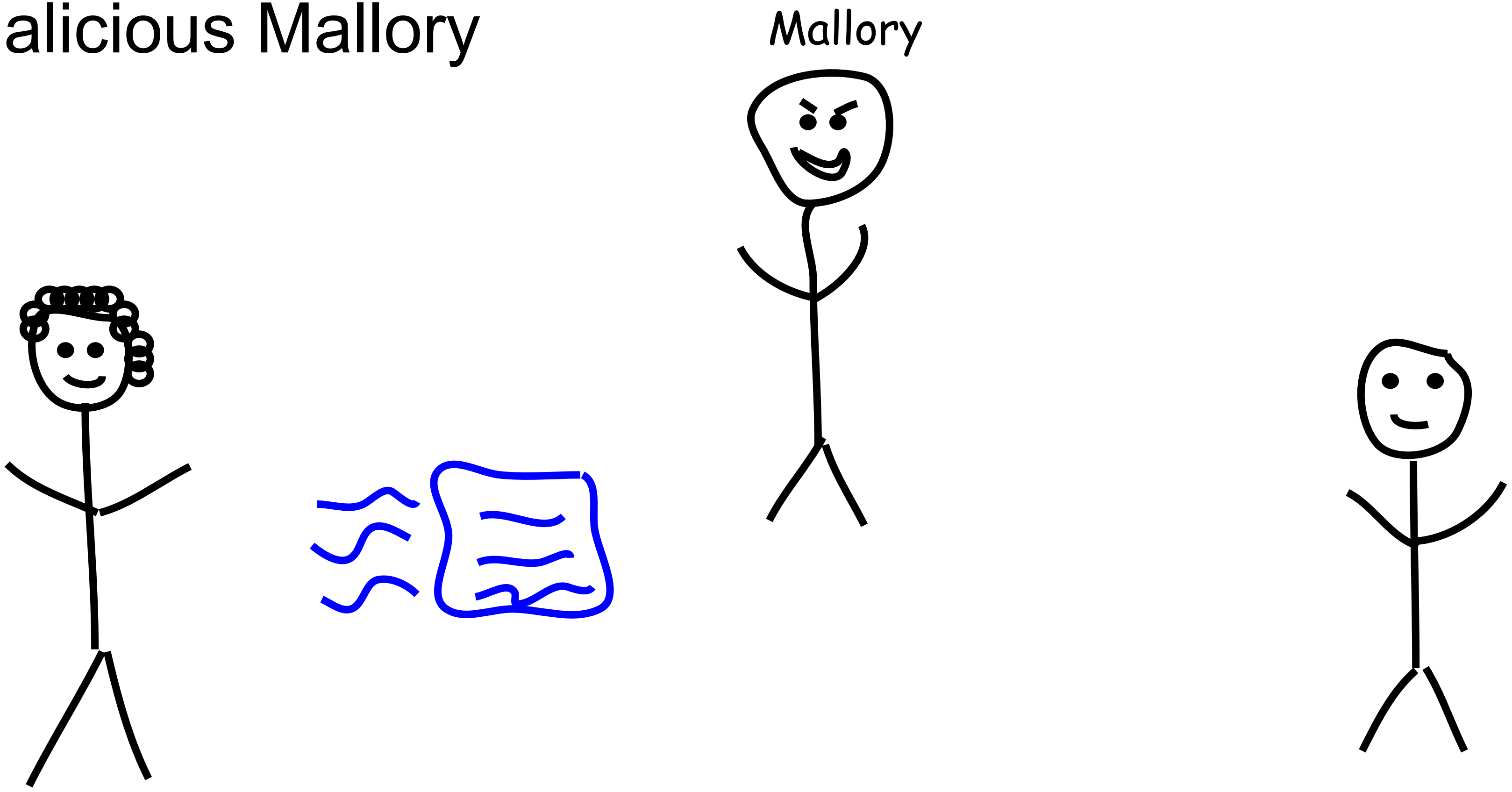


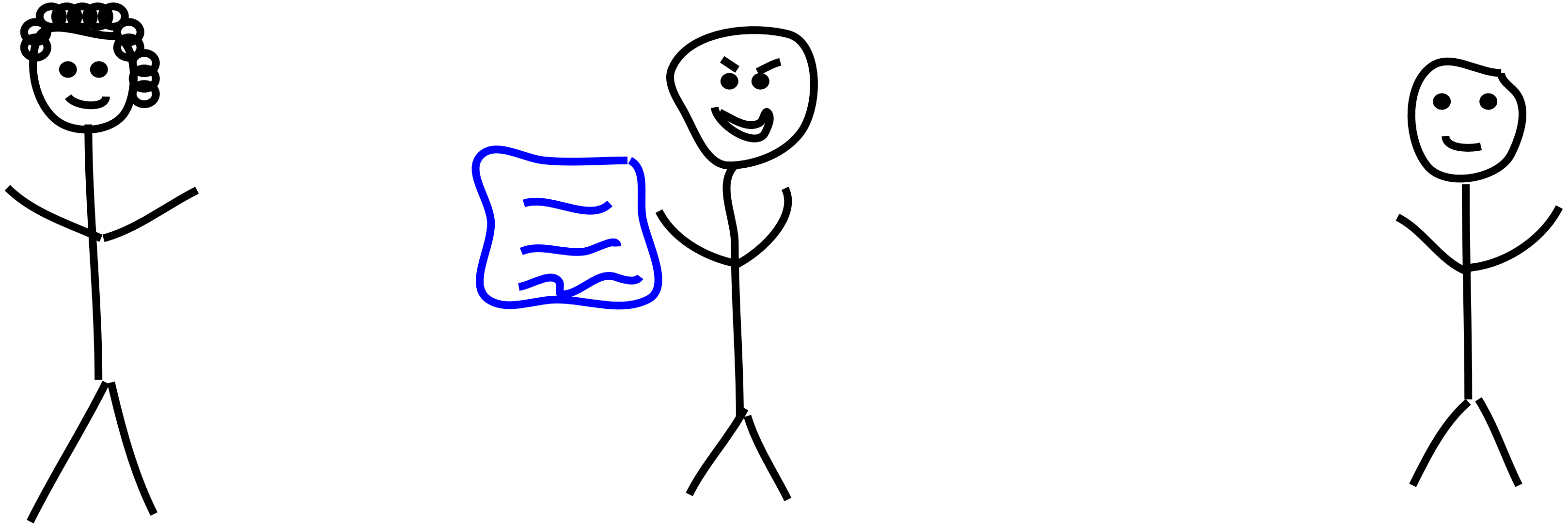
Alice wants to send an invoice to Bob

Pay rent to
account
#79BE667E

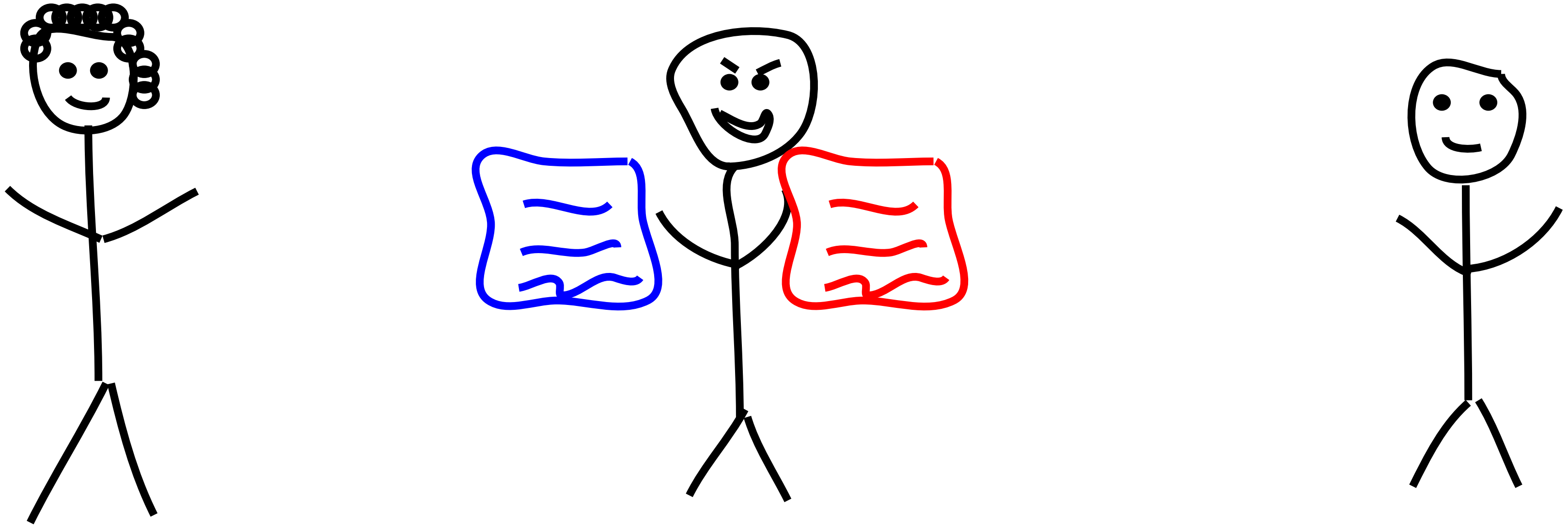

OK!

Malicious Mallory

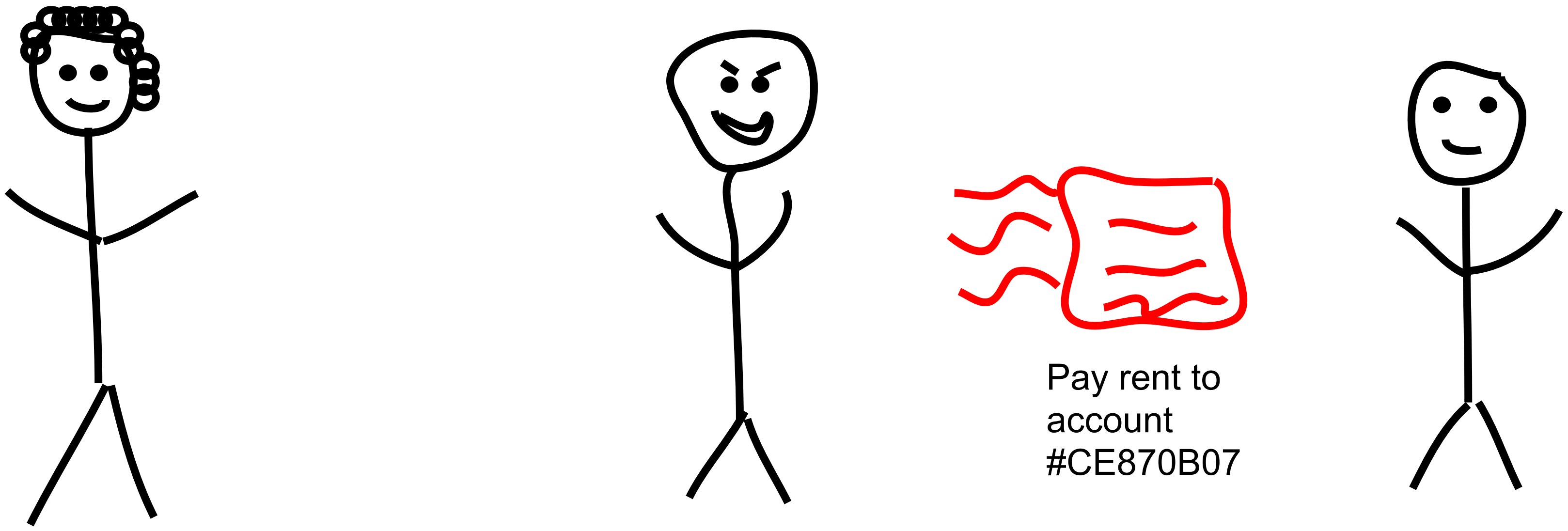


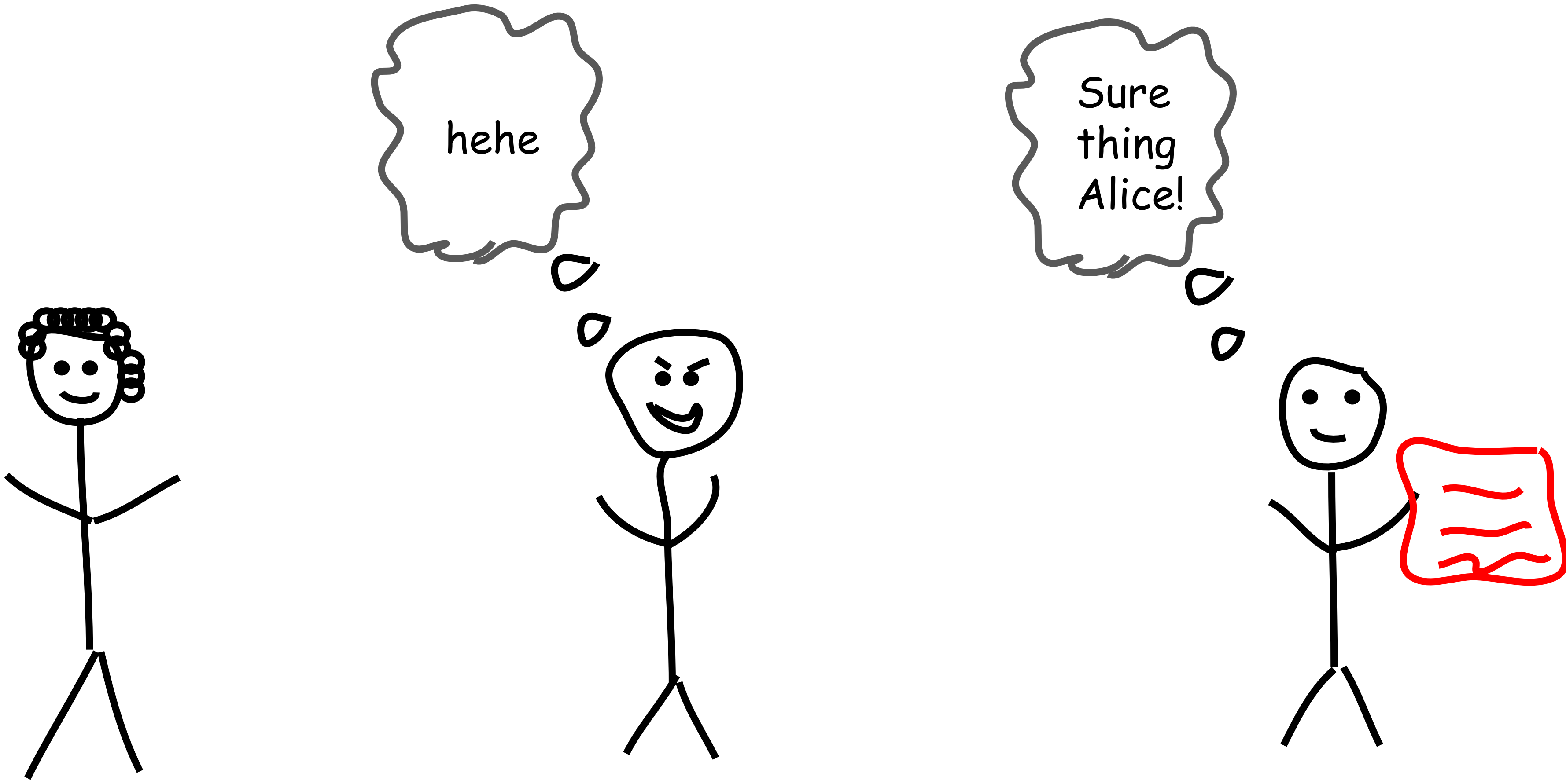


Mallory replace Alice’s message with her own



Mallory replace Alice’s message with her own




hehe
Sure thing Alice!

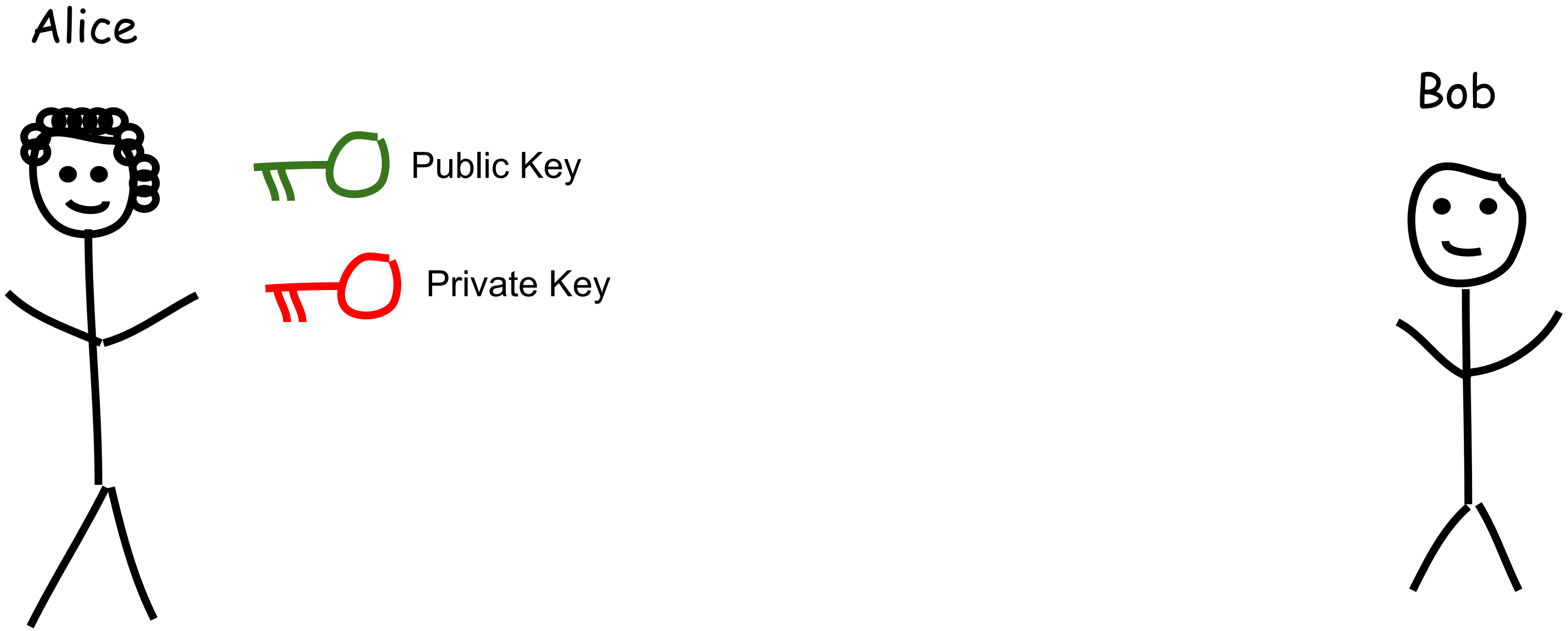
Analog Signatures

[illegible]

Digital Signatures

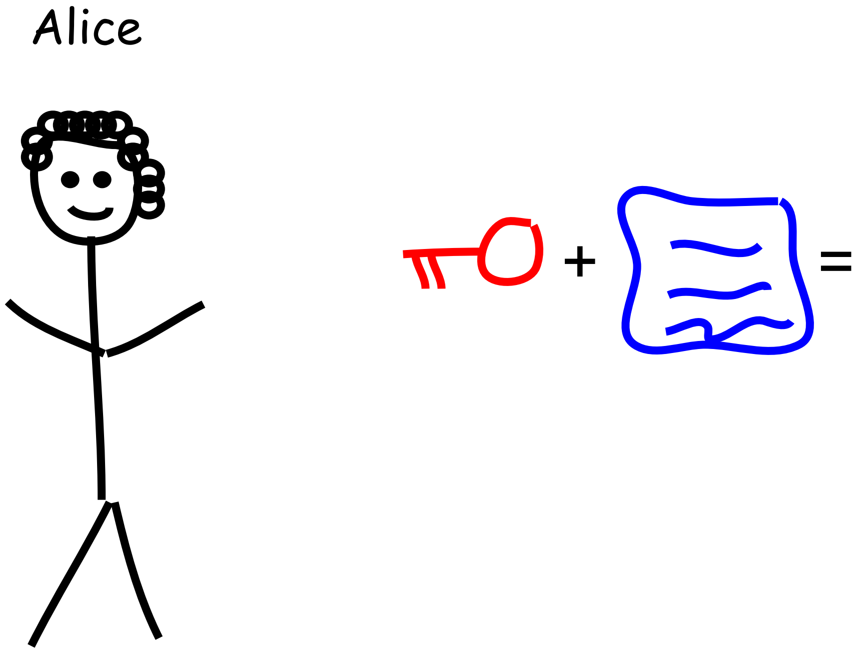
```
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1

iJwEAQEKAAYFAlRGkIcACgkQU805K6
3BbbvgkQP/cJktaCbNQtxCfV/ZXIiwn6Mv
tVELtCdcF/JWKD/1BPGaKXT6BiVa6vrB
6dOwRWqUGiZbV1VWkj/LglaMqPa1ZEn
Z
Bwpux8hyUYRNbjnyVSDYCyyBH/qvhE/9
wGgeLRJ5eK/Na6QoKw4XDAo2RHoiBF
3o
wwm6vk4PZF8DacCv64o=
=SadA
-----END PGP SIGNATURE-----
```

Public Private Key Cryptography



Public Private Key Cryptography

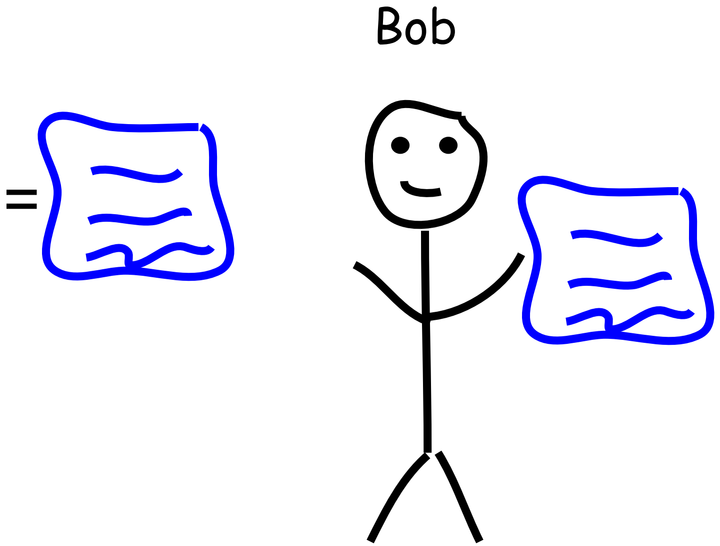
Public Private Key Cryptography



```
-----BEGIN PGP
SIGNATURE-----
Version: GnuPG v1

iJwEAQEKAAYFAlRGkH
…..gHFLn+Lw1x6LUroOj
kl2zjpoCB
6pmQPd09MglBXJfnrBI=
=ET9V
-----END PGP
SIGNATURE-----
```

Use the public key to verify the message

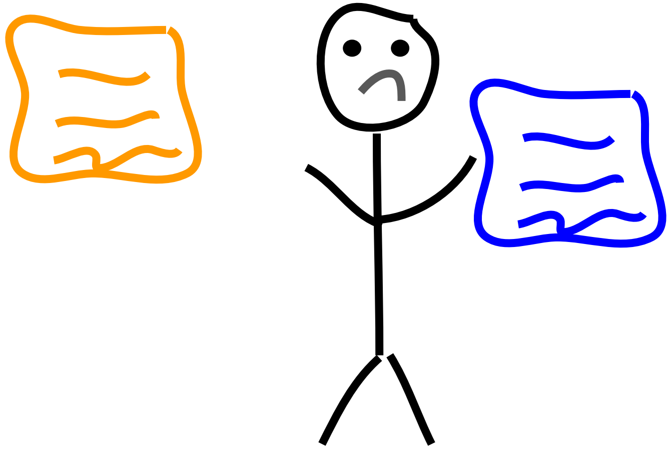
```
-----BEGIN PGP
SIGNATURE-----
Version: GnuPG v1

iJwEAQEKAAYFAlRGkH
…..gHFLn+Lw1x6LUroOj
kl2zjpoCB
6pmQPd09MglBXJfnrBI=
=ET9V
-----END PGP
SIGNATURE-----
```



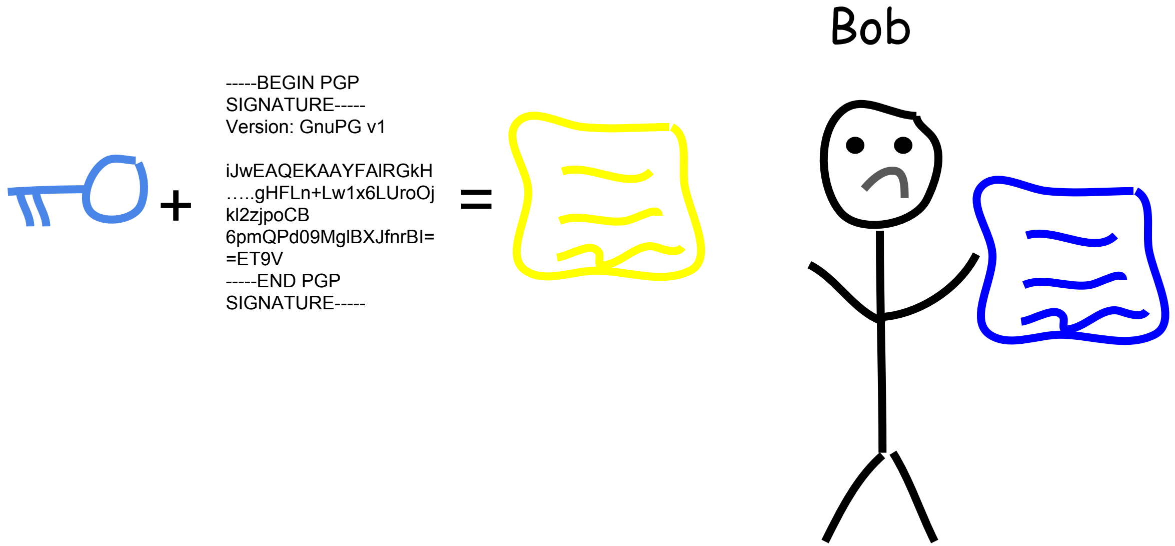
Different public key *will not* verify the message

```
-----BEGIN PGP PUBLIC
KEY BLOCK-----
Version: 2.6.i

mQCNAi+UeBsAAAEEA
MP0kXU75GQdzwwlMiw
….kYboAFx
xHg43Cnj60OeZG2PKp/k
U91ipOJP1cs8/xYOGkeo
AMqDfwPeFlkBiA==
=ddBN
-----END PGP PUBLIC
KEY BLOCK-----
```

=



Different public key *will not* verify the message



```
-----BEGIN PGP
SIGNATURE-----
Version: GnuPG v1

iJwEAQEKAAYFAIRGkH
…..gHFLn+Lw1x6LUroOj
kl2zjpoCB
6pmQPd09MgIBXJfnrBI=
=ET9V
-----END PGP
SIGNATURE-----
```

How to build a Digital Signature Algorithm?

$y = x + 2$

$$
\begin{aligned}
&R^{(2)}_{6,WL}(u_1,u_2,u_3) = \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(H.1)}\\
&\frac{1}{24}\pi^2 G\left(\frac{1}{1-u_1},\frac{u_2-1}{u_1+u_2-1};1\right)+\frac{1}{24}\pi^2 G\left(\frac{1}{u_1},\frac{1}{u_1+u_2};1\right)+\frac{1}{24}\pi^2 G\left(\frac{1}{u_1},\frac{1}{u_1+u_3};1\right)+\\
&\frac{1}{24}\pi^2 G\left(\frac{1}{1-u_2},\frac{u_3-1}{u_2+u_3-1};1\right)+\frac{1}{24}\pi^2 G\left(\frac{1}{u_2},\frac{1}{u_1+u_2};1\right)+\frac{1}{24}\pi^2 G\left(\frac{1}{u_2},\frac{1}{u_2+u_3};1\right)+\\
&\frac{1}{24}\pi^2 G\left(\frac{1}{1-u_3},\frac{u_1-1}{u_1+u_3-1};1\right)+\frac{1}{24}\pi^2 G\left(\frac{1}{u_3},\frac{1}{u_1+u_3};1\right)+\frac{1}{24}\pi^2 G\left(\frac{1}{u_3},\frac{1}{u_2+u_3};1\right)+\\
&\frac{3}{2}G\left(0,0,\frac{1}{u_1},\frac{1}{u_1+u_2};1\right)+\frac{3}{2}G\left(0,0,\frac{1}{u_1},\frac{1}{u_1+u_3};1\right)+\frac{3}{2}G\left(0,0,\frac{1}{u_2},\frac{1}{u_1+u_2};1\right)+
\end{aligned}
$$

“Easy” Problems

Discrete Logarithm Problem

h = G^x (mod n)

what is x?

“Hard” Problems

These numbers are huuuuuge

- 2^256, 256 bits (1 followed by 77 0’s)
  - 100000000000000000000000000000000000000000000000000000000000000000000000000
    000
- We cannot imagine the size of this number
  - Age of Universe (10^10 years)
  - Number of atoms in galaxy (10^67)
  - Trillion computers doing a trillion computation every trillionth of a second (< 10^56)

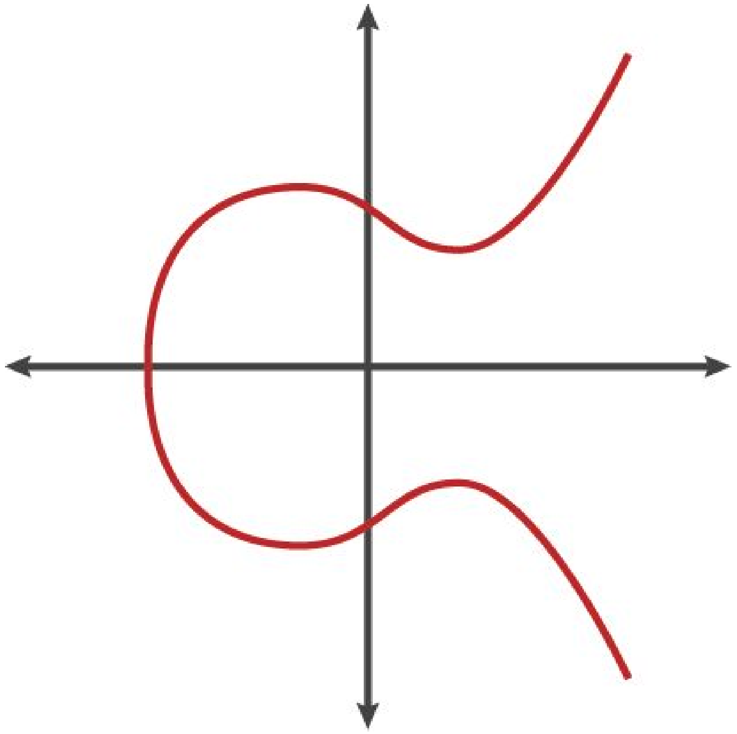
Discrete Logarithm Problem Difficulty

Unproven!

Break...

1. Crypto Basics
    a. Public/Private Key Digital Signatures
    b. “Hard” problems and “Easy” problems
    c. Discrete Logarithm Problem

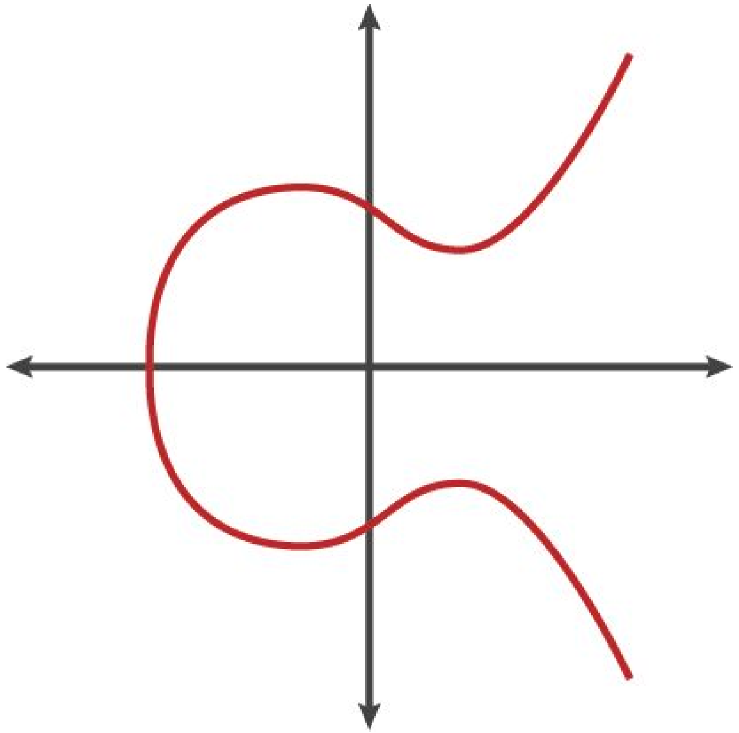
Elliptic Curves + DLP = Digital Signatures



What is an Elliptic Curve?

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

- Diophantine Equations
- Addition (P + Q)
- Multiplication (k * Q)



Adding P + Q

Let (x_1, y_1), (x_2, y_2) and (x_3, y_3) denote the coordinates of P, Q and $P+Q$ respectively. Suppose $P \neq Q$. We want to express x_3 and y_3 in terms of x_1, y_1, x_2, y_2. Let $y = \alpha x + \beta$ be the equation of the line through P and Q. Then,

$$\alpha = \frac{(y_2 - y_1)}{(x_2 - x_1)} \text{ and } \beta = y_1 - \alpha x_1.$$

A point on the line l i.e. a point $(x, \alpha x + \beta)$, lies on the elliptic curve if and only if

$$(\alpha x + \beta)^2 = x^3 + ax + b.$$

Hence, there is one intersection point for each root of the cubic equation

$$x^3 - (\alpha x + \beta)^2 + ax + b.$$

$$x_3 = \alpha^2 - x_1 - x_2.$$

This leads to an expression for x_3, and hence $P + Q = (x_3, -(\alpha x_3 + \beta))$, in terms of x_1, x_2, y_1, y_2:

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

Adding P + Q

Pictures!

Let (x_1, y_1), (x_2, y_2) and (x_3, y_3) denote the coordinates of P, Q and $P + Q$ respectively. Suppose $P \neq Q$. We want to express x_3 and y_3 in terms of x_1, y_1, x_2, y_2. Let $y = \alpha x + \beta$ be the equation of the line through P and Q. Then,

$$\alpha = \frac{(y_2 - y_1)}{(x_2 - x_1)} \text{ and } \beta = y_1 - \alpha x_1.$$

A point on the line l i.e. a point $(x, \alpha x + \beta)$ lies on the elliptic curve if and only if

$$(\alpha x + \beta)^2 = x^3 + ax + b.$$

Hence, there is one intersection point for each root of the cubic equation

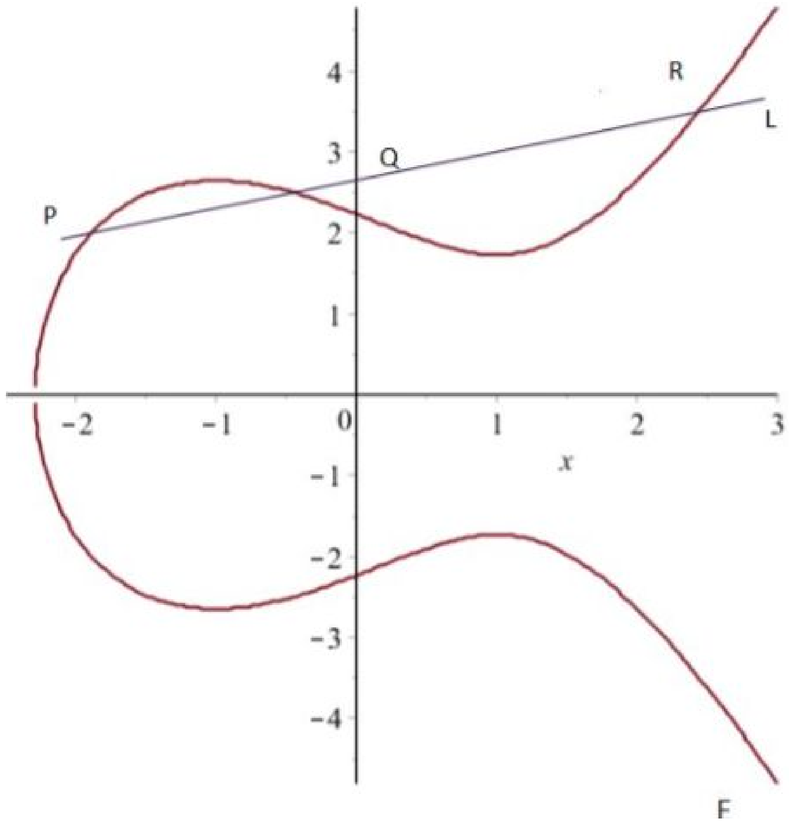
$$x^3 - (\alpha x + \beta)^2 + ax + b.$$

$$x_3 = \alpha^2 - x_1 - x_2.$$

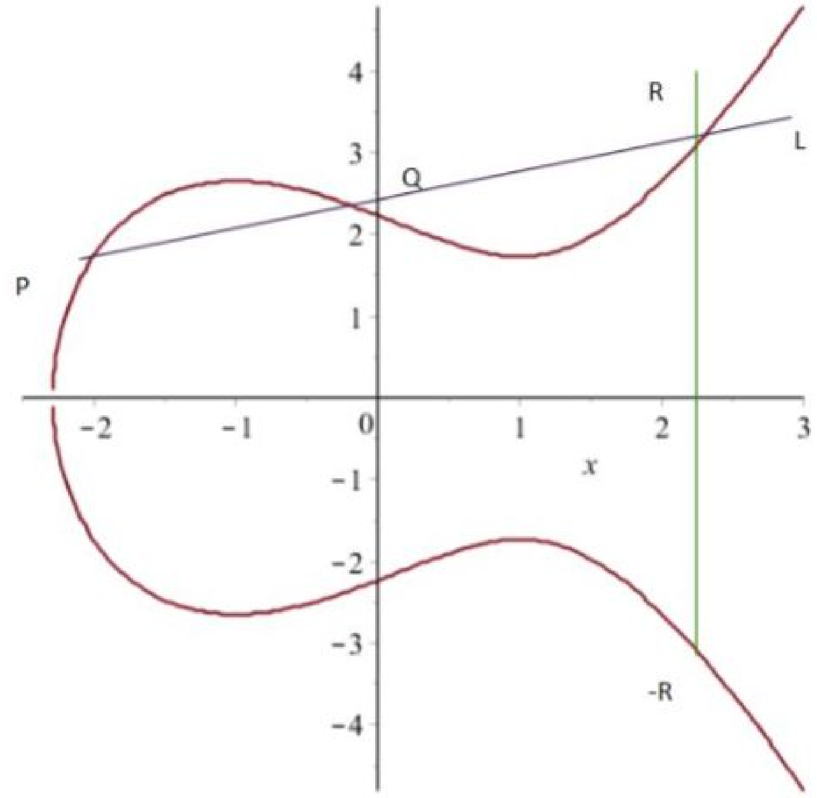
This leads to an expression for x_3, and hence $P + Q = (x_3, -(\alpha x_3 + \beta))$, in terms of x_1, x_2, y_1, y_2:

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

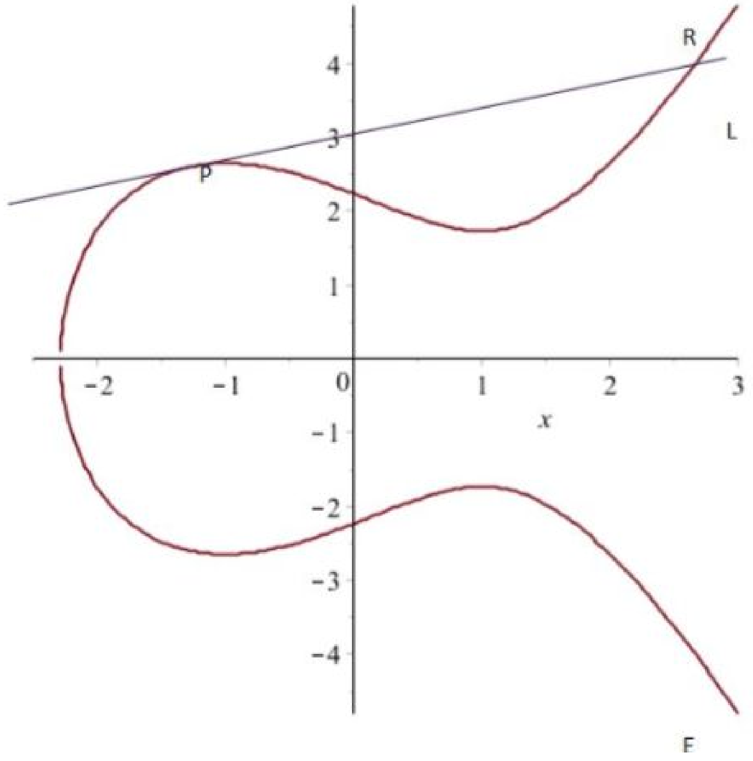
Adding P + Q = R (Chord-Tangent Process)



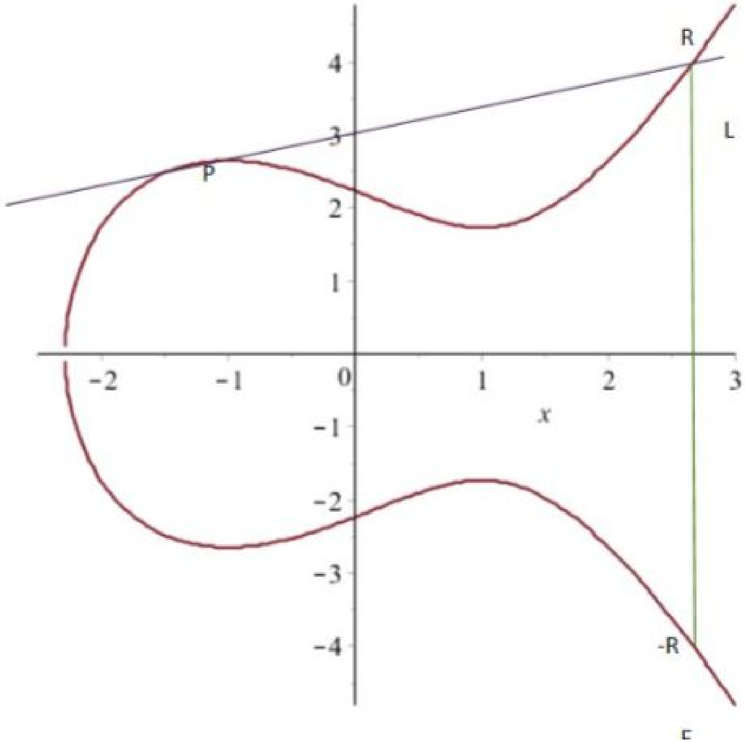
Adding P + Q = R (Chord-Tangent Process)



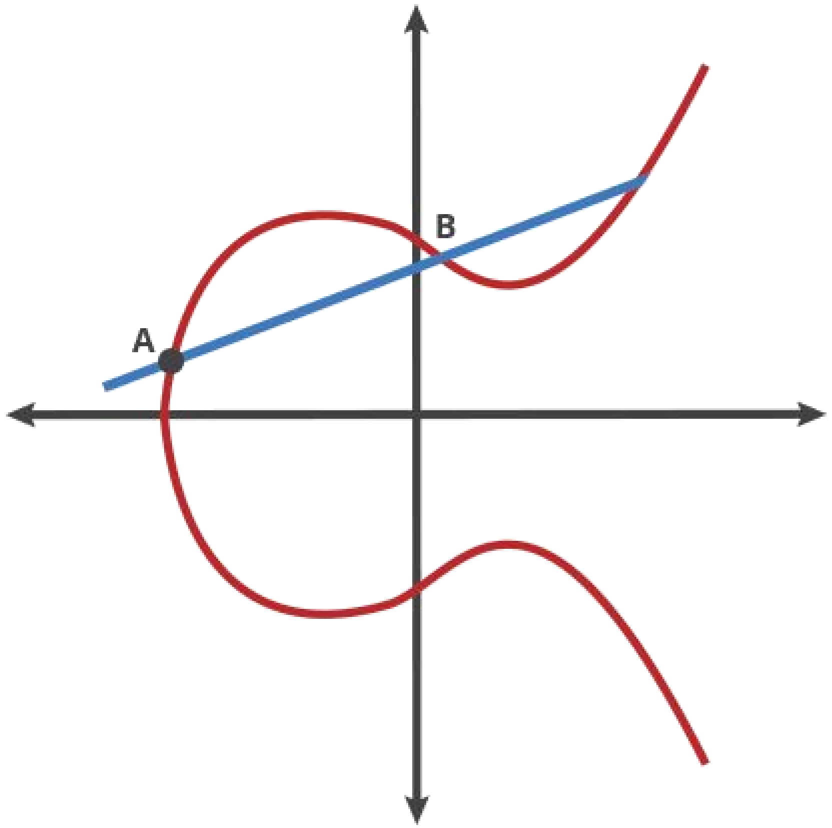
Point Doubling P + P = R (Chord-Tangent Process)



Point Doubling P + P = R (Chord-Tangent Process)

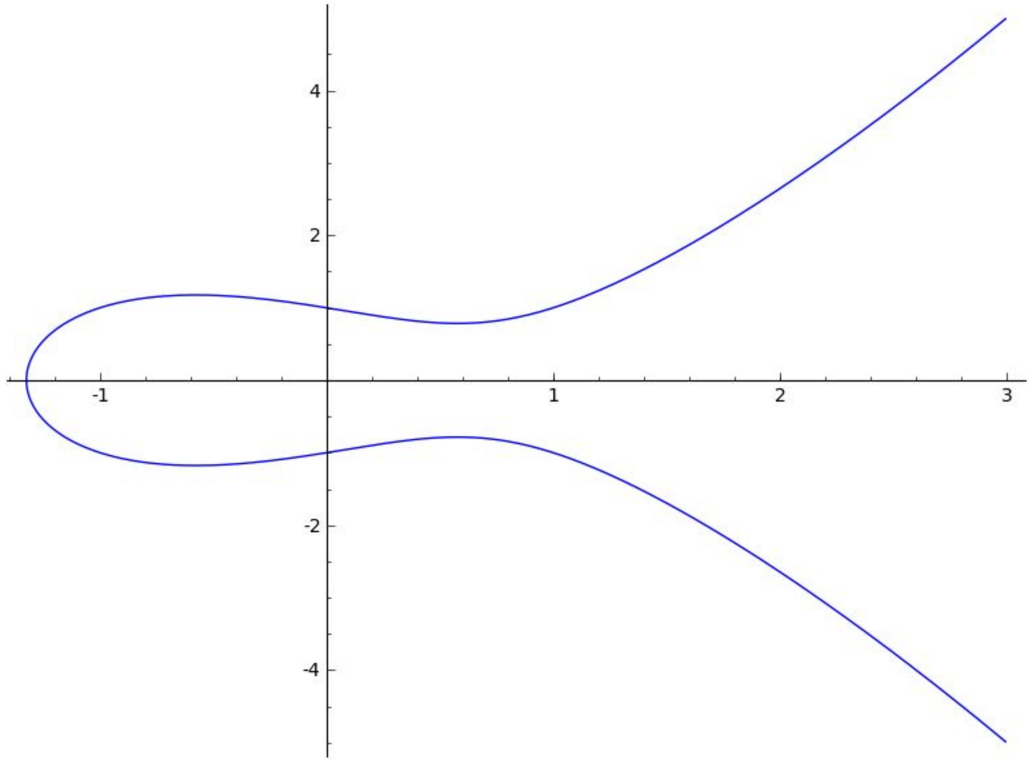


Elliptic GIF! (courtesy of Cloudflare)



Finite Fields (Making it Discrete)

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

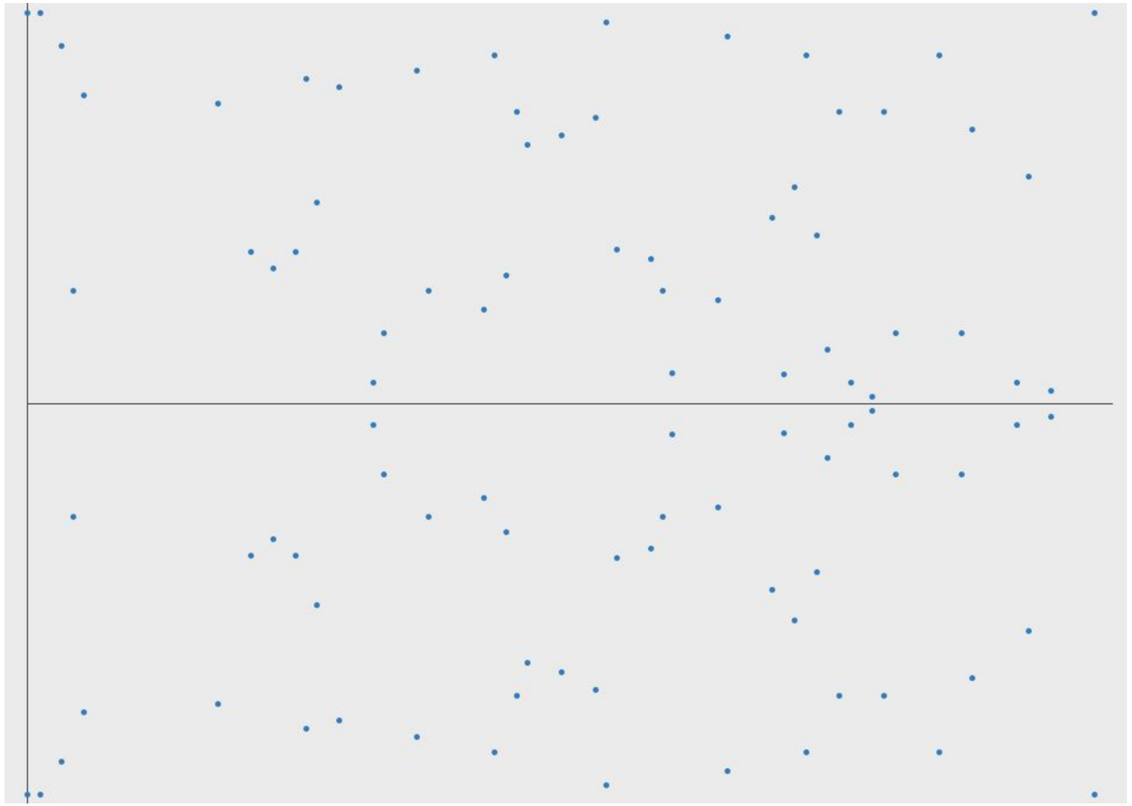


Finite Fields (Making it Discrete)

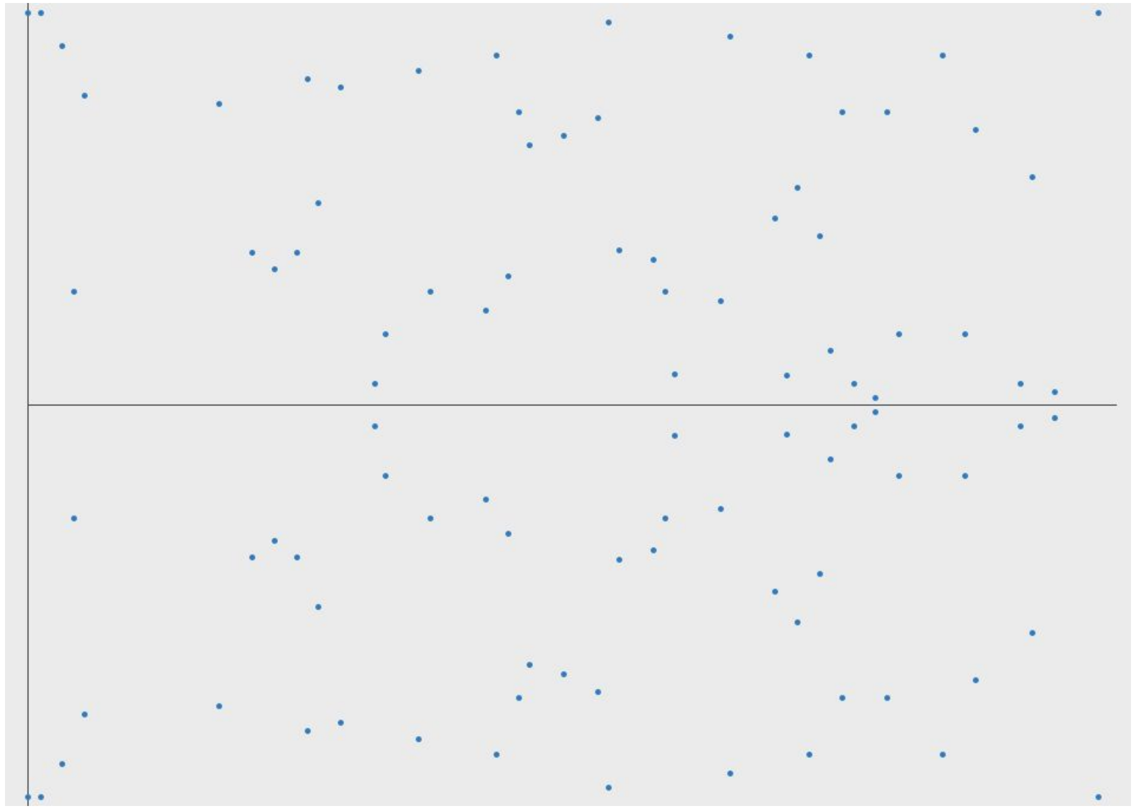
$$y^2 = x^3 + ax + b$$

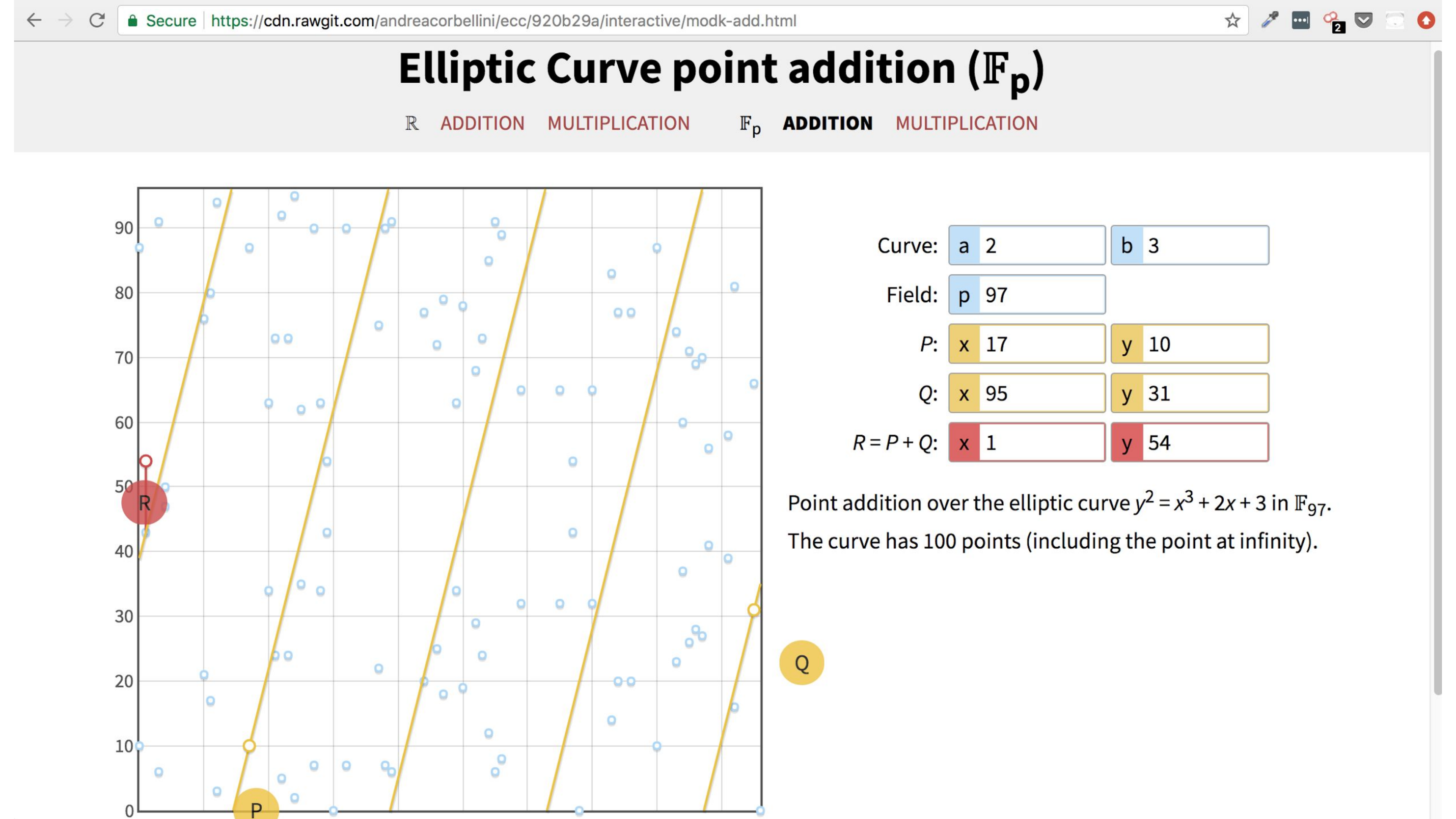
Where 0 <= x <= n -1

and x is an integer



Finite Field GIF!





https://cdn.rawgit.com/andreacorbellini/ecc/920b29a/interactive/modk-add.html

Elliptic Curve point addition ($\mathbb{F}_p$)

$\mathbb{R}$ ADDITION MULTIPLICATION $\mathbb{F}_p$ **ADDITION** MULTIPLICATION

Curve:	a 2	b 3
Field:	p 97	
P:	x 17	y 10
Q:	x 95	y 31
R = *P* + *Q*:	x 1	y 54

Point addition over the elliptic curve $y^2 = x^3 + 2x + 3$ in $\mathbb{F}_{97}$.

The curve has 100 points (including the point at infinity).

Finite Fields

- Finite set of integer elements {0, 1, 2, 3, 4 5, 6}

- Some concept of addition and multiplication (point adding/multiplying we saw earlier)

- We’re interested in fields modulo some prime number **p**
  - x mod p (clock math)

Modular Arithmetic (remainder math)

3 (mod 12) = 3

5 (mod 12) = 5

11 (mod 12) = 11

12 (mod 12) = 12

13 (mod 12) = 1

24 (mod 12) = 12

25 (mod 12) = 1

Domain Parameters of Finite Fields modulo a prime

- Calculate N, the order of the field (the number of points in the field)
  - Schoof's algorithm (runs in polynomial time, rather than exponential time!)

- Find a cyclic subgroup of F_p with large prime order n
  - The larger the order, the greater the security of our ECC algorithm
  - When it's prime, every point is guaranteed to have a multiplicative inverse, which means we can do ECDSA!
  - Due to Hasse's theorem, n is guaranteed to be a divisor of N

- Calculate the base point G
  - Using multiplication of G, we can generate all n elements in the cyclic subgroup of order n

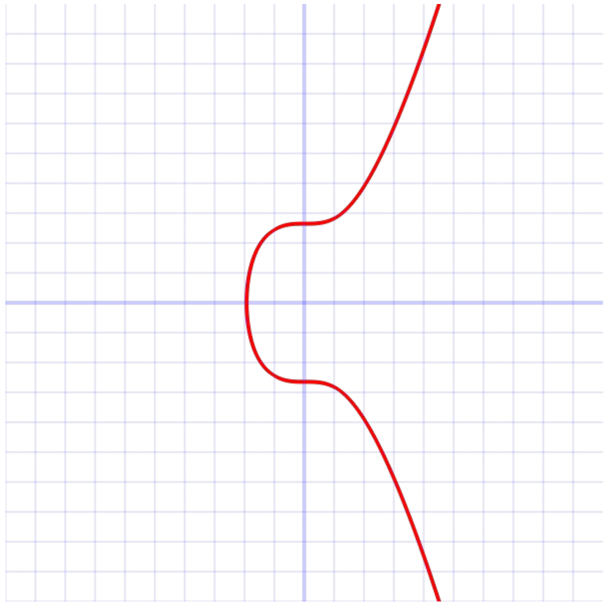
6 Domain Parameters for a Finite Field Elliptic Curve

(p, a, b, n, G, h)

- p: the prime modulus
- a: the a in y^3 + x^3 + ax + b
- b: the b in y^3 + x^3 + ax + b
- n: the order of the cyclic subgroup
- G: the base point of the prime order cyclic subgroup
- h: the cofactor equal to N / h

Bitcoin’s 6 Domain Parameters

secp256k1



- p: $2^{256} - 2^{32} - 2^{9} - 2^{8} - 2^{7} - 2^{6} - 2^{4} - 1$
- a: 0
- b: 7
- n: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141
- G_x: 0x79be667e f9dcbbac 55a06295 ce870b07 029bfcdb 2dce28d9 59f2815b 16f81798
- G_y: 0x483ada77 26a3c465 5da4fbfc 0e1108a8 fd17b448 a6855419 9c47d08f fb10d4b8
- h: 1

Break...

1. Crypto Basics
    a. Public/Private Key Digital Signatures
    b. “Hard” problems and “Easy” problems
    c. Discrete Logarithm Problem
2. Elliptic Curves
    a. EC Points
    b. Point Addition
    c. Finite Fields

ECDSA

- Given a public key N and a private key d, we can create digital signatures using elliptic curves on a finite field

- Uses small (relative to RSA) key sizes for the same level of security, 256 bit instead of 2048 bit
    - TLS handshake:
        - 256 ECC: 9516.8 sign/sec
        - 2048 RSA: 1001.8 sign/sec
    - This makes it faster to create signatures

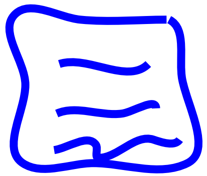
ECDSA

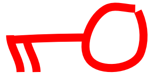
- Generate private key **priv**
  - Random number d which is between [1 … n] (remember n is the order of the cyclic subgroup)
- Generate public key **pub**
  - Calculate **pub = priv * G** using double and add algorithm

🔑 = pub

🔑 = priv

ECDSA Signature Creation

 = m,

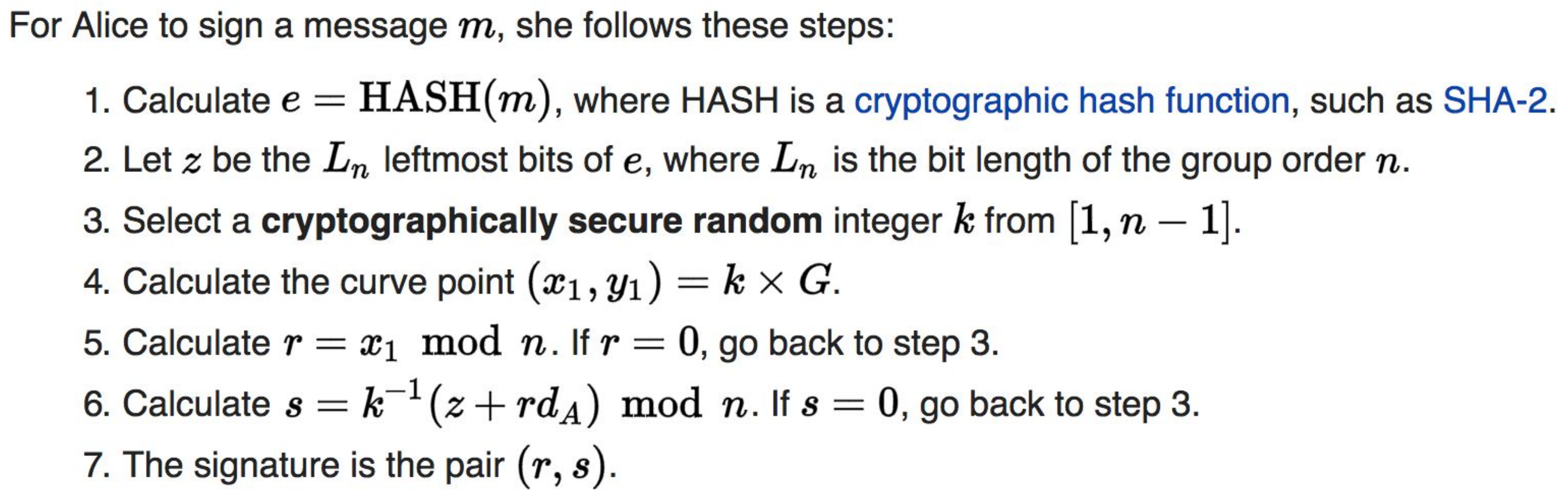
 = priv, $\Rightarrow$

```
-----BEGIN PGP
SIGNATURE-----
Version: GnuPG v1

iJwEAQEKAAYFAlRGkH
…..gHFLn+Lw1x6LUroOj
kl2zjpoCB
6pmQPd09MglBXJfnrBI=
=ET9V
-----END PGP
SIGNATURE-----
```

= (r, s)

random number = k

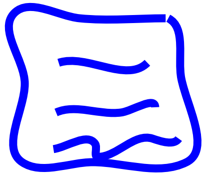
ECDSA Signature Creation



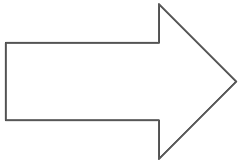
For Alice to sign a message m, she follows these steps:

1. Calculate $e = \mathrm{HASH}(m)$, where HASH is a cryptographic hash function, such as SHA-2.
2. Let z be the L_n leftmost bits of e, where L_n is the bit length of the group order n.
3. Select a **cryptographically secure random** integer k from $[1, n-1]$.
4. Calculate the curve point $(x_1, y_1) = k \times G$.
5. Calculate $r = x_1 \mod n$. If $r = 0$, go back to step 3.
6. Calculate $s = k^{-1}(z + rd_A) \mod n$. If $s = 0$, go back to step 3.
7. The signature is the pair (r, s).

ECDSA Signature Verification

 = m

= pub



True, False

-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1

iJwEAQEKAAYFAlRGkH
…..gHFLn+Lw1x6LUroOj
kl2zjpoCB
6pmQPd09MglBXJfnrBI=
=ET9V
-----END PGP SIGNATURE-----

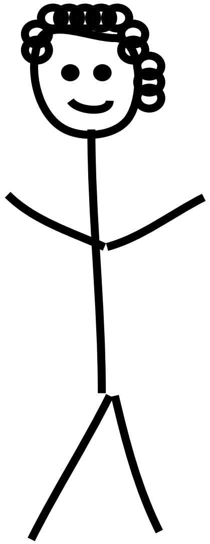
= (r, s)

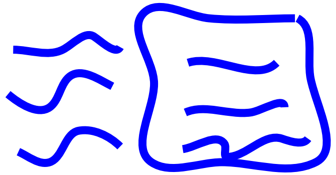
ECDSA Signature Verification

1. Verify that r and s are integers in $[1, n-1]$. If not, the signature is invalid.
2. Calculate $e = \mathrm{HASH}(m)$, where HASH is the same function used in the signature generation.
3. Let z be the L_n leftmost bits of e.
4. Calculate $w = s^{-1} \bmod n$.
5. Calculate $u_1 = zw \bmod n$ and $u_2 = rw \bmod n$.
6. Calculate the curve point $(x_1, y_1) = u_1 \times G + u_2 \times Q_A$. If $(x_1, y_1) = O$ then the signature is invalid.
7. The signature is valid if $r \equiv x_1 \pmod{n}$, invalid otherwise.

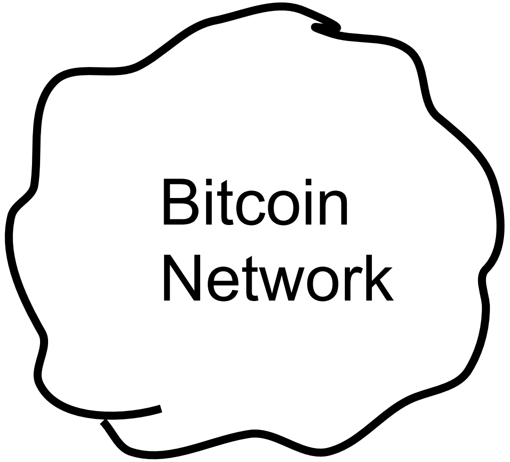
ECDSA and Bitcoin

ECDSA and Bitcoin

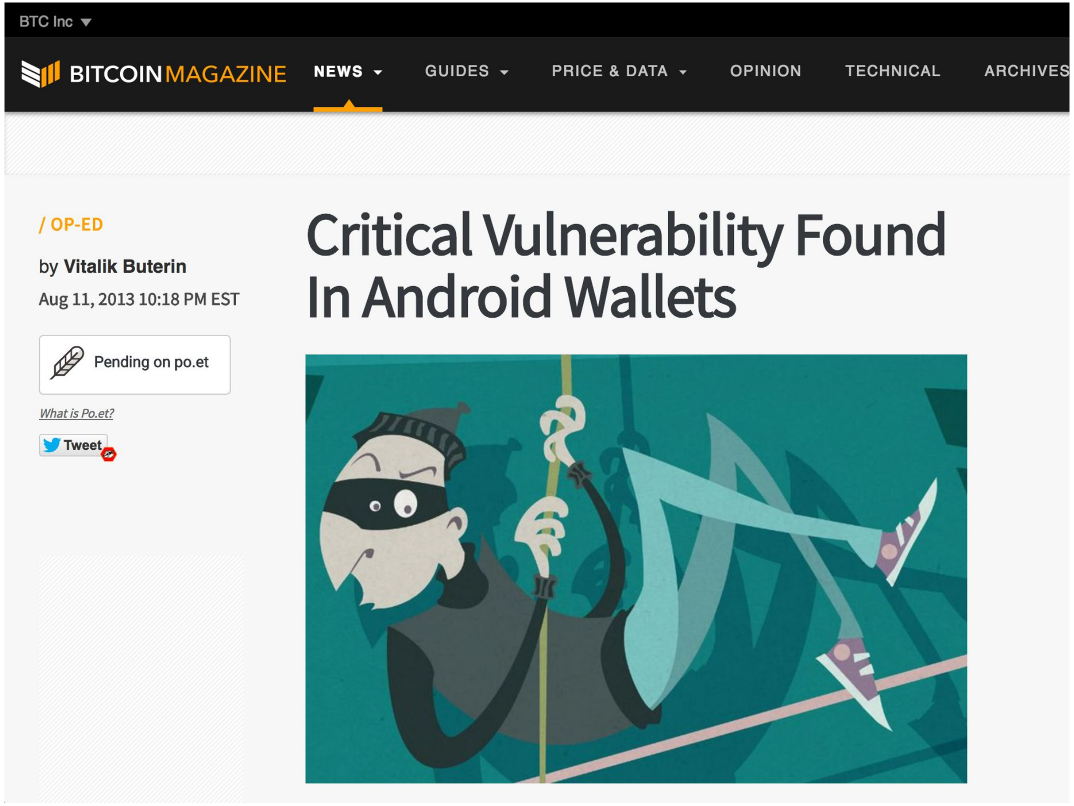




I sent 5 BTC to address:
1Nf69xJt68KCAaoRHfyxsDTsQ99GuTm5s9



ECDSA and non-random numbers


BTC Inc

BITCOIN MAGAZINE NEWS GUIDES PRICE & DATA OPINION TECHNICAL ARCHIVES

/ OP-ED

by Vitalik Buterin

Aug 11, 2013 10:18 PM EST

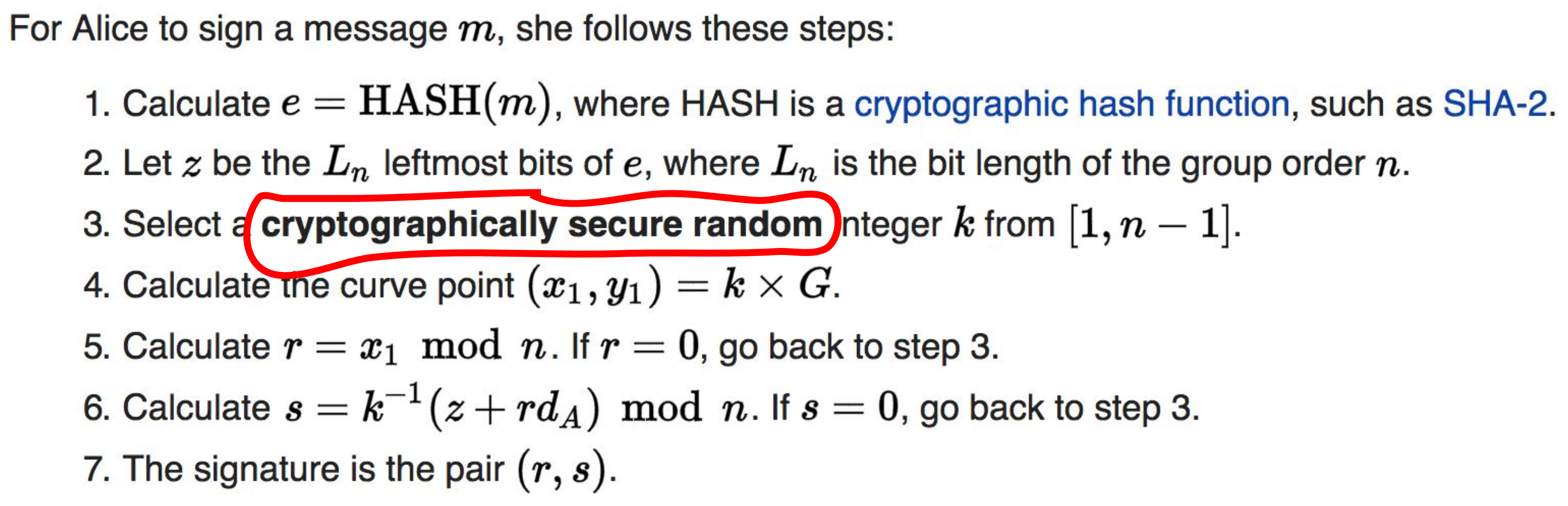
Pending on po.et

What is Po.et?

Tweet

Critical Vulnerability Found In Android Wallets

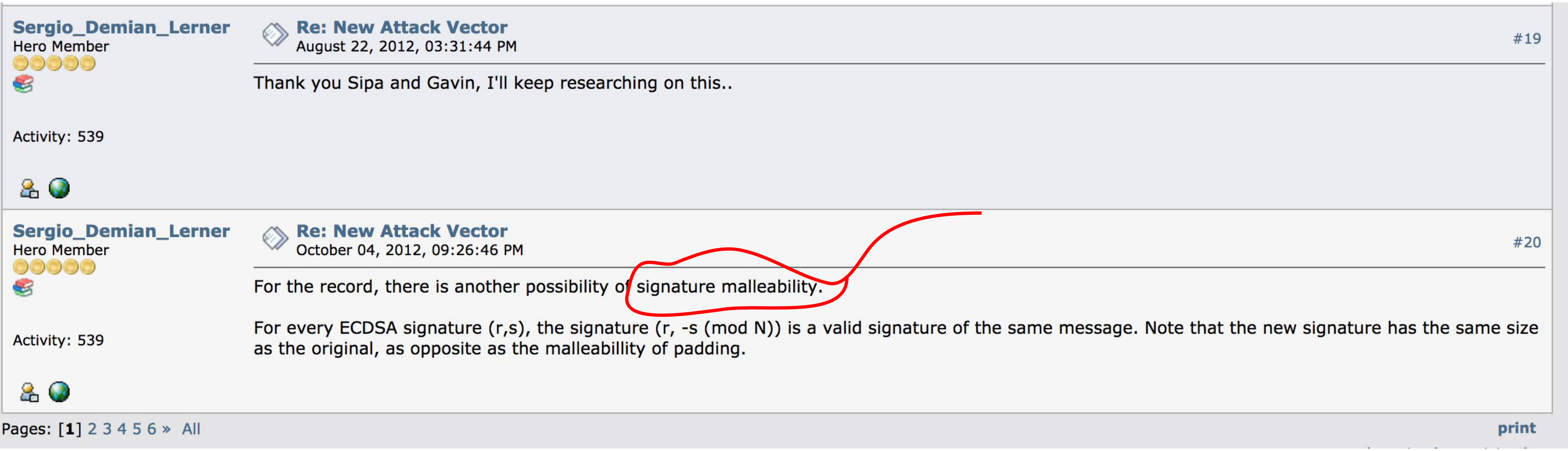
ECDSA and non-random numbers



For Alice to sign a message m, she follows these steps:

1. Calculate $e = \mathrm{HASH}(m)$, where HASH is a cryptographic hash function, such as SHA-2.
2. Let z be the L_n leftmost bits of e, where L_n is the bit length of the group order n.
3. Select a **cryptographically secure random** integer k from $[1, n-1]$.
4. Calculate the curve point $(x_1, y_1) = k \times G$.
5. Calculate $r = x_1 \mod n$. If $r = 0$, go back to step 3.
6. Calculate $s = k^{-1}(z + rd_A) \mod n$. If $s = 0$, go back to step 3.
7. The signature is the pair (r, s).

ECDSA and Transaction Malleability



Sergio_Demian_Lerner
Hero Member

Activity: 539

Re: New Attack Vector
August 22, 2012, 03:31:44 PM

#19

Thank you Sipa and Gavin, I'll keep researching on this..

Sergio_Demian_Lerner
Hero Member

Activity: 539

Re: New Attack Vector
October 04, 2012, 09:26:46 PM

#20

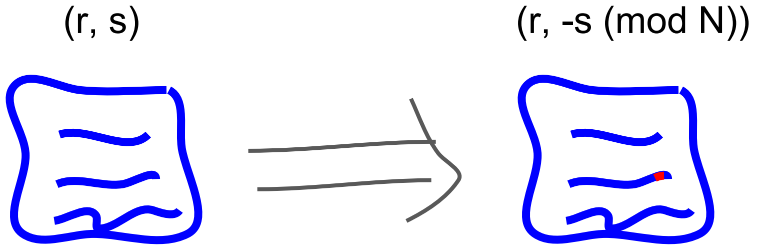
For the record, there is another possibility of signature malleability.

For every ECDSA signature (r,s), the signature (r, -s (mod N)) is a valid signature of the same message. Note that the new signature has the same size as the original, as opposite as the malleabillity of padding.

Pages: [**1**] 2 3 4 5 6 » All

print

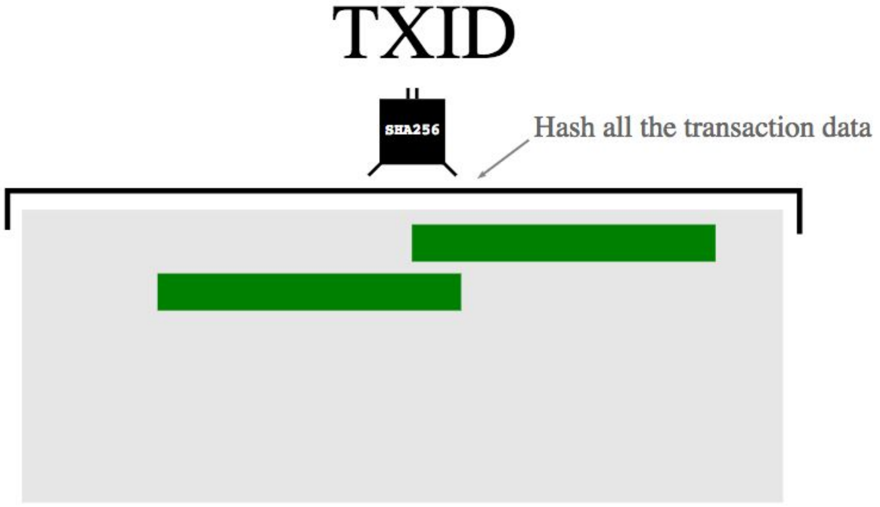
ECDSA and Transaction Malleability

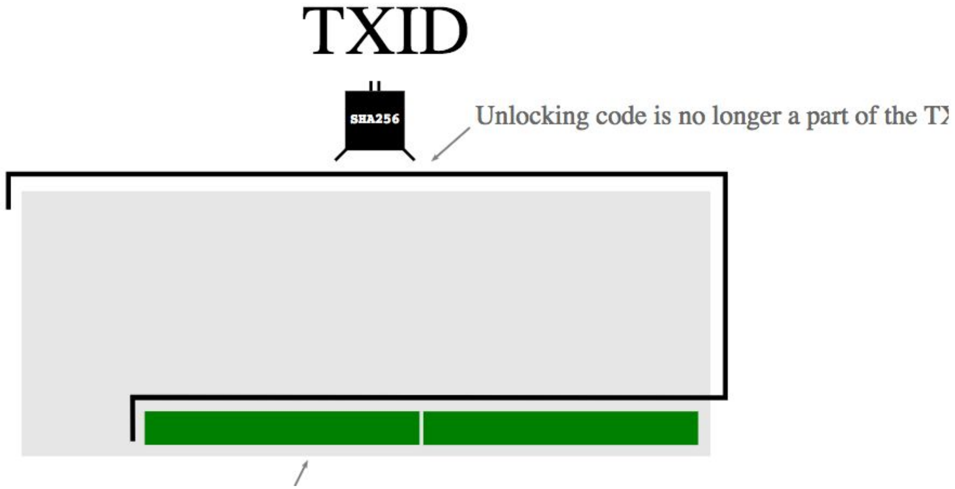


TXID:
137f8145c9126fb3da1e5
58b4dac89936a719d8e7
5582786f533ce17330f52

TXID:
fd41cfb4d290a0d7e88c6
ddfb0eebef2115499d219
7f8469238c7eae3fa1930

ECDSA and Transaction Malleability





Future Study

- Methods for solving the Discrete Logarithm on Elliptic Curves
  - Pohlig-Hellman
  - Baby-step/Giant-step
  - Pollard’s Rho Algorithm
- Schnorr Signatures
  - aggregatable signatures (yay multisig!)
- Curves besides secp256k1
  - Twisted Edwards curves
- Quantum Computer and ECC
  - Shor’s Algorithm: polynomial time solution to DLP :0
- Read npm’s **elliptic** source code

References:

- https://en.wikipedia.org/wiki/Elliptic-curve_cryptography
- https://en.wikipedia.org/wiki/Finite_field
- https://bitcoin.stackexchange.com/questions/21907/what-does-the-curve-used-in-bitcoin-secp256k1-look-like
- https://crypto.stackexchange.com/questions/653/basic-explanation-of-elliptic-curve-cryptography#657
- https://cdn.rawgit.com/andreacorbellini/ecc/920b29a/interactive/modk-add.html
- http://andrea.corbellini.name/2015/05/23/elliptic-curve-cryptography-finite-fields-and-discrete-logarithms/
- https://ellipticnews.wordpress.com/2010/12/26/elliptic-curve-cryptography-books/
- https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/
- https://blog.cloudflare.com/ecdsa-the-digital-signature-algorithm-of-a-better-internet/
- http://www.cs.bris.ac.uk/~nigel/Crypto_Book/
- Applied Cryptography, Bruce Schneier
- https://en.bitcoin.it/wiki/Secp256k1
- https://math.berkeley.edu/~ribet/parc.pdf
- https://www.math.brown.edu/~jhs/Presentations/WyomingEllipticCurve.pdf
- Below is a very informative article!
- https://github.com/bellaj/Blockchain/blob/6bffb47afae6a2a70903a26d215484cf8ff03859/ecdsa_bitcoin.pdf
- https://github.com/bitcoin/bitcoin/pull/6769

Double and add algorithm

- Makes generating public keys go from “Hard” to “Easy”!

- O(n) -> O(log(n))

- Example
  - 16P = P * P * P * … * P = 2P * 2P * 2P * 2P * 2P * 2P * 2P * 2P = 4P * 4P * 4P *4P = 8P * 8P
  - 151P = P^7 * P^4 * P^2 * P^1 * P^0