CSE 207B: Applied Cryptography

Nadia Heninger

UCSD

Fall 2023 Lecture 9

Announcements

1. HW 4 is due next lecture!
2. HW 5 is available! It uses Sage! You should try to install Sage ASAP to make sure you can run the code.
3. HW 5 is considered *hard*, particularly if the number theory is new to you. So start early!

Last time:

- Modular exponentiation: Given a compute g^a mod p. Efficient to compute.
- Discrete log: Given g^a mod p compute a. Not efficient to compute.

This time:

- Diffie-Hellman key exchange
- Elementary discrete log algorithms

"We stand today on the brink of a revolution in cryptography."

— Diffie and Hellman, 1976

New Directions in Cryptography

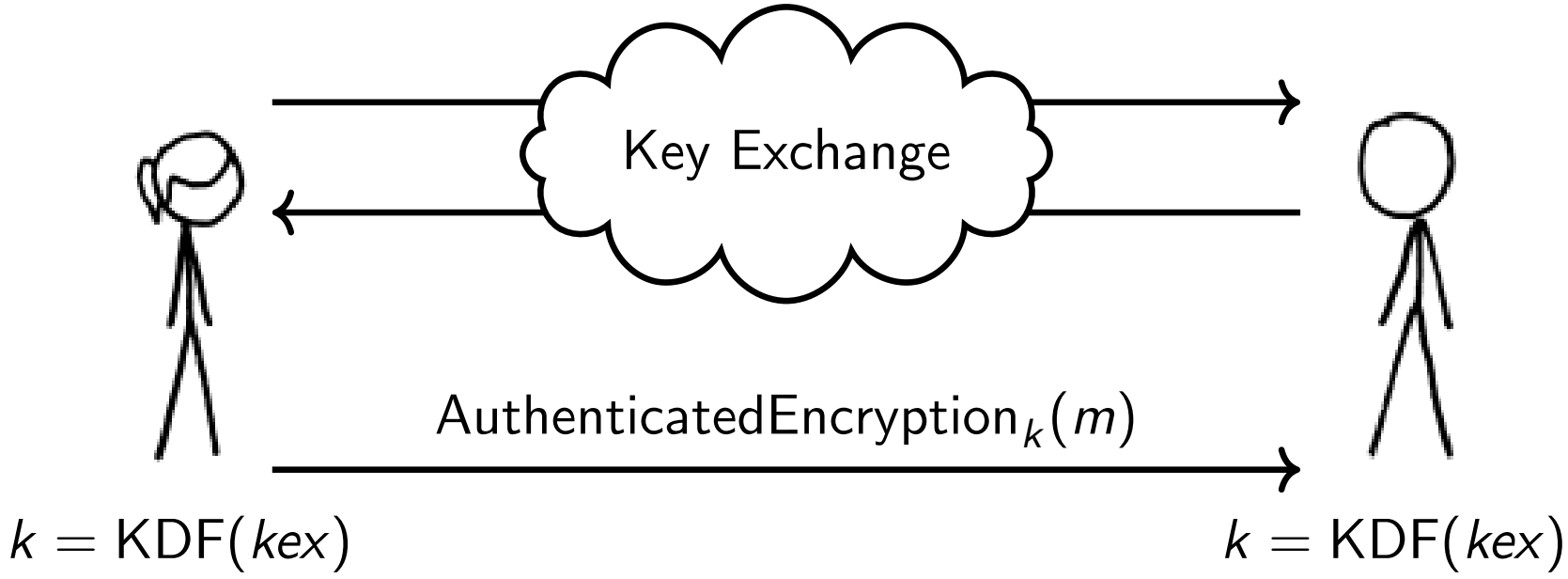
Invited Paper

WHITFIELD DIFFIE AND MARTIN E. HELLMAN, MEMBER, IEEE

The symmetric cryptography we just covered

Public key crypto idea # 1: Key exchange

Solving key distribution without trusted third parties



1. Alice, Bob exchange messages.
2. They derive a shared secret from the messages and use this to derive symmetric keys.
3. Passive eavesdropper can't recover shared secret from exchange messages.

Textbook Diffie-Hellman

[Diffie Hellman 1976]

Public Parameters

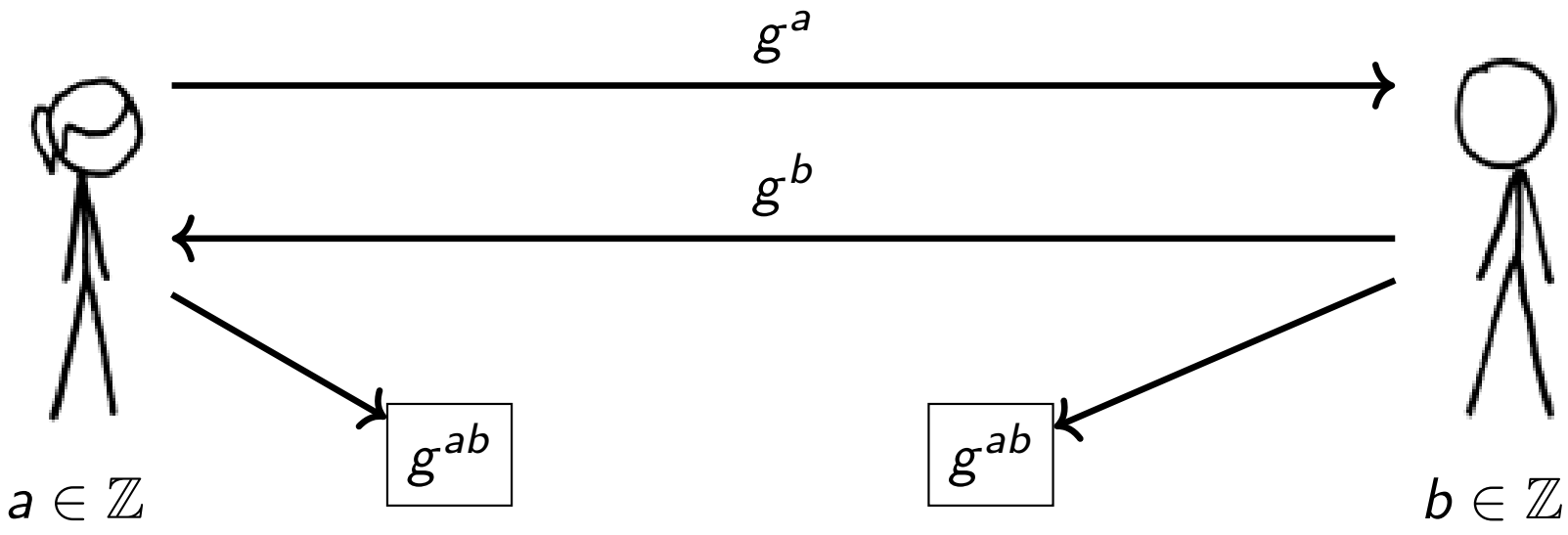
G a cyclic group

g group generator

Group desired properties:

- Efficient exponentiation
- Hard discrete log
- Commutativity in exponent

Key Exchange



Prime Field Diffie-Hellman

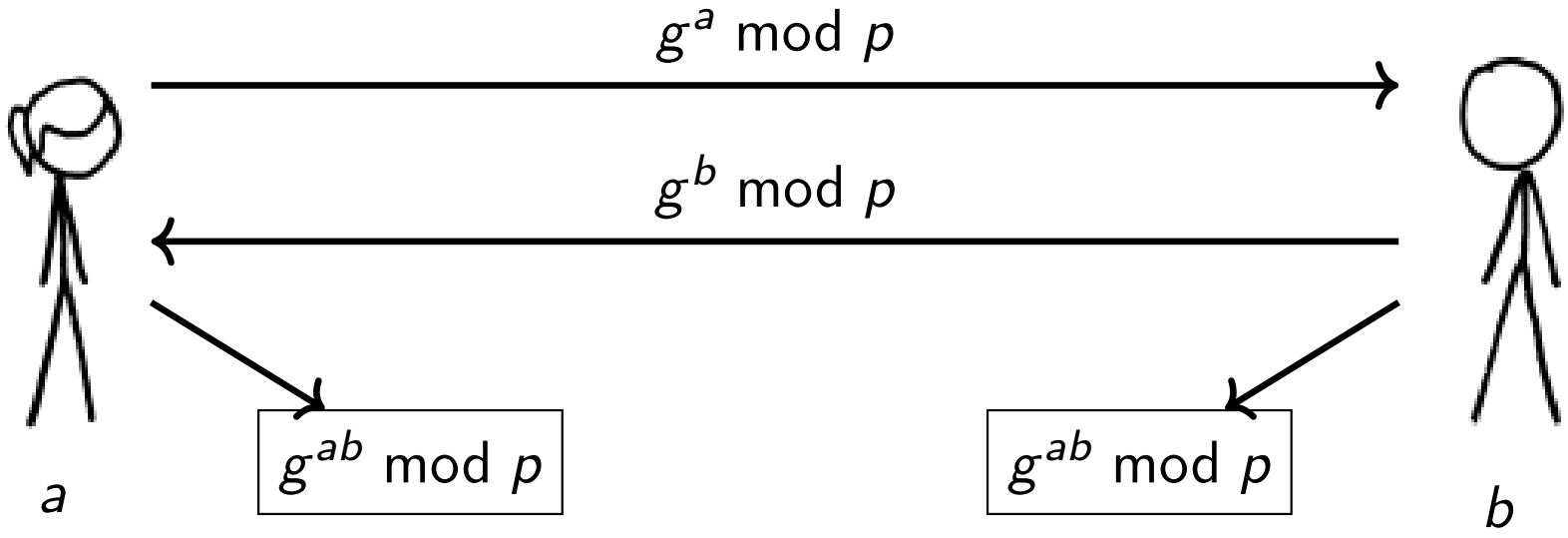
Public Parameters

p a prime

q a subgroup order; $q \mid (p-1)$

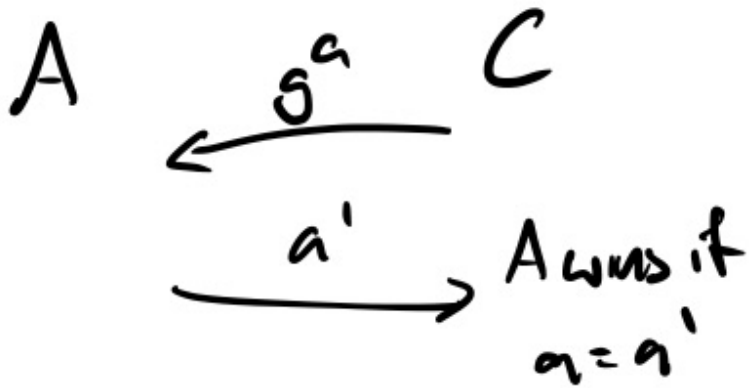
$g \in \mathbb{F}_p^*$ a generator of subgroup of order q

Key Exchange



The discrete log assumption

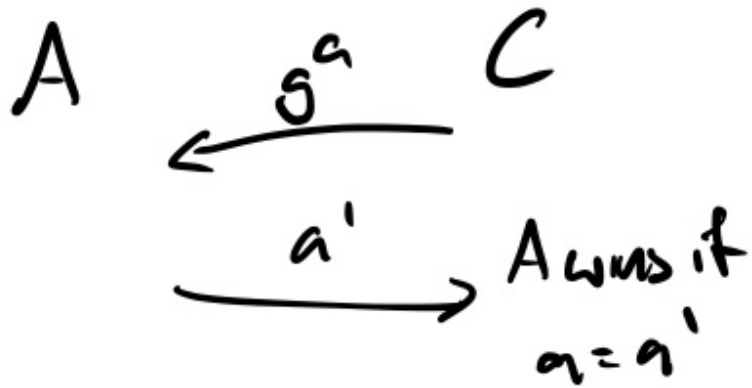
G a cyclic group with generator g.



Discrete log assumption:
Pr[A wins] is negligible.

The discrete log assumption

G a cyclic group with generator g.

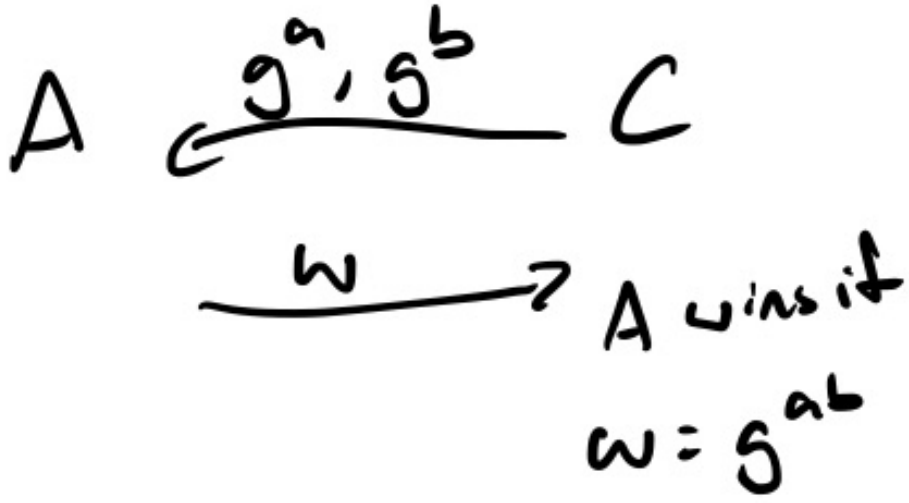


Discrete log assumption:
Pr[A wins] is negligible.

- Discrete log is easy $\implies$ Diffie-Hellman is easy to break.
  (Compute a and then compute public value.)
- But! Diffie-Hellman easy to break $\overset{?}{\not\implies}$ discrete log is easy.
- Discrete log is in NP and coNP $\implies$ not NP-complete
- Shor's algorithm solves discrete log with a quantum computer in polynomial time.

Computational Diffie-Hellman Assumption

G a cyclic group with generator g.

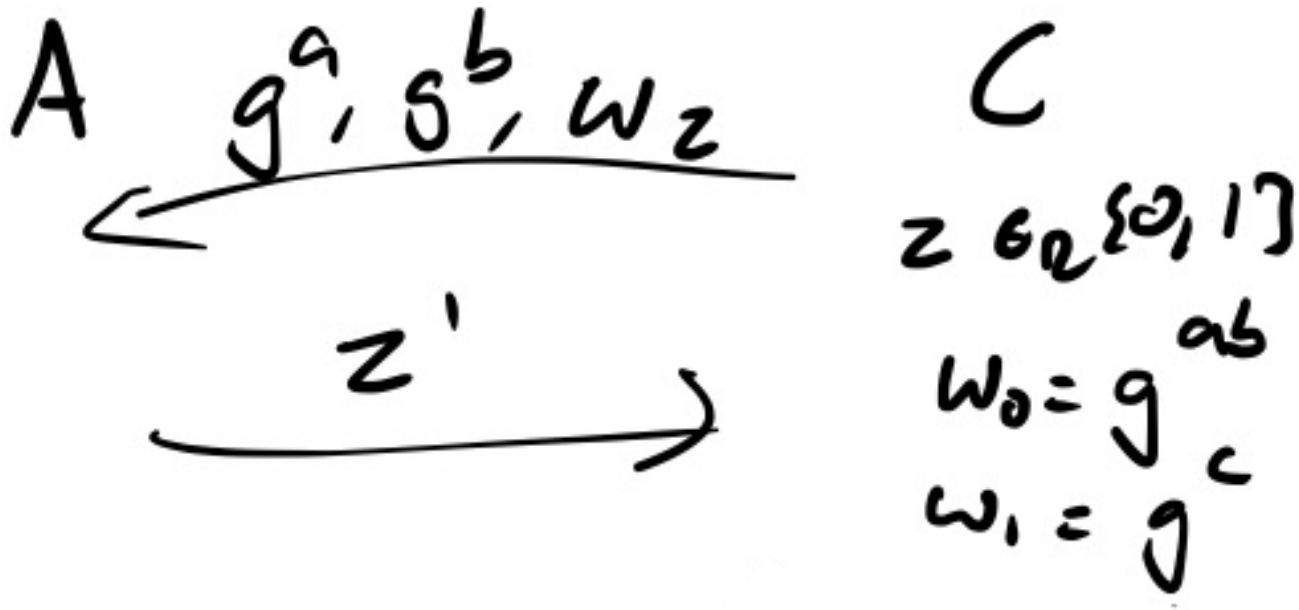


CDH assumption:
Pr[A wins] is negligible.

- Equivalent to computing the shared Diffie-Hellman secret.
- Discrete log is easy $\implies$ CDH is false.
- Not known if equivalent to computing discrete log
- It is hard to even *verify* a correct solution (g^a, g^b, g^{ab}).

Decisional Diffie-Hellman Assumption

G a cyclic group with generator g.



DDH assumption:
$|\Pr[A = 1 \mid z = 0] - \Pr[A = 1 \mid z = 1]|$ negligible.

- DDH hard $\implies$ difficult to distinguish Diffie-Hellman triples (g^a, g^b, g^{ab}) from random group elements (g^a, g^b, g^c).
- DDH hard $\implies$ CDH hard
- Assumption used for most security proofs.

Secure group choices for Diffie-Hellman

Why do we think discrete log is hard? Because there are groups for which there are no "good" algorithms known to compute them.

Group parameters are designed to avoid known cryptanalytic attacks.

Discrete log remains the best way to break Diffie-Hellman and related cryptosystems.

Types of groups used for Diffie-Hellman in practice now:

- "Prime-field Diffie-Hellman": Subgroups of $\mathbb{Z}_p^*$.
- "Elliptic curve Diffie-Hellman": Elliptic curve groups

Discrete log algorithms

Three families of discrete log algorithms:

1. Algorithms whose running time depends on the size of the order of the subgroup.
   - Good for computing discrete logs in subgroups of small or smooth order. "Smooth" = has many small prime factors
   - Includes generic group algorithms: Baby step giant step, Pollard rho.

Discrete log algorithms

Three families of discrete log algorithms:

1. Algorithms whose running time depends on the size of the order of the subgroup.
   - Good for computing discrete logs in subgroups of small or smooth order. "Smooth" = has many small prime factors
   - Includes generic group algorithms: Baby step giant step, Pollard rho.

2. Algorithms whose running time depends on the size of the exponent.
   - Good for computing discrete logs in a known small interval.
   - Pollard lambda algorithm.

Discrete log algorithms

Three families of discrete log algorithms:

1. Algorithms whose running time depends on the size of the order of the subgroup.
   - Good for computing discrete logs in subgroups of small or smooth order. "Smooth" = has many small prime factors
   - Includes generic group algorithms: Baby step giant step, Pollard rho.

2. Algorithms whose running time depends on the size of the exponent.
   - Good for computing discrete logs in a known small interval.
   - Pollard lambda algorithm.

3. Algorithms whose running time depends on the size of the modulus.
   - Good for computing discrete logs in subgroups of large order.
   - Number field sieve algorithm.

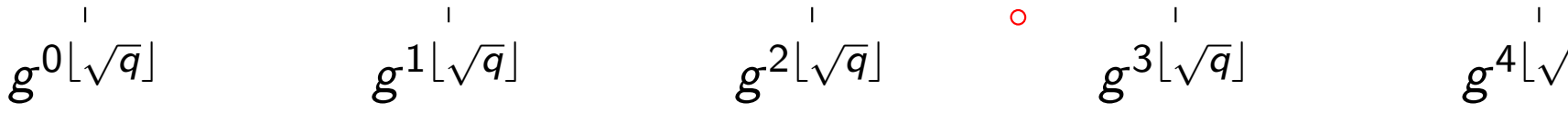
Computing Discrete Logs in $O(\sqrt{q})$ time

Input: Target t, prime p, group gen g, order q

Goal: Find ℓ where $g^\ell \equiv t \bmod p$.

Baby-Step Giant-Step Algorithm

1. "Giant steps": Compute $g^0, g^{\lfloor\sqrt{q}\rfloor}, g^{2\lfloor\sqrt{q}\rfloor}, \ldots g^{\lfloor\sqrt{q}\rfloor^2} \bmod p$.



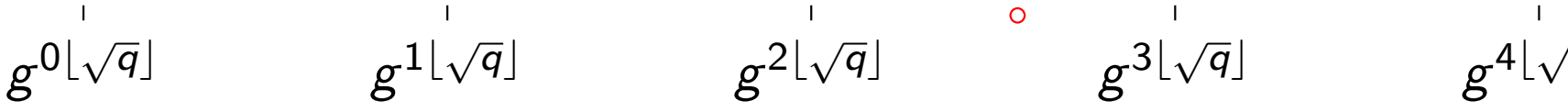
Computing Discrete Logs in $O(\sqrt{q})$ time

Input: Target t, prime p, group gen g, order q

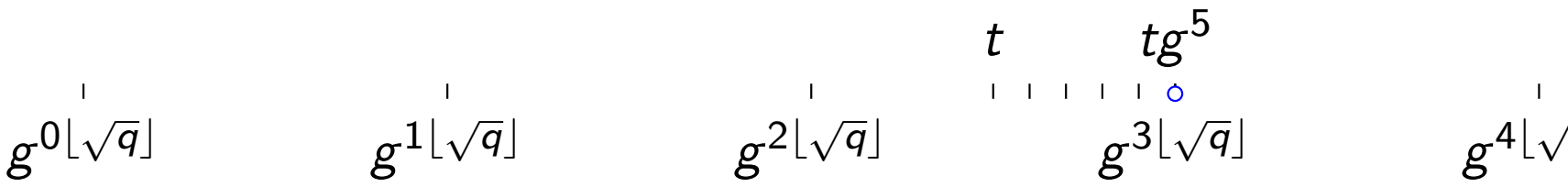
Goal: Find ℓ where $g^\ell \equiv t \bmod p$.

Baby-Step Giant-Step Algorithm

1. "Giant steps": Compute $g^0, g^{\lfloor\sqrt{q}\rfloor}, g^{2\lfloor\sqrt{q}\rfloor}, \ldots g^{\lfloor\sqrt{q}\rfloor^2} \bmod p$.



2. "Baby steps": Compute $tg^1, tg^2, \ldots \bmod p$ until collision.



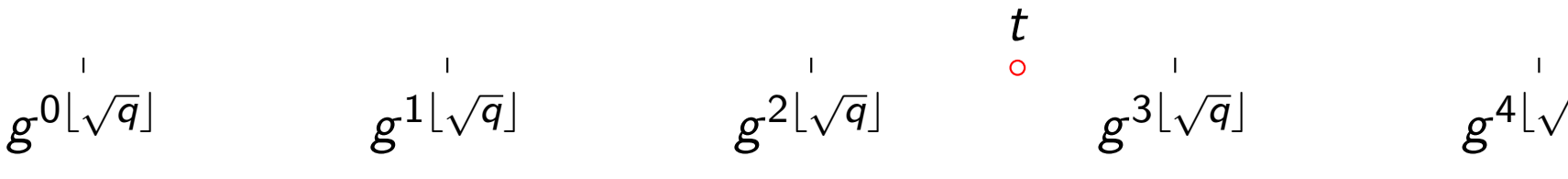
Computing Discrete Logs in $O(\sqrt{q})$ time

Input: Target t, prime p, group gen g, order q

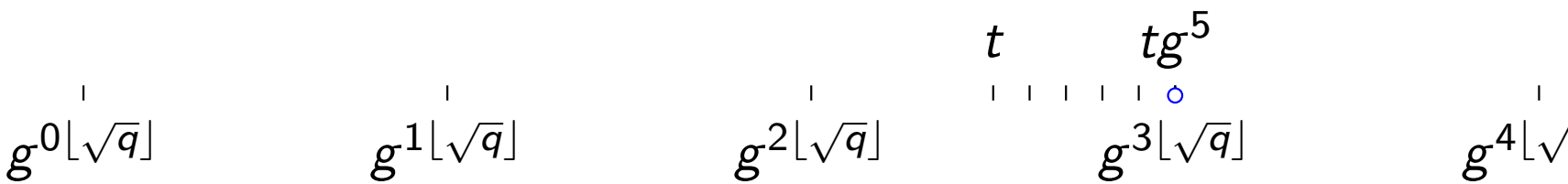
Goal: Find ℓ where $g^\ell \equiv t \bmod p$.

Baby-Step Giant-Step Algorithm

1. "Giant steps": Compute $g^0, g^{\lfloor\sqrt{q}\rfloor}, g^{2\lfloor\sqrt{q}\rfloor}, \ldots g^{\lfloor\sqrt{q}\rfloor^2} \bmod p$.



2. "Baby steps": Compute $tg^1, tg^2, \ldots \bmod p$ until collision.



3. Solve for $\log t$ from collision:

$$tg^j = g^{i\lfloor\sqrt{q}\rfloor} \bmod p$$

$$t = g^{i\lfloor\sqrt{q}\rfloor - j} \bmod p$$

$$\log t = i\lfloor\sqrt{q}\rfloor - j \bmod q$$

Baby Step Giant Step running time

- Storage: $O(\sqrt{q})$ group elements
- Running time: $O(\sqrt{q})$ group multiplications
- Running time: $O(\sqrt{q})$ table lookups

Total: $O(\sqrt{q}\,\mathrm{polylog}\,p)$

Baby Step Giant Step running time

- Storage: $O(\sqrt{q})$ group elements
- Running time: $O(\sqrt{q})$ group multiplications
- Running time: $O(\sqrt{q})$ table lookups

Total: $O(\sqrt{q}\,\mathrm{polylog}\,p)$

Idea: Reduce storage using random walk.

Pollard rho for discrete log

Input: Target t, prime p, group gen g, order q
Goal: Find ℓ where $g^\ell \equiv t \bmod p$.

- Take a random walk on elements $t^{a_i} g^{b_i}$.

Pollard rho for discrete log

Input: Target t, prime p, group gen g, order q

Goal: Find ℓ where $g^\ell \equiv t \bmod p$.

- Take a random walk on elements $t^{a_i} g^{b_i}$.
- A collision solves the problem :

$$t^{a_i} g^{b_i} = t^{a_j} g^{b_j} \bmod p$$

$$a_i \ell + b_i \equiv a_j \ell + b_j \bmod q$$

$$\ell \equiv (b_j - b_i)(a_i - a_j)^{-1} \bmod q$$

Pollard rho for discrete log

Input: Target t, prime p, group gen g, order q
Goal: Find ℓ where $g^\ell \equiv t \bmod p$.

- Take a random walk on elements $t^{a_i} g^{b_i}$.

- A collision solves the problem
- Pollard suggests this random walk:
  1. Start at arbitrary $a_0, b_0, x_0 = t^{a_0} g^{b_0}$
  2. $x_i = t^{a_i} g^{b_i}$

$$x_{i+1} = \begin{cases} tx_i & 1 \le x_i < p/3 \\ x_i^2 & p/3 \le x_i < 2p/3 \\ gx_i & 2p/3 \le x_i < p \end{cases} \qquad (a_{i+1}, b_{i+1}) = \begin{cases} (a_i + 1, b_i) \\ (2a_i, 2b_i) \\ (a_i, b_i + 1) \end{cases}$$

Using more intervals works better. (Teske)

Pollard rho analysis

Use Floyd cycle-finding algorithm. Pick an arbitrary starting point and iterate.

- Running time: $O(\sqrt{q})$ steps until collision.
- Storage: $O(1)$ pointers storing elements of size $\log p$ and $\log q$.

Countermeasure

Make sure Diffie-Hellman group orders have length twice the desired security.

More math review: Algebraic rings

- A ring is a set closed under two operations: $+$, $\times$
- $+$ has identity, inverses, associative, commutative
- $\times$ has identity, associative, commutative, might not have inverses
- Distributive law for $+$, $\times$

Canonical examples to remember:

- $\mathbb{Z}$
- $\mathbb{Z}_N$, the integers modulo N

From last time: $\mathbb{Z}_N^* = \{z \in \mathbb{Z}_N \text{ s.t. } \gcd(z, N) = 1\}$
This is an abelian group under $\times$.

Chinese remainder theorem

Theorem

$N = pq$ p, q relatively prime, not necessarily prime.

$$\mathbb{Z}_N \simeq \mathbb{Z}_p \times \mathbb{Z}_q \qquad \mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^*$$

$f : x \to (x \bmod p, x \bmod q)$ is an explicit isomorphism.

Theorem (Alternative Statement)

Given r_1, r_2 there is a unique $x \in \mathbb{Z}_N$ satisfying $x \equiv r_1 \bmod p$, $x \equiv r_2 \bmod q$.

Chinese Remainder Theorem, constructively

Want to solve for x s.t. $x \equiv r_p \bmod p \quad x \equiv r_q \bmod q$.

$$\gcd(p, q) = 1 \quad \Longrightarrow \quad ap + bq = 1$$

$$bq \equiv 1 \bmod p \quad bq \equiv 0 \bmod q$$
$$ap \equiv 0 \bmod p \quad ap \equiv 1 \bmod q$$

Think of these as "basis vectors".

Then to solve $x \equiv r_p \bmod p \quad x \equiv r_q \bmod q$, multiply the "magnitudes" by the "basis vectors":

$$x = (1, 0) \cdot r_p + (0, 1) \cdot r_q = bq \cdot r_p + ap \cdot r_q$$

Chinese Remainder Theorem in more "dimensions"

Want to solve for x s.t. $x \equiv r_1 \bmod p_1 \quad x \equiv r_k \bmod p_k$.

Let $N = \prod_i p_i$ and then set $N_i = N/p_i$.

$$\gcd(p_i, N_i) = 1 \quad \Longrightarrow \quad a_i p_i + b_i N_i = 1$$

Our "basis vectors":

$$b_i N_i \equiv 1 \bmod p_i \quad b_i N_i \equiv 0 \bmod p_j j \neq i$$

Then to solve $x \equiv r_i \bmod p_i$,

$$x = \sum_i r_i b_i N_i$$

Pohlig-Hellman Algorithm for Discrete Logs

How to compute discrete logs when the group order is composite

Want to solve $g^x = y \bmod p$. Know $\text{ord}(g) | p - 1$.

Say $\text{ord}(g) = m = p_1 p_2 \dots p_r$ so is composite.

Pohlig-Hellman Algorithm for Discrete Logs

How to compute discrete logs when the group order is composite

Want to solve $g^x = y \bmod p$. Know $\text{ord}(g) | p - 1$.

Say $\text{ord}(g) = m = p_1 p_2 \dots p_r$ so is composite.

Algorithm:

1. Solve the discrete log in each prime-order subgroup.
2. Use the Chinese Remainder Theorem to reconstruct the discrete log for the whole group.

Solving the discrete log in a prime-order subgroup

Want to solve $g^x = y \bmod p$ where
$\mathrm{ord}(g) = m = p_1 p_2 \ldots p_r \mid p - 1$.

Solving the discrete log in a prime-order subgroup

Want to solve $g^x = y \bmod p$ where
$\text{ord}(g) = m = p_1 p_2 \dots p_r \mid p - 1$.

By Lagrange's theorem, $g^m = 1 \bmod p$.

So g^{m/p_i} generates group of order $p_i \bmod p$.

We know:

$$\begin{aligned} g^x &\equiv y \bmod p \\ (g^x)^{m/p_i} &\equiv y^{m/p_i} \bmod p \\ \left(g^{m/p_i}\right)^x &\equiv y^{m/p_i} \bmod p \end{aligned}$$

Solving the discrete log in a prime-order subgroup

Want to solve $g^x = y \bmod p$ where
$\text{ord}(g) = m = p_1 p_2 \dots p_r \mid p - 1$.

By Lagrange's theorem, $g^m = 1 \bmod p$.

So g^{m/p_i} generates group of order $p_i \bmod p$.

We know:

$$\begin{aligned} g^x &\equiv y \bmod p \\ (g^x)^{m/p_i} &\equiv y^{m/p_i} \bmod p \\ \left(g^{m/p_i}\right)^x &\equiv y^{m/p_i} \bmod p \end{aligned}$$

Solving this discrete log problem, we learn a relation

$$x \equiv x_i \bmod p_i$$

Applying the Chinese Remainder Theorem

Solve discrete logs as above, we get many equivalences:

$$
\begin{aligned}
x &\equiv x_1 \bmod p_1 \\
x &\equiv x_2 \bmod p_2 \\
&\vdots \\
x &\equiv x_r \bmod p_r
\end{aligned}
$$

Use CRT to construct $x \bmod \prod_i p_i$.

Total algorithm time dominated by discrete log:
$O(\text{polylog}(p) \cdot max_i \sqrt{p_i})$

Countermeasures

Always do Diffie-Hellman and other discrete log-based cryptography over large prime order subgroups.

Common choices for p:

- "Safe" primes $p = 2q + 1$, with q prime, choose g so it generates group of order q.
- Some implementations choose q much smaller (e.g. 256 bits) and construct $p = qh + 1$ and choose g so it generates group of order q.

Next time

Formalizing key exchange and public-key encryption.