Attacking Discrete-Log

Last Time:

- Mining P's & Q's

$$p_1 q_1 \quad p_1 q_2 \quad p_3 q_3 \quad p_4 q_4 \;\to\; \text{GCD tree} \to p_1$$

- Coppersmith's attack: Infineon edition
  "optimized" keygen:
  $p \leftarrow k \cdot M + 65537^a \% M$
  $q \leftarrow \ell \cdot M + 65537^b \% M$

Bad Randomness => Broken Crypto

Today: Direct attacks on discrete log

- Generic group attacks
  - Baby-step, giant-step
  - Pollard's rho
  - Shoup's lower bound

  matching!
- $\mathbb{Z}_p^*$ attack:
  - index calculus

Generic Discrete-Log Attack

← works for any group

- Fix group G, order q, generator g.
- given: $h \leftarrow g^x$, $x \xleftarrow{\$} \mathbb{Z}_q$
- find: x.

Warm-up Attack: Naive

Brute-force search:
for $i \in \{0, \dots, q-1\}$:
  if $g^i = h$: return i

$$g^0 \to g^1 \to g^2 \cdots \to g^x = h$$

uses "baby steps": $g^i \to g^{i+1}$, but "giant steps": $g \to g^n$ are cheap too: $\leq \log_2 n$ group-ops!

Baby-Step, Giant-Step [Shanks '71]

Idea: precompute $\sqrt{q}$-spaced signposts

$$g^0 \to g^{\sqrt{q}} \to g^{2\sqrt{q}} \cdots g^{i\sqrt{q}}$$

Map M:

$$g^{i\sqrt{q}} \mapsto i\sqrt{q}$$

$(i \in \{0, \dots, \sqrt{q}\})$

Offline phase (G, g, q):
  build M
  → $O(\sqrt{q} \log q)$ g-ops
  → $O(\sqrt{q} \log q)$ bits in M what are we assuming?

Online phase (h):
  find $j \in \{0, \dots, \sqrt{q}\}$
  s.t. $hg^j \in M$
  and return $M[hg^j] - j$

$$M[hg^j] - j = \log_g(hg^j) - j = \log_g h + j \log_g g - j = \log_g h \checkmark$$

Time: $O(\sqrt{q} \log q)$
  ← b/c table lookup (w/ tree)

Space is bottleneck

Take $q = 2^{80}$ (real: $q = 2^{256}$)
time: $\frac{3}{2}\sqrt{q} \log \sqrt{q}$ g-ops $= \frac{3}{2} 2^{40} \cdot 40$ g-ops $= 94$ cpu-years   ← parallelize!   (curve25519-dalek: $45 \mu s$/g-op)
space: $\sqrt{q}$ g-elements $= 2^{40} \cdot 32\,B \cdot 2 \approx 70\,TB$ (in one table!)
can we reduce space?

Pollard's Rho Algorithm

Idea: random walk + cycle finding!

$u_0 = g^{a_0} h^{b_0} \to u_1 = g^{a_1} h^{b_1} \to \cdots \to u_i = g^{a_i} h^{b_i} \to \cdots \to u_j = g^{a_j} h^{b_j}$

// equal!

$\Rightarrow g^{a_i} h^{b_i} = g^{a_j} h^{b_j} \Rightarrow a_i + \underbrace{x}_{\log_g h} b_i = a_j + x b_j$

$\Downarrow$

$x = \dfrac{a_j - a_i}{b_i - b_j}$

For a random transition $O(\sqrt{q})$ steps give a cycle w/ all but negl prob.

Need:
1. a pseudo-random transition fn
2. cycle finding.

1. Pseudo-random steps
   - can't hash $H(g^a h^b) \to g'$
     - lose dlog information for g'!
   - simple alternative
     - split G into random $G = S_0 \uplus S_g \uplus S_h$

$$\text{step}(u = g^a h^b) = \begin{cases} g^{2a} h^{2b} & u \in S_0 \\ g^{a+1} h^b & u \in S_g \\ g^a h^{b+1} & u \in S_h \end{cases}$$

   effective when split is unrelated to G's structure

2. Cycle Finding: Floyd's tortoise & hare [Knuth '69]

Problem:
Given an infinite sequence $x_1, x_2, \ldots$ that eventually cycles.
find $i > j$ s.t. $x_i = x_j$

Solution:
two pointers: $t_i = x_i$ ← tortoise
$h_i = x_{2i}$ ← hare

distance between them: i

for cycle of length ℓ, $t_j = h_j$ for first j divisible by ℓ after t_i enters the cycle!

runtime: O(distance to cycle closure)

3. Analysis

$O(\sqrt{q})$ time

overwhelming success probability ← $\geq 1 - \text{negl}(\lambda)$

$O(1)$ space
$\Rightarrow$ 2 pointers!

Can we do better?

Shoup's Lower Bound [Shoup '97]

Any group-generic dlog attack requires $\Omega(\sqrt{q})$ g-ops to attain non-negl. success probability.
→ Baby-step, Giant-step & Pollard's Rho are time-optimal (* log factors)
→ Pollard's Rho is space-optimal why? :)
→ Uses **Generic Group Model**

→ Really nice proof. See Thm of today's reading.

Next: non-generic attacks!

Part II: Non-generic attacks

Warmup: $(\mathbb{Z}_p, +)$

Consider group $(\mathbb{Z}_p, +)$ ← prime

- elements in $\mathbb{Z}_p = \{0, 1, \dots, p-1\}$
- group operation: $a, b \mapsto (a+b) \% p$
- d-log problem:

  given $g, h \in \mathbb{Z}_p$, find x such that $\underbrace{g + \cdots + g}_{x \text{ times}} = h$

  - easy solution: division!

    if $h / g = k$, then $\underbrace{g + \cdots + g}_{k \text{ times}} = h$ (in $\mathbb{Z}_p$)

    ↑ in $\mathbb{Z}_p^*$

Math Facts

1. There are (asymptotically) $\frac{\beta}{\log \beta}$ primes in the first β integer
2. A number is "β-smooth" if its prime factors are all $\leq \beta$. There are $\frac{N}{u^u}$ β-smooth #s $\leq N$, for $u = \frac{\log N}{\log \beta}$

Index calculus $(\mathbb{Z}_p, \times)$ (aka $\mathbb{Z}_p^*$)

$\mathbb{Z}_p^*$:

elements from $\{1, 2, \dots, p-1\}$

under $\times \pmod p$

assume (today) that $q = \frac{p-1}{2}$ is prime "safe prime"

and that g generates an order q multiplicative subgroup $(|\{g, g^2, g^3, \dots, g^q = 1\}| = q)$

example: $\mathbb{Z}_7^*$

$p = 7$, $q = 3$, $g = 2$ generates $\{2, 4, 1\}$

Our goal:

- a sub-exponential (but super-poly) attack
- define $L_N(\alpha, c) = \exp(c \log^\alpha N (\log \log N)^{1-\alpha})$, so
  - $L_N(0, c) = (\log N)^c$ poly(log N) = poly(N's size)
  - $L_N(1, c) = N^c$ exp(log N)
  - $L_N(1/2, c) = \exp(c\sqrt{\log N \cdot \log \log N})$ "sub-exponential"
- We'll build a $L_q(1/2, 3)$ attack

Plan of attack:

1. Let $B = \{2, 3, 5, \dots, p_t\}$ contain all primes $\leq \beta$ ← to be set later

   a "factorization basis"

   ex: $120 = 2^3 \cdot 3 \cdot 5$ factors in $\{2, 3, 5\}$

   $140 = 2^2 \cdot 5 \cdot 7$ does not

2. Compute $\log_g p_i$ for $i \in [t]$

   ↑ we'll explain this next

3. Compute $\log_g h$ from $\{\log_g p_i\}$
   - use "random self-reducibility"
   - sample $r \xleftarrow{\$} \mathbb{Z}_q$ until hg^r factors in B. That is, $g h^r = \prod_i p_i^{e_i}$

     $\to \log_g(hg^r) = \sum_i e_i \log_g p_i \to \log_g h = -r + \sum_i e_i \log_g p_i$
   - How many r do we need to sample?

     Math fact 2 implies: $\Pr[\beta\text{-smooth}] \approx 1/u^u$ $u = \frac{\log q}{\log \beta}$

     w/ $\beta = L_q(1/2, 1)$, you can show $\Pr[\beta\text{-smooth}] \gtrsim \beta$

     $\Rightarrow$ expected # of r-values $\leq \beta$.

How? I'm glad you asked.

$\beta = L_q(1/2, 1)$

$\log \beta = \sqrt{\log q \cdot \log \log q}$

$u = \sqrt{\log q} / \sqrt{\log \log q}$

$u^u = \exp(u \log u) = \exp\left(\sqrt{\frac{\log q}{\log \log q}} \log\left(\sqrt{\frac{\log q}{\log \log q}}\right)\right)$

$= \exp\left(\frac{1}{2}\sqrt{\frac{\log q}{\log \log q}} [\log \log q - \cancel{\log \log \log q}]\right)$

$\leq \exp\left(\frac{1}{2}\sqrt{\log q \cdot \log \log q}\right)$

$\leq \exp(\sqrt{\log q \cdot \log \log q}) = \beta$

notice the slack

better analysis? Yes!

- How much work per r-value?

  $\frac{\beta}{\log \beta} \cdot \text{polylog } \beta = \tilde{O}(\beta)$

  size of B → $\frac{\beta}{\log \beta}$; ↑ division time

  $\Rightarrow$ total runtime $\tilde{O}(\beta^2)$

Step 2: Getting the $\{\log_g p_i\}$

- Sample r s.t. g^r factors in B.

  $g^r = \prod p_i^{e_i}$

  $\Rightarrow r = \sum e_i \log_g p_i$ ← linear eqn in $\log_g p_i$

  ↑ $\tilde{O}(\beta^2)$ time to find r.

- Repeat $O(|B|) = \tilde{O}(\beta)$ times to get a solvable linear system
- solve for all $\log_g p_i$

  → $O(\beta^3)$ time

Total runtime: $\tilde{O}(\beta^3) = L_q(1/2, 3)$

Best known attack:

$L_q(1/3, 2)$

for $q \approx 2^{2048}$, $\approx 2^{122}$ ← close to 128

⇒ source of thousand-bit moduli for $\mathbb{Z}_p^*$

RSA?

- Similar attacks [Lenstra '87][Lenstra '93]

Conclusion

We need better d-log groups!

Next time: elliptic curve groups!