SIGPwny

SP2025 Week 05 • 2025-02-27

Crypto IV - Elliptic Curve Cryptography

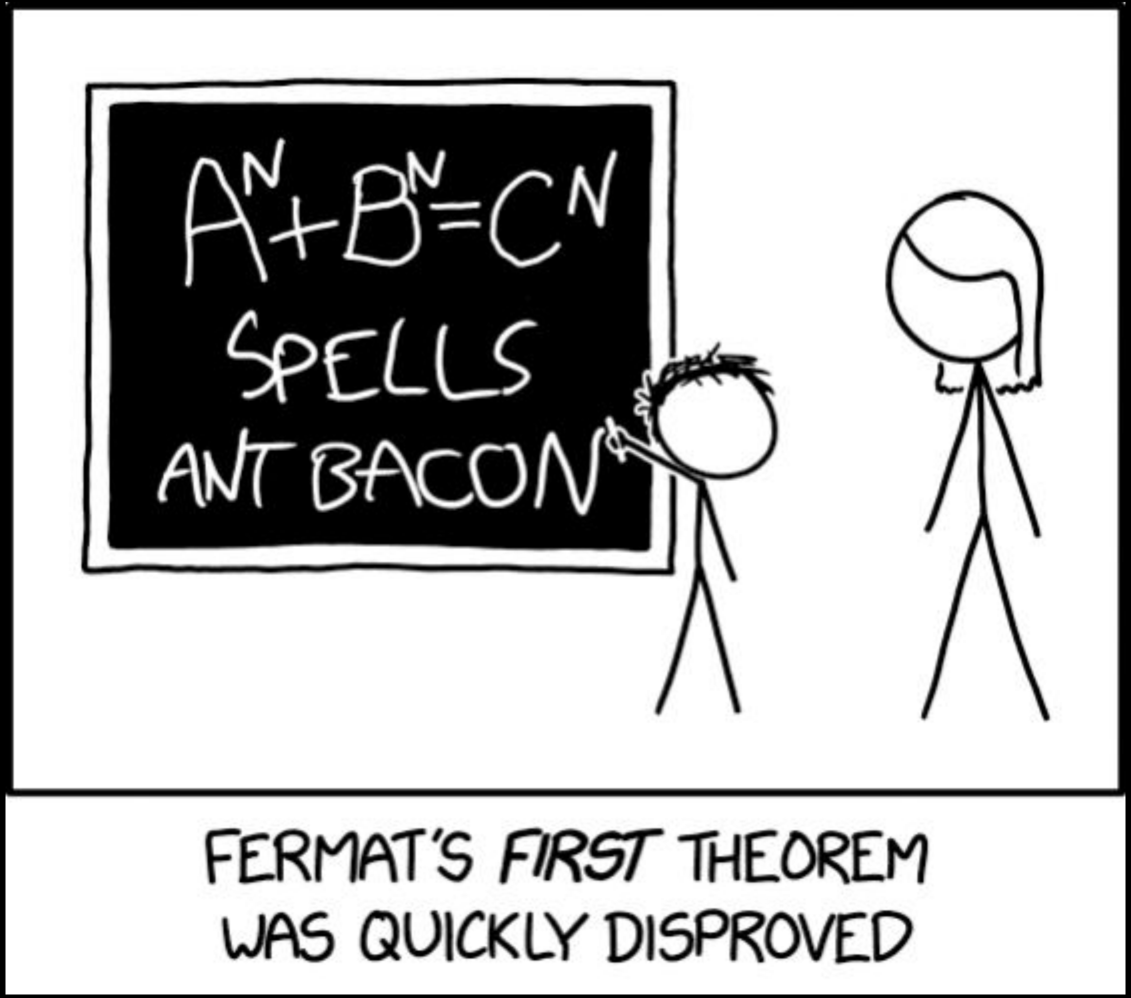
Sagnik and George

Announcements

- Security careers this Sunday (3/2)
  - SIGPwny alums will be there!

ctf.sigpwny.com

sigpwny{d3f1n1t3ly_n0t_b4ckd00r3d}



What are elliptic curves?

Background: Groups and Fields

Group: A set and some binary operation on that set (e.g. $(\mathbb{Z}, +)$)

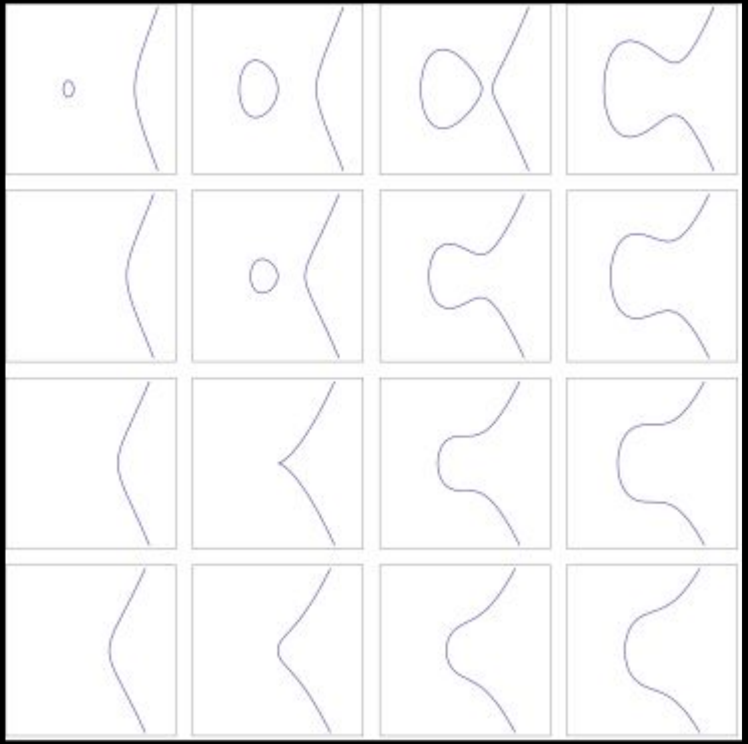
Group must be:

- Closed $a + b \in \mathbb{Z}$
- Have a (unique) identity $e \mid \forall a \in \mathbb{Z}, a + e = a$
- Each element must have an inverse $\forall\ a\ \exists\ a^{-1} \mid a + a^{-1} = e$
- The operation must be associative $a + (b + c) \equiv (a + b) + c$

Field: Add another operation (multiplication) to a commutative (abelian) group that satisfies the same properties*

What are elliptic curves?

(Basically), studies of cubic functions that look like $Y^2 = X^3 + aX + b$ (we care about those in the *Weierstrass* form)



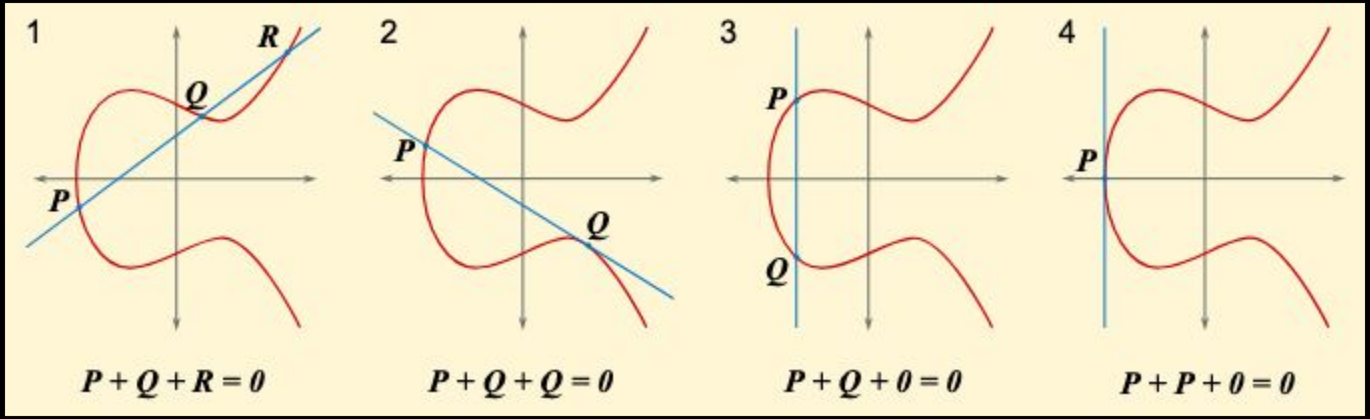
Desmos example

Point Addition

To add two points P and Q, draw a line through them (tangent if $P = Q$). Find the third point the line intersects, and reflect it over the x-axis

Problem: not all lines intersect 3 points on the curve

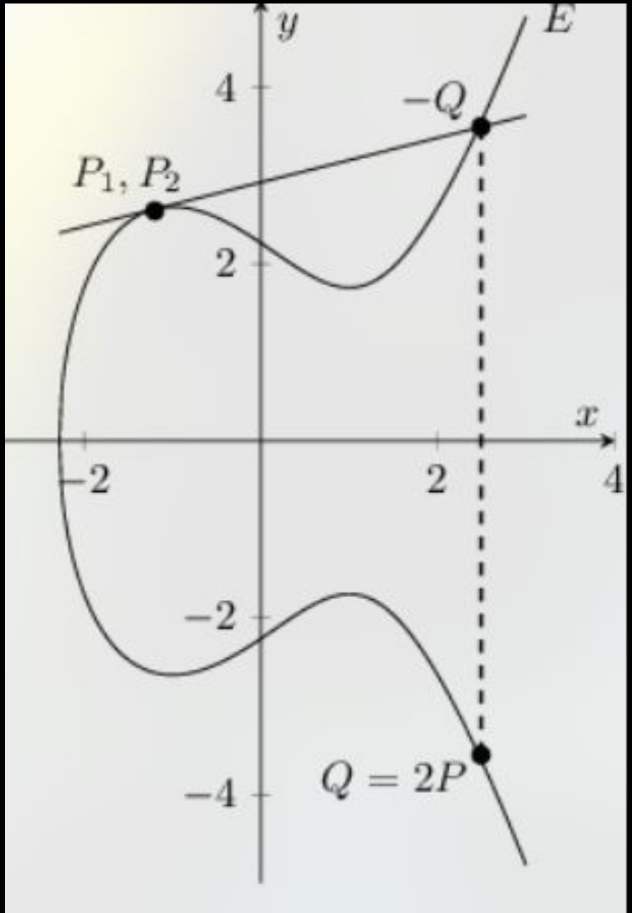
Let's define all such lines to intersect with a point at infinity, O.



Scalar Multiplication

Repeated addition = multiplication

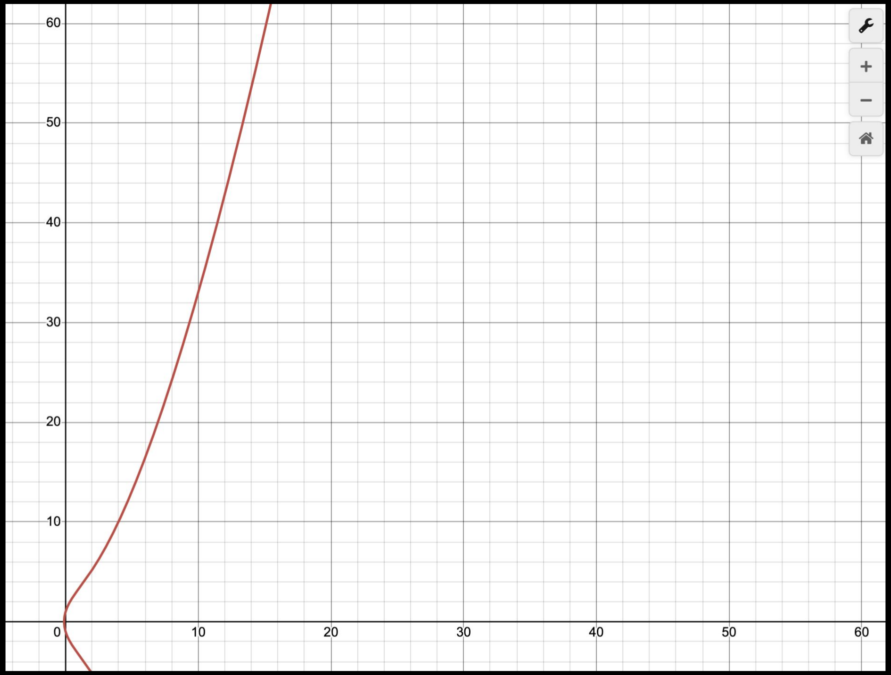
If we then repeatedly add a point to itself, we can calculate any integer scalar-multiplied point `Q = [x]P` on the curve!



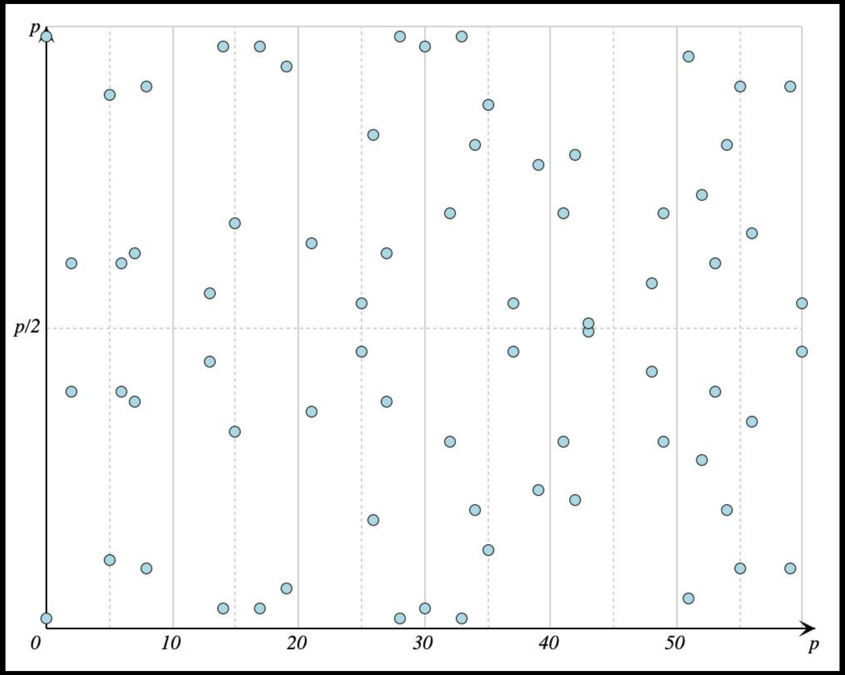
(b) Point doubling

Making it Discrete

Instead of considering the set of solutions to the curve $(x, y) \in \mathbb{R}^2$, only consider points in a **finite field** (and the point at infinity O)



$$Y^2 = X^3 + 9X + 1$$



Same curve in $\mathbb{F}_{61}$

Show me the crypto

Recall: RSA and Diffie–Hellman

Alice and Bob want to establish a *shared secret*

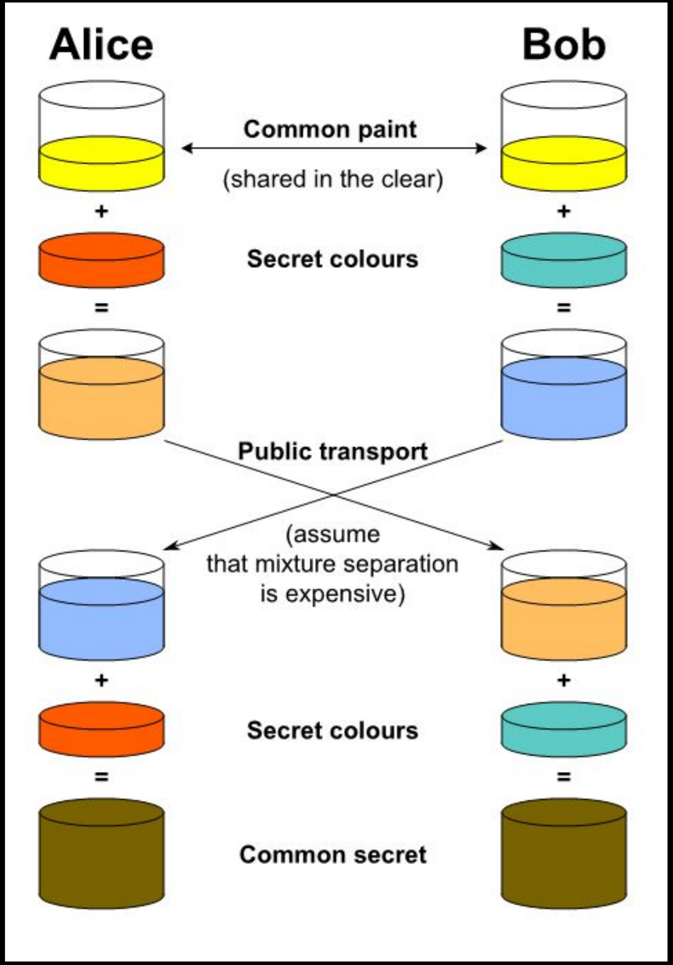
Use hard-to-invert (trapdoor) functions acting on groups

For DH key exchange,

$$(g^{a} \pmod{p})^{b} \pmod{p} = (g^{b} \pmod{p})^{a} \pmod{p}$$

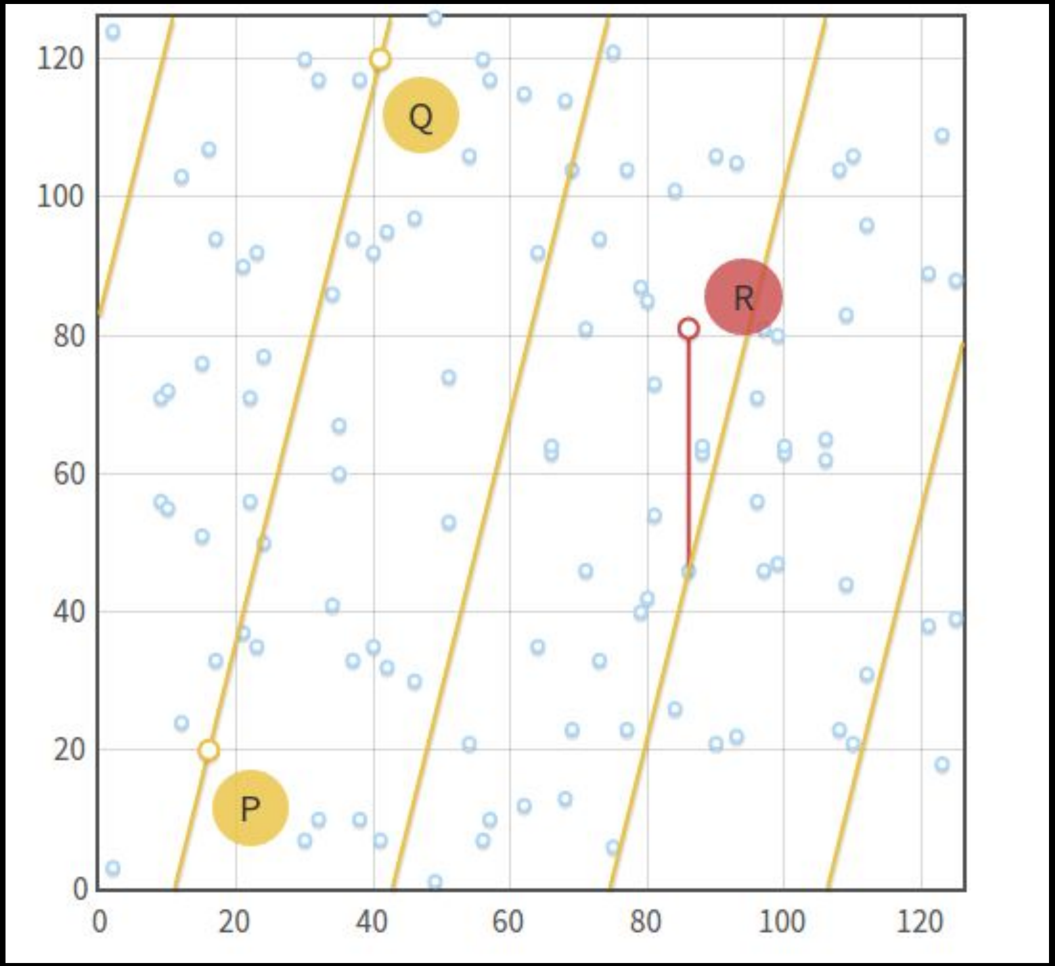
Old Diffie Hellman (Boring!)

- Alice, Bob share g and large prime p
- Alice has secret a, Bob has b
- Alice sends $A = g^a \bmod p$, Bob sends $B = g^b \bmod p$
- Both can compute $s = A^b \bmod p = B^a \bmod p$



New Diffie Hellman!

- Alice and Bob share G, large prime p (on a selected curve)
- Alice sends [a]G mod p
- Bob sends [b]G mod p
- Both can compute [ab]G mod p



The Discrete Log Problem (ECC)

Given some Elliptic curve E, some primitive element/point P on the curve, and a target point T on the curve, find the integer d such that:

$$\underbrace{P + P + P + \ldots + P}_{d \text{ times}} = dP = T$$

The Discrete Log Problem (ECC)

What ball was hit (and how was it hit) to get to this state?

The conditions for DLog

- Cryptosystems are based on the idea that *d* is large, kept secret, and attackers cannot compute it easily
- If d is known, an efficient method to compute *dP* is required to create a reasonable cryptosystem
- For ECs, we can use the known Double and Add algorithm

```
D&A(E, P, d, $d_i$):
    Init T to 0
    For i = t-1 (mod n)..0:
        T = T+T (mod n)
        if ($d_i$== 1):
            T = T+P (mod n)
    endfor
    Return T
```

Example of Double-and-Add:

Suppose you want to find T = 13P:

- $13_{10} = 1101_{2}$
- First bit is 1 so first compute T + T (0). Then do T+P = P
- Second bit is 1 again so compute T = T+T = 2P, T + P = 3P
- Third bit is 0, so just compute the double T = 6P
- Last bit is 1, so T=T+T+P = 12P+P = 13P

Why we care: Performance/Size

```
ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIHy60AXoP5Qbn1oxSTB/Zy2Tl6NZDHjcv
36J70WsE4KY

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDDD8NiqekaqR23MGmy/zEXMlc4b
CTMmXrlUdrI2mJk5mNqrRHKzJjrJTG2uqWwJldWtATFo7cb3sVUHrDZ3P
6wMK66pFN7PKHnzDUzqDuX/OBqzqCPgRgo/X8gSHgwLEzhYlAa+coZGP9
X1nsRjBsCSRZ9gThf478vtufVKVCaZIWejdXp5s4Wd3bUbfK7xL3frctl
i/hXla2e62jQk6hwmVNtf09NlEiXXFSTtt5KDJWKQNRyA+WsYXvQqzNuz
PjTsZY1JlTJLfzQ6QtLTT0dIoPVVPrS+/UvKoG2TOmVEq8uNwfpdPu6Ah
jlOlg1iJ0U3h/iQdaiAinzWdD9Rc0LGSJdq5ki6KPQ7iBjHq/X1IjIDdn
e9PT/HNUDc34m+hnjqeopkhKdfz/G+MutlZkTntqDdZDuILTvrUu2BMI5
XXbTFQeCr4VO6b9auGuZdpRlaUfJIDX35Z3p6ry6RWfv/vWv620pJP9Le
d/9F6/rrw47RFT8qnDKbkcgXqbU0c0=
```

TLS 1.3

ed25519: We choose a field $\mathbb{F}_{2^{255}-19}$ and a curve $y^2 = x^3 + 486662x^2 + x$

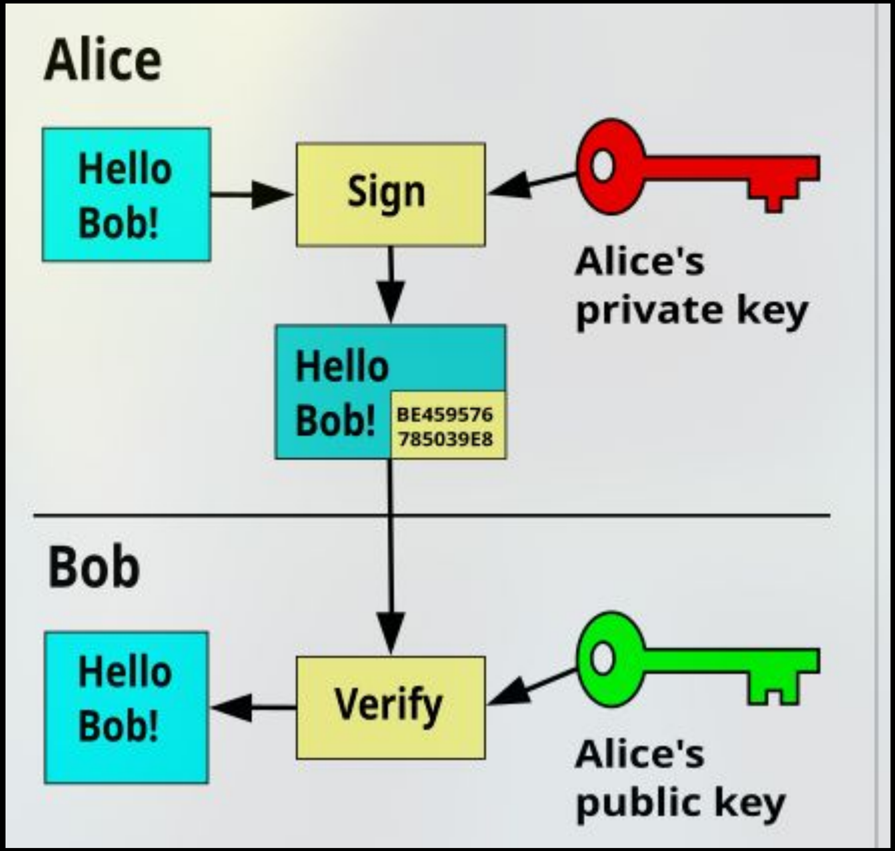
(Note: the curve was chosen by searching the space of “all curves with the following properties that make them fast on hardware” and then somewhat arbitrarily picking 486662 as the coefficient)

Other common curves are secp256k1(Bitcoin) and NIST curves

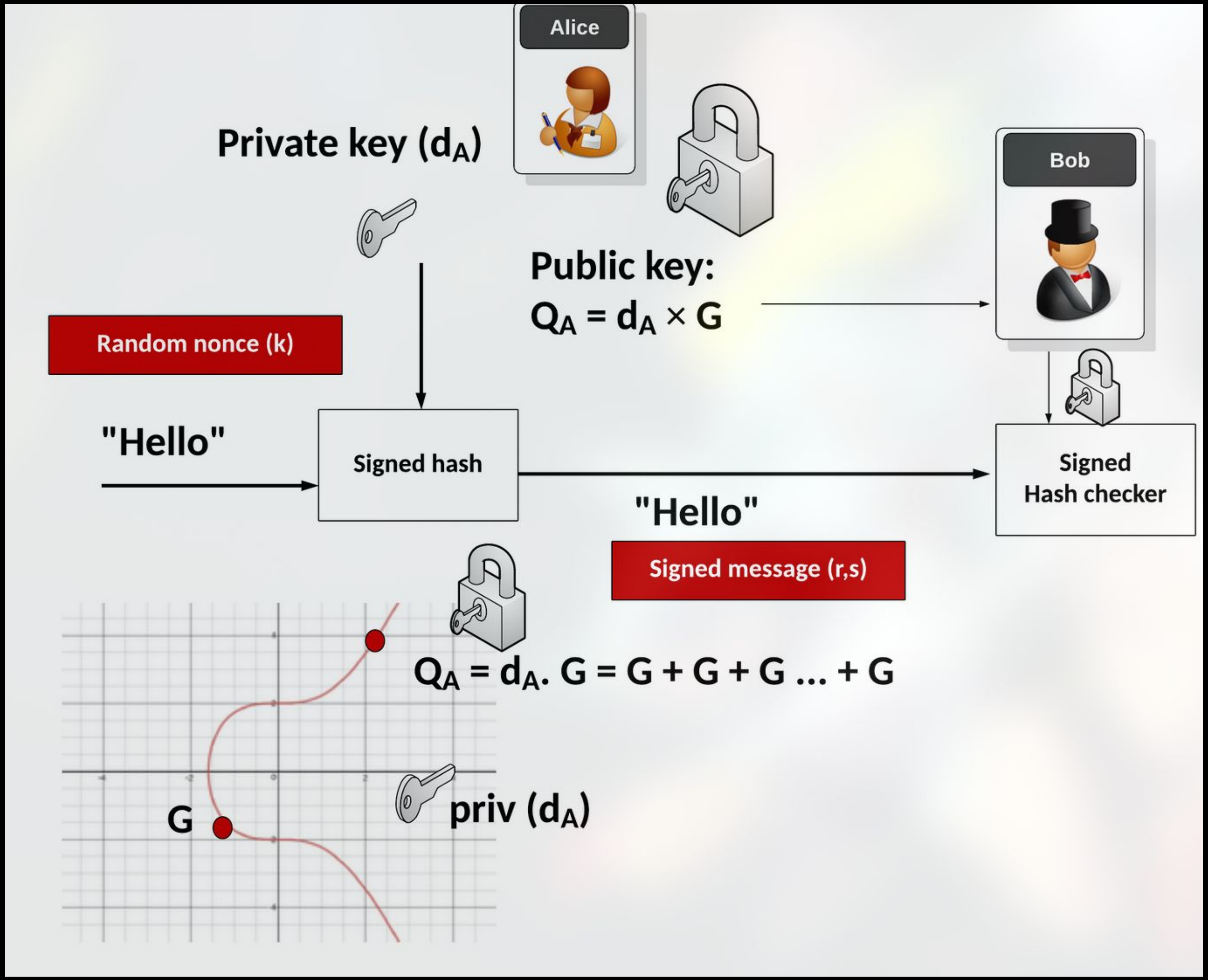
See also how to choose safe curves

Signatures

- Use the private key to sign a message
- Anyone who has the signer's pub key can
  - Decrypt the signature
  - Recompute the hash of the OG message
  - Check if the hash matches



ECDSA


Alice
Private key (d_A)
Public key:
$Q_A = d_A \times G$
Bob
Random nonce (k)
"Hello"
Signed hash
Signed Hash checker
"Hello"
Signed message (r,s)
$Q_A = d_A . G = G + G + G \ldots + G$
priv (d_A)
G

ECDSA

Alice signs:

- Create hash $e = H(m)$
- Let h be the L_n leftmost bits of e, L_n has group order N
- Create a random number k in $[1,...,N-1]$
- Calculate a point $(x_1, y_1) = kG$
- Calculate $r = x_1 \mod N$, if $r = 0$, go back to 3
- Find $s = k^{-1}(h+rd_A) \mod N$. If $s = 0$, go back to 3.
- Return (r,s)

ECDSA

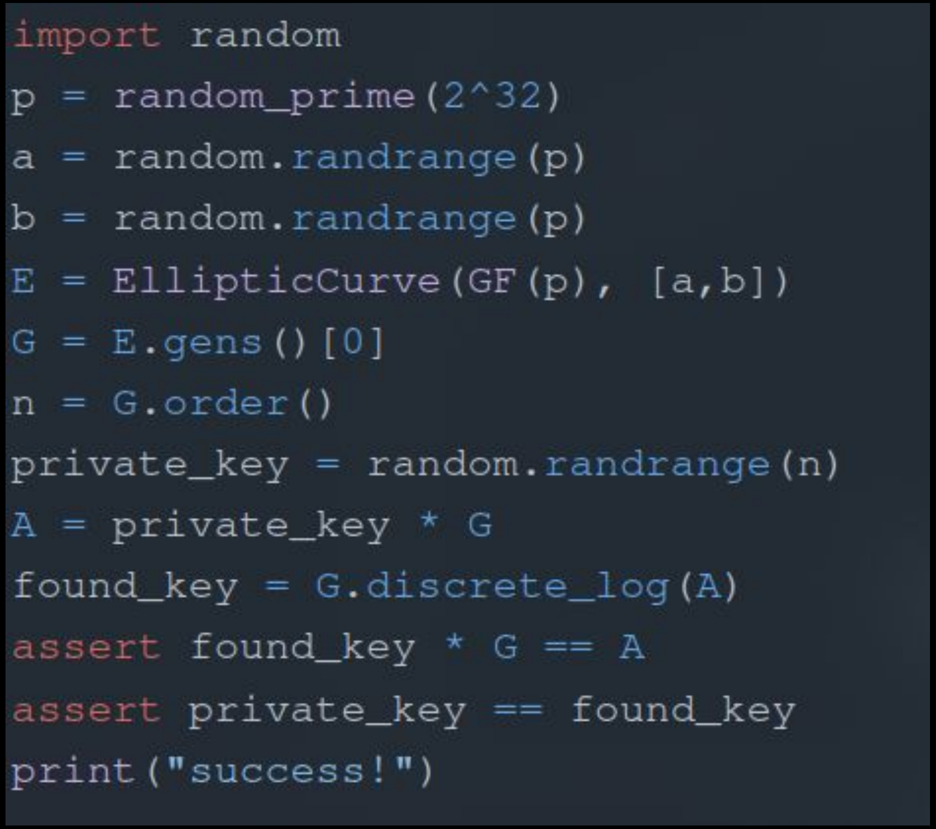
Bob checks:

- Compute the hash of the sent message $e = H(m)$,
- Let h be the L_n leftmost bits of e
- Calculate $c = s^{-1} \pmod N$
- Calculate $u_1 = hc \pmod N$, $u_2 = rc \pmod N$
- Calculate the point on the curve $(x_1, y_1) = u_1 \times G + u_2 \times Q_A$. Ensure (x_1, y_1) is nonzero
- if $r = x_1 \bmod N$, the signature passes

Attacks

If the generator's order is small

- Sometimes ECDH implementations might use a small n
- It becomes possible to solve the ECDLP in O(√n)
- These functions can be called in with our favorite tool SageMath:



```
import random
p = random_prime(2^32)
a = random.randrange(p)
b = random.randrange(p)
E = EllipticCurve(GF(p), [a,b])
G = E.gens()[0]
n = G.order()
private_key = random.randrange(n)
A = private_key * G
found_key = G.discrete_log(A)
assert found_key * G == A
assert private_key == found_key
print("success!")
```

Resources

- CryptoHack
- Ben Lynn’s notes
- Implementing Curve25519
- Visualizing Elliptic Curve Cryptography

Next Meetings

`2025-03-02` • **This Sunday**

- Careers in cybersecurity; learn what SIGPwny alumni are up to!

`2024-03-06` • **Next Thursday**

- Password Cracking with Adarsh!

ctf.sigpwny.com

sigpwny{d3f1n1t3ly_n0t_b4ckd00r3d}

Meeting content can be found at sigpwny.com/meetings.

SIGPwny