Dec1

Tuesday, December 1, 2020 2:03 PM

Lenstra's Elliptic Curve Factorization

Goal Factor N.

Setup E elliptic curve

$P \in E(\mathbb{Z}/N\mathbb{Z})$ (not $\mathcal{O}$)

Computing

$P, 2P, 3P, 4P, \ldots$

Eventually got stuck!

Tried to add $(x_1, y_1) + (x_2, y_2)$

Need: $\frac{\Delta y}{\Delta x} \leftarrow \underline{\underline{\Delta x}} \notin (\mathbb{Z}/n\mathbb{Z})^*$

→ i.e. $\gcd(\Delta x, N) \neq 1$ ←

→ factored N !!!

What's going on?

Say $p \mid \Delta x$. & N.

$P + Q$ on $E(\mathbb{Z}/N\mathbb{Z})$

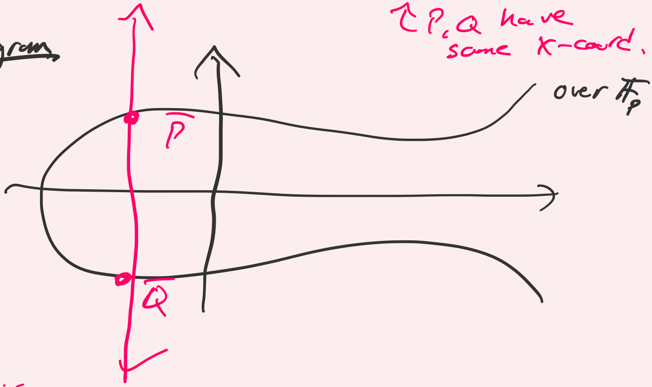
$$\rightsquigarrow \text{reduce mod } p$$

$\overline{P}+\overline{Q}$ on $E(\mathbb{F}_p)$

p divides $\Delta x \iff \underline{\Delta x \equiv 0} \bmod p.$

↑ P, Q have same x-coord.

Diagram



$\Longrightarrow \overline{P}+\overline{Q}=\mathcal{O}.$

Looking For

$$\begin{tikzcd}
P \in E(\mathbb{Z}/N\mathbb{Z}) \arrow[r, rightsquigarrow] \arrow[d, rightsquigarrow] & \overline{P} \in E(\mathbb{F}_p) \arrow[d, rightsquigarrow] \\
nP \ \underline{\text{get stuck}} & n\overline{P} = \mathcal{O} \arrow[l, rightsquigarrow, "p \mid \Delta x"']
\end{tikzcd}$$

Point $P \in E(\mathbb{Z}/N\mathbb{Z})$ with small order in $E(\mathbb{F}_p)$.

Rmks:

Pollards $p-1$ algorithm.
$a \in \mathbb{Z}/N\mathbb{Z}$ s.t. a had small order in $\mathbb{Z}/p\mathbb{Z}$ for some $p \mid N$.

Method:

- Take $P \in E(\mathbb{Z}/N\mathbb{Z})$

$|P|$ much more likely to divide $n!$ then n.

- Compute $Q_n = n! P$ for various n.

(notice $Q_{n+1} = (n+1) \cdot Q_n$.)

↑ Double & add.

1) Work (keep going)
2) Won't work, tried to divide by d w/ $g = \gcd(d, N) \neq 1$
3) If $1 < g < N$ I win

   If $g = N$ stuck

   ↑ won't happen often.

↪ $E(\mathbb{F}_q)$ & $E(\mathbb{F}_p)$ have different sizes so unlikely tha P have same order mod p & q.

Concern

Need E & need $P \in E(\mathbb{Z}/N\mathbb{Z})$.

Given $E: \; y^2 = \overbrace{x^3 + Ax + B}^{f(x).}$.

Pick x. Need square root of $f(x)$ mod N to get y.

Proved earlier

Square roots modulo N ⟷ Factor N.

Solution Takehome 2 (Problem 4)

① Start w/ $P = (a, b) \in (\mathbb{Z}/N\mathbb{Z})^2$

② Random $A \in \mathbb{Z}/N\mathbb{Z}$

③ use $b^2 = a^3 + Aa + B$ & solve for B.

FindEllipticCurveAndPoint *

Lenstra's Elliptic Curve Factorization Algorithm

Input N to be factored.

① Find E & $P \in E(\mathbb{Z}/N\mathbb{Z})$ $P \neq O$ using TH2 Prob 4 *

② Loop $j = 2, 3, 4, \dots$

③ $Q = jP \in E(\mathbb{Z}/N\mathbb{Z})$ Set $P = Q$

④ If computation fails ⇝ find $d > 1$ w/ $d \mid N$ *

⑤ If $\underline{d < N}$ we win. Else go back to step ① w/ new E & P.

"proof"

Computation is algebraic. Only way it could fail is if $\div \Delta x$ (or $\div 2y$) fails, whence Euclidean algorithm provides d.

Remarks on Time Complexity ←

Factorization Algs

① Naive (try 2, try 3, try 4, ...)

⇝ $O(p)$ p smallest prime dividing N.

② Pollard's ρ method

⇝ $O(\sqrt{p})$ p smallest prime dividing N.

③ Pollards $p-1$

⇝ $O(q)$ q smallest prime dividing $p-1$ some p prime dividing N.

①-③ All exponential.

④ Quadratic Sieve.

$$O\left(e^{\sqrt{\log N \log\log N}}\right)$$

Chap 3

⑤ Lenstra's

$$O\left(e^{\sqrt{2\log p \log\log p}}\right)$$

p smallest prime dividing N.

If $N=pq$, $p \approx q$ these are ≈ same order

④ faster.

Else Lenstra is faster.

⚠ Quadratic Sieve has faster implementation (smaller coeffs)

Hash Functions: A brief introduction

Motivation: Digital Signatures

Sam: The signer. (K^{pub}, K^{priv})

Victor: The verifier.

Signing Process

Sam take a document D & produce a signed document D^{sig}. $S(D, K^{priv}) = D^{sig}$

Verification Process

Victor takes D & D^{sig} & verifys that D^{sig} is D signed by Sam.

$$V(D, D^{sig}, K^{pub}) = \begin{cases} \text{True} & \leftarrow \text{if D signed is } D^{sig} \\ \text{False} & \leftarrow \text{else.} \end{cases}$$

Problem

$D \in \mathbb{F}_p^*$. If $p \approx 2^{1024}$ then

D limited to 128 bytes (128 chars!)

what If I want to download & install a program $\underset{D}{\wedge}$ is 1GB in size??

↑ Too big to sign.

Potential Solutions

Option 1: To slow

Break D down into peices $D_1, D_2, \ldots$ each 128 bytes.

Sign each one $D_1^{sig}, D_2^{sig}, \ldots$

I verify each peice.

Issue Need ≈ 10 million 128 byte peices!

Sign 10 million docs
Verify 10 million docs
Whole ness.

Option 2 (BAD) (unsafe).

Just sign D_1.

Eve can send $D' = \underline{D_1 E_2 E_3 \cdots}$

& D_1^{Sig} ← intercepted.

Verificitaton will match D_1^{Sig} & D

but E_i are not safe.

Option 3: Hash Functions

"smart option 2"

Replace D w/ something smaller. But each D_i has an effect, so eve can't mess w/ it w/o changing smaller thing.

Idea

Build a public function

avg $\underline{n}$

$$\text{HASH}: \begin{pmatrix}\text{Docs of} \\ \text{any length}\end{pmatrix} \longrightarrow \{0,1\}^{1024}$$

s.t.

① Easy to compute even for huge documents.

② Hard to invert.

Lack of meaningful collisions

Lot's of documents D, D' have $\text{HASH}(D) = \text{HASH}(D')$ (collisions)

Hard to find collisions.

Ex Option 2 fails property 2.

$D = D_1 D_2 D_3 \dots$ & $D' = D_1 E_2 E_3 \dots$

have same stupid hash.

Eve can slip whatever she wants into D' after D_1.

Impt idea

Modifying D should almost certainly modify D'.

Implementation

Sam

Computes $\text{HASH}(D)$ & signs it $\left(\text{HASH}(D)^{sig}\right)$

Victor

Computes $\text{HASH}(D)$ & verifies that $\text{HASH}(D)^{sig}$ is correct signature.

If Eve iserts nevarious code into D. $\rightsquigarrow D'$

Hash(D') doesn't match signed doc.

Importantly (hard to invert *)

1) If D is meaninful. Hash(D) = Hash(D') then D' is likely meaningless.

2) Even if they are both meaninful, Eve's best way to find D' should be Guess & Check. ($O(2^n)$)

Example of what to avoid

D = "I give Eve \$5"

D' = "I give Eve \$5,000,000"

I sign Hash(D)

If Hash(D') = Hash(D) then Eve could go to bank, present them w/ D' & $Hash(D)^{sig} = Hash(D')^{sig}$ & Bank verifies signature & hands over my 5,000,000 to Eve.

We should design Hash function to avoid these sorts of collisions

How to build

Introduce some bitstring operations

① xor (exlusive or)
$\oplus$

$\beta, \beta' \in \{0,1\}$ to bits.

$$\beta \text{ xor } \beta' = \beta \oplus \beta' = \begin{cases} 0 & \text{if } \beta = \beta' \\ 1 & \text{else.} \end{cases}$$

(addition modulo 2)

If $\alpha = \beta_1 \beta_2 \cdots \beta_n$ $\alpha' = \beta_1' \beta_2' \cdots \beta_n'$

2 bitstrings. $(\beta_i \in \{0,1\})$

$$\alpha \oplus \alpha' = [\beta_1 \oplus \beta_1'][\beta_2 \oplus \beta_2'] \cdots [\beta_n \oplus \beta_n']$$

Ex $\alpha = 1011 \leftarrow$
$\alpha' = 0110$

$$\alpha \oplus \alpha' = [1 \oplus 0][0 \oplus 1][1 \oplus 1][1 \oplus 0]$$

$$= 1101$$

Remark VERY FAST TO COMPUTE

faster than $+, \times, -, \ldots$

② Concatenation

$\alpha = \beta_1 \beta_2 \cdots \beta_n \qquad \alpha' = \beta_1' \beta_2' \cdots \beta_m'$

2 bitstring s

$$\alpha \| \alpha' = \beta_1 \beta_2 \cdots \beta_n \beta_1' \cdots \beta_m'$$

Ex $\quad \alpha \| \alpha' = 1011010110$

An example of a hash function

$$\text{HASH}: \begin{pmatrix} \text{Docs of} \\ \text{arb length} \end{pmatrix} \longrightarrow \{0,1\}^n$$

$$D \longmapsto \mathcal{H}(D)$$

Input Public mixing function

- $\mathcal{M}: \{0,1\}^n \longrightarrow \{0,1\}^n$

  Fast to compute.

  ad hoc
  digit rotation
  & xoring

- Value $H_0 \in \{0,1\}^k$

Step 1

Add zeros to D untill the # of bits is divible by n.

$$D = D_1 \| D_2 \| \cdots \| D_k \qquad D_i \in \{0,1\}^n$$

Step 2 $\quad i = 1, \ldots, k$

$$H_i = \underline{H_{i-1}} \oplus \mathcal{M}(D_i)$$

Step 3

Returne $\mathcal{H}(D) = H_k$

---

* Easy to compute Hash(D)
* Hard to reverse engineer a document w/ a given Hash.
* Hard to control document.

Rmk

The mixing function could also depend on the H_i in some complicated way.

Example (SHA-1) $n = 512$

$H_0 = h_0 || h_1 || \cdots || h_4$ fixed values.

To mix

$D_i = w_0 || w_1 || \cdots || w_{15}$

Rotate letters of w_i to form

$v_0, v_1, \ldots, v_{79}$

For $i = 0, \ldots, 79$

$a = h_0$, $b = h_1, \ldots, e = h_4$

Rotate leters of $a, b, \dots, e$

Permute $a, b, \dots, e$.

Replace a w/ $a \oplus \sigma_i$:

Let $h_0 = h_0 \oplus a$

$h_1 = h_1 \oplus b$

$h_2 = h_2 \oplus c$

$h_3 = h_3 \oplus d$

$h_4 = h_4 \oplus e$

$\dashrightarrow H_i = h_0 \| h_1 \| \dots \| h_4$

Best known alg to invert SHA-1 is $\Theta(2^n)$.