Crypto II

Kuruwa

ToC

Symmetric Cryptography Revisited

$$\begin{tikzcd}
 & K \arrow[d] & & K \arrow[d] & \\
x \arrow[r] & \boxed{e_K(x)} \arrow[rr, "y"] & & \boxed{d_K(x)} \arrow[r] & x
\end{tikzcd}$$

- The same secret key K is used for encryption and decryption
- Encryption and Decryption are very similar (or even identical) functions

Symmetric Cryptography: Shortcomings

- Key distribution problem: The secret key must be **transported securely**
- Number of keys: n users in the network require n(n-1)/2 keys, each user stores (n-1) keys
- Alice or Bob can **cheat each other,** because they have identical keys

Asymmetric (Public-Key) Cryptography

$$\begin{tikzcd}
 & (K_{pub},K_{pr})=K \arrow[l, "K_{pub}"'] \\
x & \\
y=e_{K_{pub}}(x) \arrow[r, "y"] & x=d_{K_{pr}}(y)
\end{tikzcd}$$

- During the key generation, a key pair K_{pub} and K_{pr} is computed
- Alice encrypts a message with the **not secret** public key K_{pub}
- Only Bob has the **secret** private key K_{pr} to decrypt the message

Basic Key Transport Protocol

Hybrid systems: incorporating asymmetric and symmetric algorithms

- **Key exchange** (for symmetric schemes) are performed with (slow) **asymmetric** algorithms
- **Encryption** of data is done using (fast) symmetric ciphers, e.g., **block ciphers or stream ciphers**

Basic Key Transport Protocol

Key Exchanbe (Asymmetric)

$$\xleftarrow{\quad K_{pub} \quad} \qquad (K_{pub}, K_{pr}) = K_B$$

choose a random symmetric key K

$$y_1 = e_{K_{pub}}(K) \xrightarrow{\quad y_1 \quad} K = d_{K_{pr}}(y_1)$$

Encryption (Symmetric)

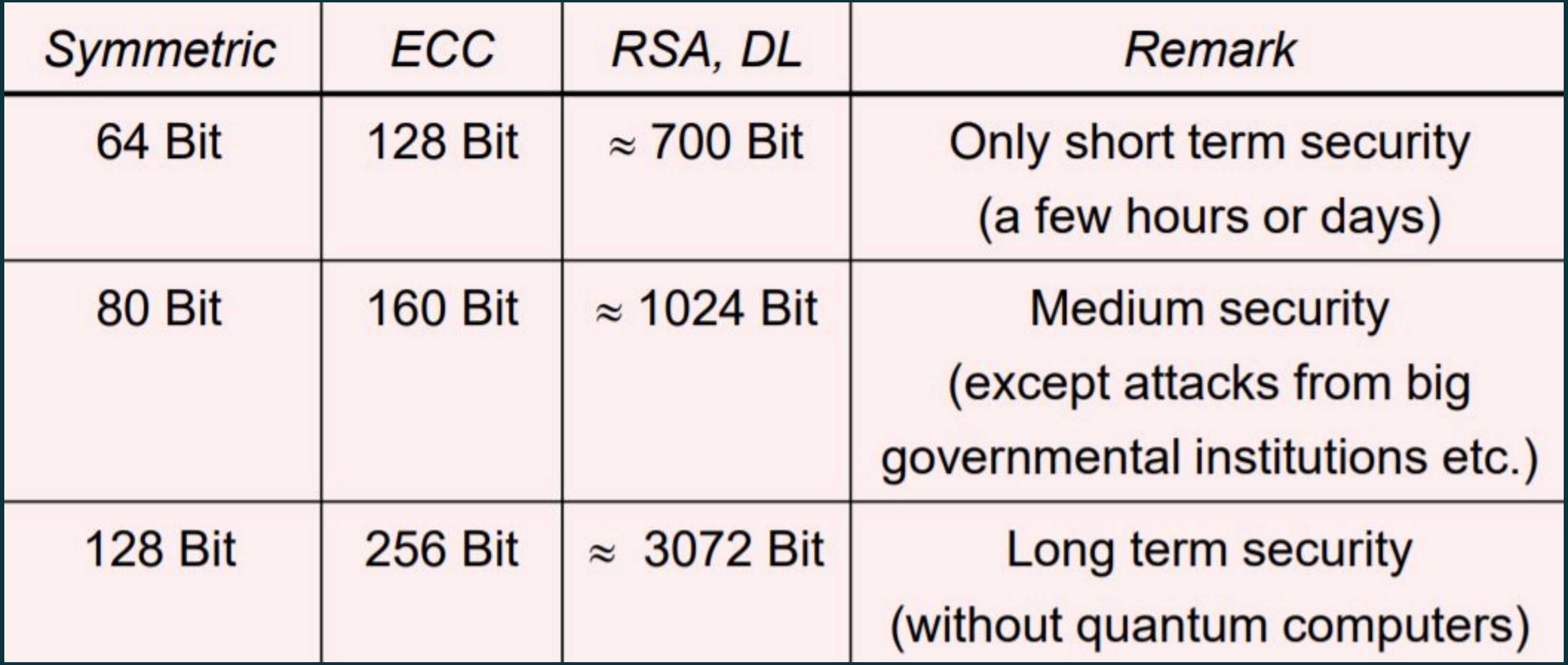
message x

$$y_2 = AES_K(x) \xrightarrow{\quad y_2 \quad} x = AES^{-1}_K(y_2)$$

How to build Public-Key Algorithms

- Asymmetric schemes are based on a "**one-way function**" f:
  - Computing $y = f(x)$ is computationally easy
  - Computing $x = f^{-1}(y)$ is computationally infeasible
- One-way functions are based on mathematically hard problems. Three main families:
  - **Factoring Integers** (RSA): Given a composite integer n, find its prime factors (Multiply two primes: easy)
  - **Discrete Logarithm** (Diffie-Hellman, Elgamal, DSA): Given a, y and m, find x such that $a^x = y \bmod m$ (Exponentiation a^x : easy)
  - **Elliptic Curves** (ECDH, ECDSA): Generalization of discrete logarithm

Key Lengths and Security Levels



Symmetric	*ECC*	*RSA, DL*	*Remark*
64 Bit	128 Bit	≈ 700 Bit	Only short term security (a few hours or days)
80 Bit	160 Bit	≈ 1024 Bit	Medium security (except attacks from big governmental institutions etc.)
128 Bit	256 Bit	≈ 3072 Bit	Long term security (without quantum computers)

RSA

Key Generation

- Choose 2 large primes p, q, compute
    - $n = pq$
    - $\varphi(n) = (p-1)(q-1)$
- Choose e such that $GCD(e, \varphi(n)) = 1$, compute
    - $d = e^{-1} \bmod \varphi(n)$
- Return $K_{pub} = (e, n)$, $K_{pr} = d$

Encryption & Decryption

- Encryption
  - $c = m^e \pmod{n}$
- Decryption
  - $m = c^d \pmod{n}$
- Correctness
  - $m^\varphi = (m^{p-1})^{q-1} = 1^{q-1} = 1 \pmod{p}$ (**Fermat's little theorem**)
  - $m^\varphi = (m^{q-1})^{p-1} = 1^{p-1} = 1 \pmod{q}$
  - $\Rightarrow m^\varphi = 1 \pmod{n}$ (**Chinese remainder theorem**)
  - $c^d = m^{ed} = m^{k\varphi+1} = m \pmod{n}$

Factorization Algorithm

- General Purpose
  - running time does not depend on the properties of n
  - fastest algorithm has running time of subexponential of logn

- Special Purpose
  - running time depends on the properties of n
  - |p-q| is small ⇒ Fermat's factorization
  - p-1 has small factors ⇒ Pollard's p-1 algorithm
  - p+1 has small factors ⇒ Williams' p+1 algorithm

Fermat's factorization

- $n = pq = (\frac{p+q}{2})^2 - (\frac{p-q}{2})^2$
- Number of steps:
  - $(p+q)/2 - \sqrt{n} = (\sqrt{p} - \sqrt{q})^2/2 = (\sqrt{n} - p)^2/2p$

```python
def fermatFactor(n):
    a = isqrt(n)
    b2 = a * a - n
    while not isqrt(b2)**2 == b2:
        a = a + 1
        b2 = a * a - n
    return a - isqrt(b2), a + isqrt(b2)
```

Pollard's p-1 Algorithm

- p-1 is B-smooth, i.e. p-1's biggest prime factor ≤ B
    - $p - 1 \mid 1 \times 2 \times \ldots \times B$
    - $2^{1\times2\times\ldots\times B} = 2^{k(p-1)} = 1 \pmod{p}$
    - $\mathrm{GCD}(2^{1\times2\times\ldots\times B} - 1, n) > 1$

```python
def pollard(n):
    a = 2
    b = 2
    while True:
        a = pow(a, b, n)
        d = gcd(a - 1, n)
        if 1 < d < n: return d
        b += 1
```

Factoring Tools

- http://factordb.com/index.php
- https://github.com/DarkenCode/yafu

How to Choose Public Exponent e

- e too small ⇒ direct e-th root, broadcast attack
- e too big ⇒ slow encryption
- Usually choose prime of form $2^x + 1$, e.g. $2^{16} + 1 = 65537$
    - 16 + 1 calculations in Square and Multiply

```python
def Square_and_Multiply(x, y):
    if y == 0: return 1
    k = Square_and_Multiply(x, y //2) ** 2
    return k * x if y % 2 else k
```

Direct e-th Root

- m, e are small such that $m^e < n$
- Find e-th root of m^e in integral domain
- Requrie random padding on m

Franklin-Reiter related-message attack

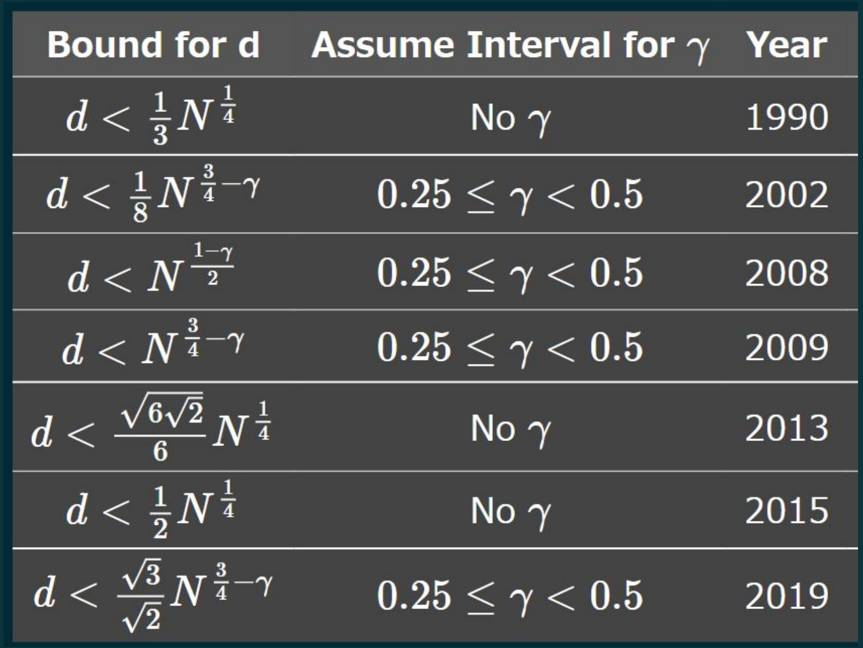
- e is small, $m_1 = f(m_2)$ for some linear polynomial $f = ax+b$
  - $c_1 = m_1^e \pmod n$
  - $c_2 = m_2^e = (am_1 + b)^e \pmod n$
- Given (n, e, c_1, c_2, f), attacker can recover m_1, m_2 efficiently
  - m_1 is a root of $g_1(x) = x^e - c_1$
  - m_1 is a root of $g_2(x) = f(x)^e - c_2$
  - $(x - m_1)$ divides both g_1, g_2
  - $\mathrm{GCD}(g_1, g_2) = x - m_1$
- GCD can be computed in quadratic time in $e \cdot \log n$ using Euclidean algorithm

Broadcast Attack

- Same message m was encrypted 3 times using the encryption exponent e = 3 but different moduli n_1, n_2, and n_3
  - $m^3 = c_1 \bmod n_1$
  - $m^3 = c_2 \bmod n_2$
  - $m^3 = c_3 \bmod n_3$
  - Using CRT, $m^3 = c \bmod n_1n_2n_3$
  - Since $m^3 < n_1n_2n_3$, $m^3 = c \Rightarrow$ cube root
- Generally require e different ciphertext to recover m

How to Choose Private Exponent d

- d too small ⇒ Wiener's attack, Boneh-Durfee's attack



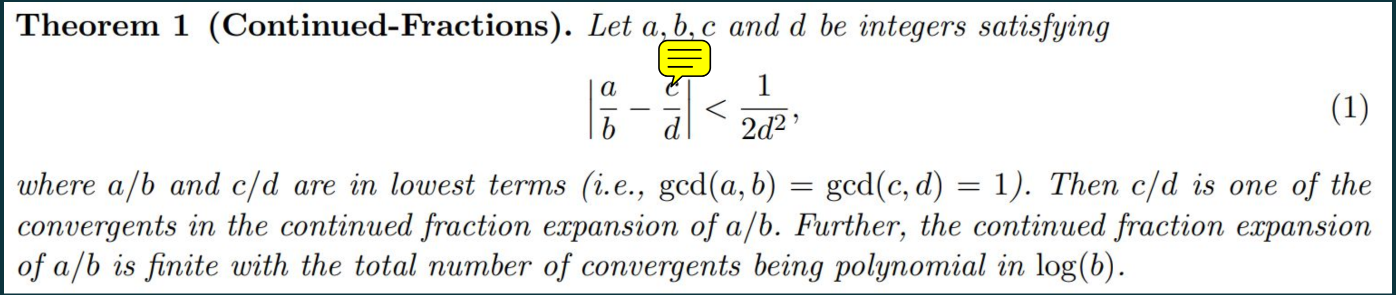
Bound for d	Assume Interval for γ	Year
$d < \frac{1}{3}N^{\frac{1}{4}}$	No γ	1990
$d < \frac{1}{8}N^{\frac{3}{4}-\gamma}$	$0.25 \leq \gamma < 0.5$	2002
$d < N^{\frac{1-\gamma}{2}}$	$0.25 \leq \gamma < 0.5$	2008
$d < N^{\frac{3}{4}-\gamma}$	$0.25 \leq \gamma < 0.5$	2009
$d < \frac{\sqrt{6\sqrt{2}}}{6}N^{\frac{1}{4}}$	No γ	2013
$d < \frac{1}{2}N^{\frac{1}{4}}$	No γ	2015
$d < \frac{\sqrt{3}}{\sqrt{2}}N^{\frac{3}{4}-\gamma}$	$0.25 \leq \gamma < 0.5$	2019

Reference: Ariffin, K., Rezal, M., Abubakar, S. I., Yunos, F., and Asbullah, M. A. (2019). New cryptanalytic attack on rsa modulus n = pq using small prime difference method.

Continued Fraction

- $$\frac{69}{420} = 0 + \cfrac{1}{6 + \cfrac{1}{11 + \frac{1}{2}}} \to [0; 6, 11, 2]$$
- $$\sqrt{19} = 4 + \cfrac{1}{2 + \cfrac{1}{1 + \frac{1}{3 + \ldots}}} = [4; 2, 1, 3, 1, 2, 8, 2, 1, 3, 1, 2, 8, \ldots]$$
- $$e = [2; 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, \ldots]$$

Wiener's Attack



Theorem 1 (Continued-Fractions). *Let a, b, c and d be integers satisfying*

$$\left|\frac{a}{b} - \frac{c}{d}\right| < \frac{1}{2d^2}, \tag{1}$$

where a/b and c/d are in lowest terms (i.e., $\gcd(a,b) = \gcd(c,d) = 1$). Then c/d is one of the convergents in the continued fraction expansion of a/b. Further, the continued fraction expansion of a/b is finite with the total number of convergents being polynomial in $\log(b)$.

- $ed = 1 + k\phi(N) = 1 + k(N - p - q + 1)$

$$\Rightarrow \frac{e}{N} - \frac{k}{d} = \frac{1}{dN} - \frac{k(p+q-1)}{dN}$$

- $k < d < \frac{1}{3}N^{\frac{1}{4}}, p + q - 1 < 3N^{\frac{1}{2}}$

$$\Rightarrow \left|\frac{e}{n} - \frac{k}{d}\right| < \left|\frac{k(p+q-1)}{dN}\right| < \frac{1}{2d^2}$$

Wiener's Attack (cont.)

- k/d will be one of the convergents in the continued fraction expansion of e/n
- $\varphi = (ed - 1)/k = (p - 1)(q - 1) = n - p - q + 1$
- Solve $x^2 - (n-\varphi+1)x + n = 0$
  - $x = p$ or q

Common Factor Attack

- $(e, n_1), (e, n_2)$ such that $GCD(n_1, n_2) \neq 1$
- Fast pairwise GCD computation
  - https://factorable.net/

Common Modulus Attack

- Same message, same modulus, different public exponent
  - $GCD(e_1, e_2) = 1$
  - $c_1 = m^{e1} \bmod n$
  - $c_2 = m^{e2} \bmod n$
- Bézout's identity
  - Exist a_1, a_2 such that $a_1e_1 + a_2e_2 = GCD(e_1, e_2) = 1$
  - a_1, a_2 can be found by extended Euclidean algorithm
- $c_1^{a_1}c_2^{a_2} = m^{a_1e_1+a_2e_2} = m \pmod n$

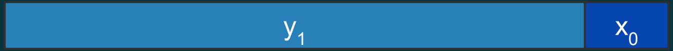
Chosen Ciphertext Attack

- Homomorphism
  - $f(x \circ y) = f(x) * f(y)$
- RSA encryption is homomorphic
  - $e(m_1 m_2) = (m_1 m_2)^e = e(m_1)e(m_2)$
- Server can decrypt anything except $c = m^e$
  - $d(2^e c) = 2m$
  - $2^{-1} \cdot 2m = m \pmod{n}$

LSB Oracle

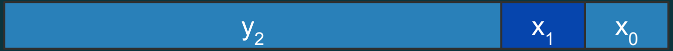
- Server can decrypt any c, but only return the least significant bit of m

LSB Oracle

- To get first bit (LSB), oracle
  - $c \rightarrow m$
- Inference
  - 

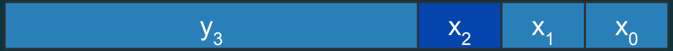
  - $m = 2y_1 + x_0$
  - $r = 2y_1 + x_0 = x_0 \pmod 2$
  - $\Rightarrow x_0 = r$

LSB Oracle (cont.)

- Oracle
  - $(2^{-1})^e c \rightarrow 2^{-1}m$
- Inference
  - 

  - $2^{-1}m = 2y_2 + x_1 + 2^{-1}x_0$
  - $r = [2y_2 + x_1 + 2^{-1}x_0]_{modn} \pmod 2$
    $= [2^{-1}x_0]_{modn} + x_1 \pmod 2$
  - $\Rightarrow x_1 = r - [2^{-1}x_0]_{modn} \pmod 2$

LSB Oracle (cont.)

- Oracle
  - $(2^{-2})^e c \to 2^{-2}m$
- Inference
  - 

  - $2^{-2}m = 2y_3 + x_2 + 2^{-1}x_1 + 2^{-2}x_0$
  - $r = [2y_3 + x_2 + 2^{-1}x_1 + 2^{-2}x_0]_{\text{modn}} \pmod 2$
    $= [2^{-2}x_0 + 2^{-1}x_1]_{\text{modn}} + x_2 \pmod 2$
  - $\Rightarrow x_2 = r - [2^{-2}x_0 + 2^{-1}x_1]_{\text{modn}} \pmod 2$

LSB Oracle (cont.)

- Can recover one bit every oracle
- Need log(n) oracles totally

Discrete Logarithm

Diffie-Hellman Key Exchange

- Set-up
  - Choose a large prime p
  - Choose an integer $\alpha \in \{2, 3, \ldots, p - 2\}$
  - Publish p and α

Diffie-Hellman Key Exchange

Choose random private key
$K_{prA} = a \in \{1, 2, \dots, p-1\}$

Choose random private key
$K_{prB} = b \in \{1, 2, \dots, p-1\}$

Compute
$A = \alpha^a \pmod{p}$

$$\xrightarrow{A}$$

$$\xleftarrow{B}$$

Compute
$B = \alpha^b \pmod{p}$

Caluculate common secret
$K = B^a = (\alpha^b)^a \pmod{p}$

Caluculate common secret
$K = A^b = (\alpha^a)^b \pmod{p}$

$$y = AES_K(x) \xrightarrow{y} x = AES^{-1}_K(y)$$

The Discrete Logarithm Problem

- Given a finite cyclic group $\mathbb{Z}_p^*$ of order $p - 1$ and a primitive element $\alpha \in \mathbb{Z}_p^*$ and another element $\beta \in \mathbb{Z}_p^*$
- The DLP is the problem of determining the integer $1 \leq x \leq p - 1$ such that

$$\alpha^x \equiv \beta \pmod{p}$$

The ElGamal Encryption Scheme

Choose $d = K_{prB} \in \{2, \dots, p-2\}$

Compute $\beta = K_{pubB} = \alpha^d \pmod p$

$$\xleftarrow{(p, \alpha, \beta)}$$

Choose $i = K_{prA} \in \{2, \dots, p-2\}$

Compute the ephemeral key
$K_E = K_{pubA} = \alpha^i \pmod p$

Compute the masking key
$K_M = \beta^i \pmod p$

Encrypt the message x
$y = x \times K_M \pmod p$

$$\xrightarrow{(K_E,\ y)}$$

Compute the masking key
$K_M = K_E^{\,d} \pmod p$

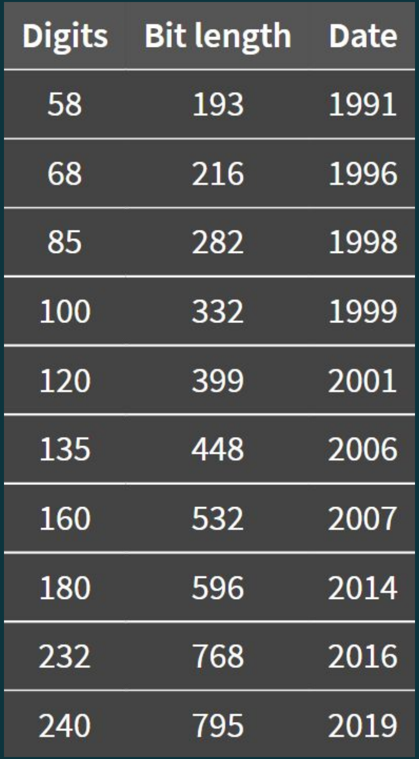
Decrypt the message
$x = y \times K_M^{-1} \pmod p$

Computational Aspects

- Key generation
  - Generation of prime p
  - p has size of at least 1024 bits
- Encryption
  - Requires two modular exponentiations and a modular multiplictation
  - All operands have the bitlength of $\log_2 p$
  - Efficient execution requires methods such as the square-and-multiply algorithm
- Decryption
  - Requires one modular exponentiation and one modular inversion
  - The inversion can be computed from the ephemeral key

Security

- Summary of records for computing discrete logarithms



Digits	Bit length	Date
58	193	1991
68	216	1996
85	282	1998
100	332	1999
120	399	2001
135	448	2006
160	532	2007
180	596	2014
232	768	2016
240	795	2019

Generalized DLP

- Generalized DLP
    - Let $(G, \circ)$ be an abelian group
    - Given $g, h \in G$, find x (if it exists) such that $g^x = h$
- The difficulty of this problem depends on the group G
    - Very easy: polynomial time algorithm
        - e.g. $(\mathbb{Z}_N, +)$
    - Rather hard: sub-exponential time algorithm
        - e.g. $(\mathbb{F}_p, \times)$
    - Very hard: exponential time algorithm
        - e.g. Elliptic Curve groups

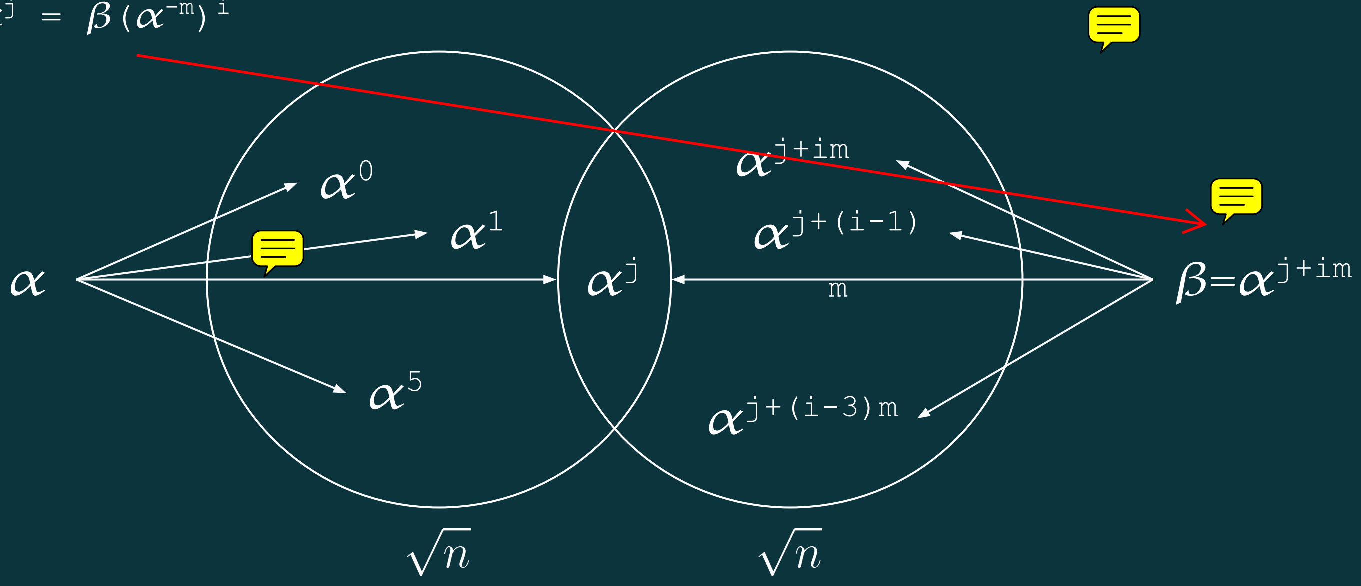
Attacks against the DLP

- Generic algorithms: Work in any cyclic group
    - Brute-Force Search
    - Baby-Step-Giant-Step
    - Pollard’s Rho Method
    - Pohlig-Hellman Method

- Non-generic Algorithms: Work only in specific groups, in particular in $\mathbb{Z}_p^*$
    - The Index Calculus Method

Baby-Step-Giant-Step

- We want to solve $\alpha^x = \beta$
- Rewrite $x = im + j$, where $m = \lceil \sqrt{n} \rceil$
  - $0 \le i < m,\ 0 \le j < m$
  - $\alpha^j = \beta(\alpha^{-m})^i$



Baby-Step-Giant-Step

Input: A cyclic group *G* of order *n*, having a generator *α* and an element *β*.

Output: A value *x* satisfying $\alpha^x = \beta$

```
1.  $m \leftarrow$ Ceiling($\sqrt{n}$)
2.  For all $j$ where $0 \le j < m$:
        1.  Compute $\alpha^j$ and store the pair ($j$, $\alpha^j$) in a table.
3.  Compute $\alpha^{-m}$.
4.  $\gamma \leftarrow \beta$.
5.  For all $i$ where $0 \le i < m$:
        1.  Check to see if $\gamma$ is the second component ($\alpha^j$) of any pair in
            the table.
        2.  If so, return $im + j$.
        3.  If not, $\gamma \leftarrow \gamma \cdot \alpha^{-m}$.
```

Pohlig-Hellman

- If $p-1 = p_1 p_2 \dots p_k$
  - $(g^{(p-1)/p_i})^{p_i} = 1$
  - $g_i = g^{(p-1)/p_i}$ has order p_i
  - $(g_i)^x = (g_i)^{(x \bmod p_i)} = y^{(p-1)/p_i} = h_i$
- Find x_i such that $(g_i)^x = h_i$
  - e.x. BSGS
  - $x_i = x \pmod{p_i}$
- Use CRT to recover x

- Runtime: $O(\sum_i (\log n + \sqrt{p_i}))$

$$\begin{tikzcd}
& y = g^x \pmod{p} \\
& \varphi(p) = p - 1 = p_1 \times \cdots \times p_k \arrow[dl, bend right] \arrow[dr, bend left] \arrow[d, phantom, "\text{Pohlig-Hellman}"] & \\
x \pmod{p_1} \arrow[dr] & \cdots \arrow[d, phantom, "\text{CRT}"] & x \pmod{p_k} \arrow[dl] \\
& x \pmod{\varphi(p)} &
\end{tikzcd}$$

Pohlig-Hellman

Input: A cyclic group *G* of order $n = p_1 \ldots p_r$, having a generator *g* and an element *h*.

Output: A value *x* satisfying $\alpha^x = \beta$

```
1.  For all i where $1 \leq i \leq r$:
        1.  Compute $g_i = g^{n/p_i}$
        2.  Compute $h_i = h^{n/p_i}$
        3.  Use BSGS to compute $x_i$ such that $g_i^{x_i} = h_i$
2.  Solve the CRT
        $x \equiv x_i \pmod{p_i} \quad \forall i \in \{1, \ldots, r\}.$
3.  Return $x$
```

Elliptic Curve

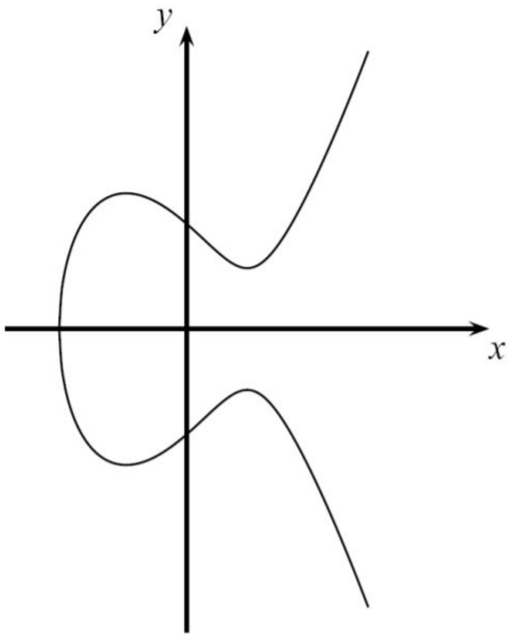
Elliptic Curve

- Elliptic curves are polynomials that define points based on the (simplified) Weierstraß equation:

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

  for parameters a, b that specify the exact shape of the curve

- On the real numbers and with parameters a, b ∈ $\mathbb{R}$, an elliptic curve looks like this



Example: $y^2 = x^3 - 3x + 3$ over ***R***

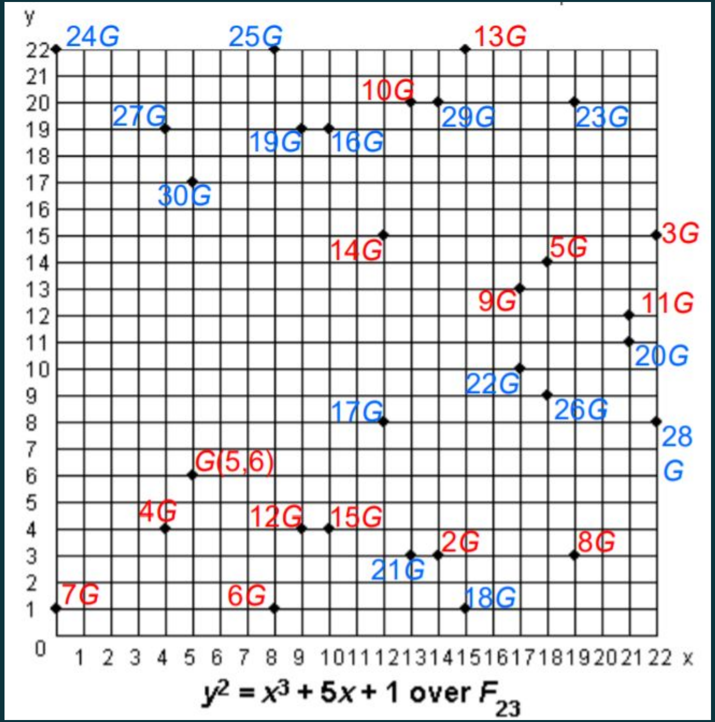
Elliptic Curve

- In cryptography, we are interested in elliptic curves modulo a prime p
- The elliptic curve over $\mathbb{Z}_p$, p > 3 is the set of all pairs (x,y) $\in \mathbb{Z}_p$ which fulfill

$$y^2 = x^3 + ax + b \pmod{p}$$

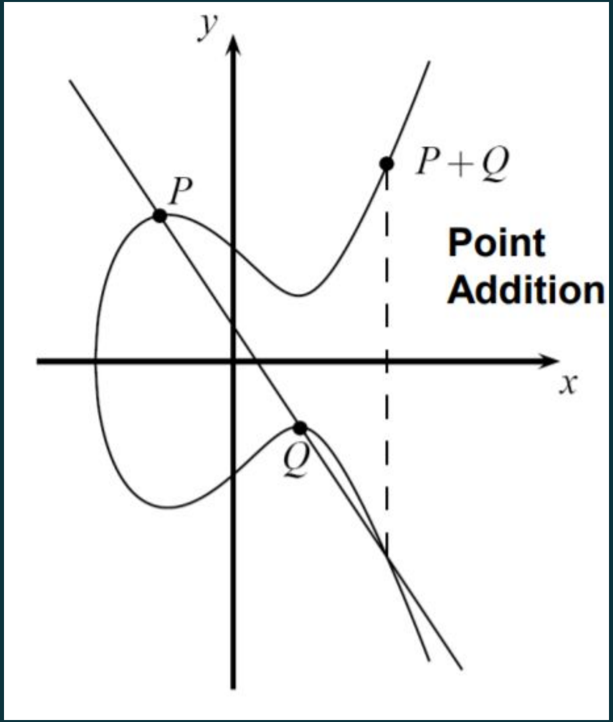
together with an imaginary point at infinity θ, where

$$4a^3 + 27b^2 \neq 0 \pmod{p}$$



Elliptic Curve

- Generating a group of points on elliptic curves based on point addition operation P + Q = R, i.e., $(x_P, y_P) + (x_Q, y_Q) = (x_R, y_R)$
- Geometric Interpretation of point addition operation
  - Draw straight line through P and Q; if P = Q use tangent line instead
  - Mirror third intersection point of drawn line with the elliptic curve along the x-axis



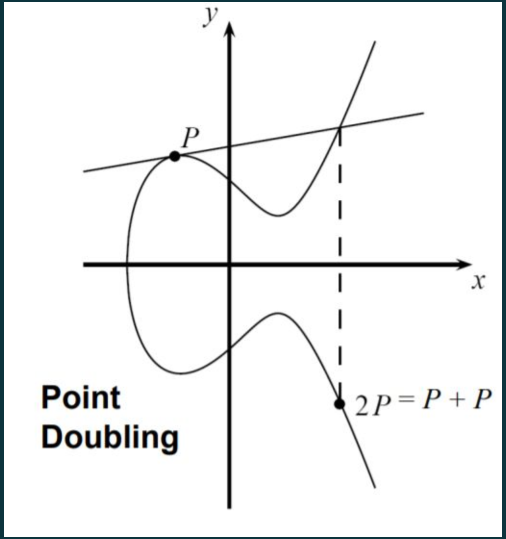
Elliptic Curve

- Elliptic Curve Point Addition and Doubling Formulas

$$s = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \mod p \text{ (addition)} \\ \frac{3x_1^2 + a}{2y_1} \mod p \text{ (doubling)} \end{cases}$$

$$x_3 = s^2 - x_1 - x_2$$

$$y_3 = s(x_1 - x_3) - y_1$$



Elliptic Curve

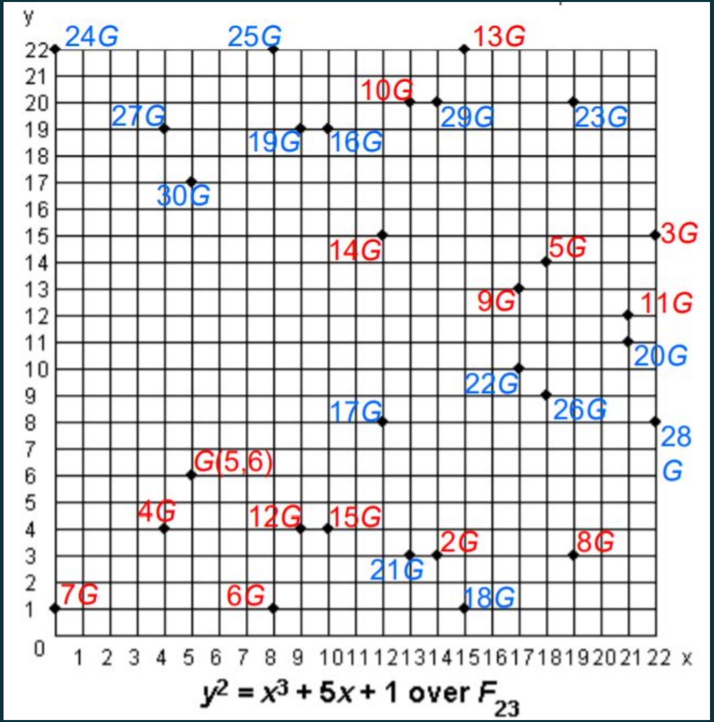
- Example: Compute 2G = G + G = (5, 6) + (5, 6) = (x_3 , y_3)

$$s = \frac{3x_1^2 + a}{2y_1} = (3 \cdot 5^2 + 5)(2 \cdot 6)^{-1} = 1 \cdot 2 = 22 \pmod{23}$$

$$x_3 = s^2 - x_1 - x_2 = 22^2 - 5 - 5 = 14 \pmod{23}$$

$$y_3 = s(x_1 - x_3) - y_1 = 22(5 - 14) - 6 = 3 \pmod{23}$$

- The points on an elliptic curve and the point at infinity θ form cyclic groups
- This elliptic curve has order #E = |E| = 31



Number of Points on an Elliptic Curve

- Hasse's Theorem:
  - Given an elliptic curve modulo p, the number of points on the curve is denoted by #E and is bounded by

$$p + 1 - 2\sqrt{p} \leq \#E \leq p + 1 + 2\sqrt{p}$$

- The number of points is "close to" the prime p
  - To generate a curve with about 2^{160} points, a prime with a length of about 160 bits is required

ECDLP

- Cryptosystems rely on the hardness of the Elliptic Curve Discrete Logarithm Problem (ECDLP)
  - Given an element P and another element Q on an elliptic curve E. The ECDLP problem is finding the integer d, where $1 \leq d \leq \#E$ such that

$$P + P + \ldots + P = dP = Q$$

- Cryptosystems are based on the idea that d is large and kept secret, and attackers cannot compute it easily
- If d is known, an efficient method to compute the point multiplication dP is required to create a reasonable cryptosystem

Double-and-Add Algorithm

- Example 25P = (11001_2)P
  - θ + θ = θ #DOUBLE
  - θ + P = P #ADD
  - P + P = 2P
  - 2P + P = 3P
  - 3P + 3P = 6P
  - #NO ADD
  - 6P + 6P = 12P
  - #NO ADD
  - 12P + 12P = 24P
  - 24P + P = 25P

```python
def Double_and_Add(d, P):
    bits = bin(d)[2:]
    Q = θ
    for bit in bits:
        Q = Q + Q
        if bit == "1":
            Q = Q + P
    return Q
```

Elliptic Curve Diffie-Hellman Key Exchange

- ECDH

Given a prime p, a suitable elliptic curve E and a point $P = (x_P, y_P)$

Choose random private key
$K_{prA} = a \in \{1, 2, \dots, \#E-1\}$

Choose random private key
$K_{prB} = b \in \{1, 2, \dots, \#E-1\}$

Compute
$A = aP = (x_A, y_A)$

$\xrightarrow{A}$

$\xleftarrow{B}$

Compute
$B = bP = (x_B, y_B)$

Caluculate common secret
$K = aB = a(bP)$

Caluculate common secret
$K = bA = b(aP)$

Parameter Choice

- E has smooth order
  - Pohlig-Hellman
- E has order equal to p (anomalous curve)
  - Transform the DLP to $(\mathbb{F}_p, +)$
  - Smart’s Attack
- E is singular
  - Node: Transform the DLP to $(\mathbb{F}_p, \times)$
  - Cusp: Transform the DLP to $(\mathbb{F}_p, +)$

Pohlig-Hellman (on ECC)

Input: Elliptic Curve E of order $n = p_1 \ldots p_r$, having a generator G and an element P.

Output: A value d satisfying $dP = Q$

```
1. For all i where $1 \leq i \leq r$:
    1. Compute $G_i = (n/p_i)G$
    2. Compute $P_i = (n/p_i)P$
    3. Use BSGS to compute $d_i$ such that $d_iG_i = P_i$
2. Solve the CRT
    $d \equiv d_i \pmod{p_i} \quad \forall i \in \{1, \ldots, r\}.$
3. Return $d$
```

Anomalous Curve

- Augmented Point Addition
  - Each Point P on curve are associated with a value in $\mathbb{F}_p$, i.e. [P, a]
  - Addition is computed as follow:

$$[P, a] \oplus [Q, b] = [P + Q, a + b + s_{PQ} \pmod{p}]$$

where s_{PQ} is the slope of PQ (tangent line if P = Q)

$s_{PQ} = 0$ if $Q = -P$ or $P = \theta$ or $Q = \theta$

- Define $\varphi(P) = \alpha$ where
  - $p[P, 0] = [P, 0] \oplus [P, 0] \oplus \dots \oplus [P, 0] = [\theta, \alpha]$

Reference: I.A. Semaev. Evaluation of discrete logarithms on some elliptic curves.
https://www.ams.org/journals/mcom/1998-67-221/S0025-5718-98-00887-4/S0025-5718-98-00887-4.pdf

Anomalous Curve

- φ is a homomorphism
  - $\varphi(P + Q) = \varphi(P) + \varphi(Q)$
- Compute $\varphi(P) = \alpha$, $\varphi(Q) = \beta$, since φ is homomorphic
  - $\beta = \varphi(Q) = \varphi(dP) = d\varphi(P) = d\alpha$
- d can be easily calculated
  - $d = \beta\alpha^{-1} \pmod{p}$

Reference: I.A. Semaev. Evaluation of discrete logarithms on some elliptic curves.
https://www.ams.org/journals/mcom/1998-67-221/S0025-5718-98-00887-4/S0025-5718-98-00887-4.pdf

Smart's Attack

- Easy implemenation on Sage
  - https://crypto.stackexchange.com/questions/70454/why-smarts-attack-doesnt-work-on-this-ecdlp
- Recommended reading
  - J. Monnerat, *Computation of the discrete logarithm on elliptic curves of trace one - Tutorial*
  - https://core.ac.uk/download/pdf/147902645.pdf

Singular Curve

- A curve is singular if $4a^3 + 27b^2 = 0 \pmod{p}$
  - ECDLP becomes much easier if curve is singular
- There are two types of singular point
  - Node: $y^2 = (x - \alpha)^2(x - \beta)$
  - Cusp: $y^2 = x^3$

Node

- $y^2 = (x - \alpha)^2(x - \beta)$
- Define $\varphi(P(x, y)) = \dfrac{y + \sqrt{\alpha - \beta}(x - \alpha)}{y - \sqrt{\alpha - \beta}(x - \alpha)}$
- If we have homomorphism $\varphi(P + Q) = \varphi(P) \times \varphi(Q)$
  - $\varphi(dP) = \varphi(P)^d$
  - Reduce to DLP on $(\mathbb{F}_p, \times)$

Prooving $\varphi(P + Q) = \varphi(P) \times \varphi(Q)$

$y^2 = (x - \alpha)^2(x - \beta)$ $\quad$ $(x, y) \mapsto \dfrac{y + \sqrt{\alpha - \beta}(x - \alpha)}{y - \sqrt{\alpha - \beta}(x - \alpha)}$

- $X = x - \alpha$, $A = 2\sqrt{(\alpha - \beta)}$, $Y = y - AX/2$

$Y^2 + AXY - X^3 = 0$ $\quad$ $(X, Y) \mapsto 1 + AX/Y$

- $X \to X/Z$, $Y \to Y/Z$ (homogenize)

$Y^2Z + AXYZ - X^3 = 0$ $\quad$ $(X, Y, Z) \mapsto 1 + AX/Y$

- $X = A^2X' - A^2Y'$, $Y = A^3Y'$, $Z = Z'$

$X'Y'Z' - (X' - Y')^3 = 0$ $\quad$ $(X', Y', Z') \mapsto X'/Y'$

- $Y' = 1$, $\mathscr{x} = X'/Y'$, $\mathscr{z} = Z'/Y'$ (dehomogenize)

$\mathscr{x}\mathscr{z} - (\mathscr{x} - 1)^3 = 0$ $\quad$ $(\mathscr{x}, \mathscr{z}) \mapsto \mathscr{x}$

Reference: The Arithmetic of Elliptic Curves, Silverman, pp 55-58
http://www.pdmi.ras.ru/~lowdimma/BSD/Silverman-Arithmetic_of_EC.pdf

Prooving φ(P + Q) = φ(P) × φ(Q) (cont.)

- If a line $z = ax + b$ intersect the curve on (x_1, z_1), (x_2, z_2), (x_3, z_3), then x_1, x_2, x_3 are the roots of

$$x(ax + b) - (x - 1)^3 = -x^3 + (a+3)x^2 + (b-3)x - 1$$

- We have $x_1x_2x_3 = 1$

$$\begin{aligned}\varphi(P + Q) &= \frac{-y_3 + \sqrt{\alpha - \beta}(x_3 - \alpha)}{-y_3 - \sqrt{\alpha - \beta}(x_3 - \alpha)} \\ &= 1/x_3 \\ &= x_1x_2 \\ &= \varphi(P) \times \varphi(Q)\end{aligned}$$

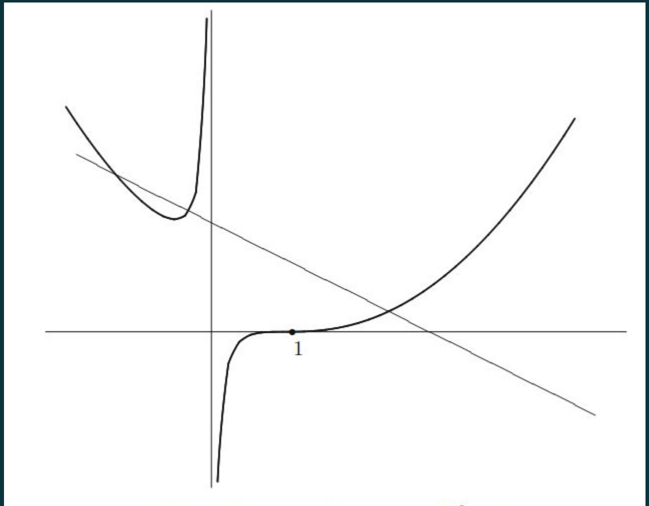


Figure 3.4: The curve $xz - (x-1)^3 = 0$.

Reference: The Arithmetic of Elliptic Curves, Silverman, pp 55-58
http://www.pdmi.ras.ru/~lowdimma/BSD/Silverman-Arithmetic_of_EC.pdf

Cusp

- $y^2 = x^3$
- Define $\varphi(P(x, y)) = x/y$
- If we have homomorphism $\varphi(P + Q) = \varphi(P) + \varphi(Q)$
    - $\varphi(dP) = d\varphi(P)$
    - Reduce to DLP on $(\mathbb{F}_p, +)$
    - $Q = dP \Rightarrow d = \varphi(Q)\varphi(P)^{-1}$

Prooving φ(P + Q) = φ(P) + φ(Q)

$y^2 = x^3$ $\qquad$ $(x, y) \mapsto x/y$

- X -> X/Z, Y -> Y/Z (homogenize)

$Y^2Z - X^3 = 0$ $\qquad$ $(X, Y, Z) \mapsto X/Y$

- $Y' = 1$, $\mathscr{x} = X'/Y'$, $\mathscr{z} = Z'/Y'$ (dehomogenize)

$\mathscr{z} - \mathscr{x}^3 = 0$ $\qquad$ $(\mathscr{x}, \mathscr{z}) \mapsto \mathscr{x}$

- If a line $\mathscr{z} = a\mathscr{x} + b$ intersect the curve on $(\mathscr{x}_1, \mathscr{z}_1)$, $(\mathscr{x}_2, \mathscr{z}_2)$, $(\mathscr{x}_3, \mathscr{z}_3)$, then $\mathscr{x}_1, \mathscr{x}_2, \mathscr{x}_3$ are the roots of

$$(a\mathscr{x} + b) - \mathscr{x}^3$$

- We have $\mathscr{x}_1 + \mathscr{x}_2 + \mathscr{x}_3 = 0$

$$\varphi(P + Q) = -\mathscr{x}_3 = \mathscr{x}_1 + \mathscr{x}_2 = \varphi(P) + \varphi(Q)$$