Course Business

- **Homework 3 Due Now**
- **Homework 4 Released**
- Professor Blocki is travelling, but will be back next week

Cryptography
CS 555

Week 11:

- Discrete Log/DDH
- Applications of DDH
- Factoring Algorithms, Discrete Log Attacks + NIST Recommendations for Concrete Security Parameters

Readings: Katz and Lindell Chapter 8.4 & Chapter 9

Recap: Cyclic Group

- $\mathbb{G} = \langle g \rangle = \{g^0, g^1, g^2, \ldots\}$ (g is generator)
- If $m = |\mathbb{G}|$ then for each $\mathrm{h} \in \mathbb{G}$ and each integer $x \geq 0$ we have

$$h^x = h^{x \bmod m}$$

Fact 1: Let p be a prime then $\mathbb{Z}_p^*$ is a cyclic group of order p-1.

Fact 2: Number of generators g s.t. of $\langle g \rangle = \mathbb{Z}_p^*$ is $\frac{|\phi(p-1)|}{p-1}$

Example (generator): p=7, g=5

<2>={1,5,4,6,2,3}

Recap: Cyclic Group

- $\mathbb{G} = \langle g \rangle = \{g^0, g^1, g^2, \ldots\}$ (g is generator)
- If $m = |\mathbb{G}|$ then for each $\mathrm{h} \in \mathbb{G}$ and each integer $x \geq 0$ we have
$$h^x = h^{x \, mod \, m}$$

Fact 1: Let p be a prime then $\mathbb{Z}_p^*$ is a cyclic group of order p-1.

Fact 2: Number of generators g s.t. of $\langle g \rangle = \mathbb{Z}_p^*$ is $\frac{|\phi(p-1)|}{p-1}$

Proof: Suppose that $\langle g \rangle = \mathbb{Z}_p^*$ and let $\mathrm{h} = g^i$ then

$\langle h \rangle = \left\{g^0, g^i, g^{2i \, mod \, (p-1)}, g^{3i \, mod \, (p-1)}, \ldots\right\}$

Recall: $\{ij \, mod \, (p-1) : j \geq 0\} = \{0, \ldots, p-1\}$ if and only if gcd(i,p-1)=1.

Recap Diffie-Hellman Problems

Computational Diffie-Hellman Problem (CDH)

- Attacker is given $\mathrm{h}_1 = g^{x_1} \in \mathbb{G}$ and $\mathrm{h}_2 = g^{x_2} \in \mathbb{G}$.
- Attackers goal is to find $g^{x_1 x_2} = (\mathrm{h}_1)^{x_2} = (\mathrm{h}_2)^{x_1}$
- **CDH Assumption**: For all PPT A there is a negligible function negl such that A succeeds with probability at most negl(n).

Decisional Diffie-Hellman Problem (DDH)

- Let $\mathrm{z}_0 = g^{x_1 x_2}$ and let $\mathrm{z}_1 = g^{r}$, where x_1,x_2 and r are random
- Attacker is given g^{x_1}, g^{x_2} and z_b (for a random bit b)
- Attackers goal is to guess b
- **DDH Assumption**: For all PPT A there is a negligible function negl such that A succeeds with probability at most ½ + negl(n).

Can we find a cyclic group where DDH holds?

- **Example 1:** $\mathbb{Z}_p^*$ where p is a random n-bit prime.
  - CDH is believed to be hard
  - DDH is *not* hard (You will prove this in homework 4 ☺)

- **Theorem:** Let p=rq+1 be a random n-bit prime where q is a large λ-bit prime then the set of r^th^ residues modulo p is a cyclic subgroup of order q. Then $\mathbb{G}_r = \left\{[h^r mod\ p] \middle| h \in \mathbb{Z}_p^*\right\}$ is a cyclic subgroup of $\mathbb{Z}_p^*$ of order q.
  - **Remark 1:** DDH is believed to hold for such a group
  - **Remark 2:** It is easy to generate uniformly random elements of $\mathbb{G}_r$
  - **Remark 3:** Any element (besides 1) is a generator of $\mathbb{G}_r$

Can we find a cyclic group where DDH holds?

- **Theorem:** Let p=rq+1 be a random n-bit prime where q is a large λ-bit prime then the set of rth residues modulo p is a cyclic subgroup of order q. Then $\mathbb{G}_r = \{[h^r mod\ p] \mid h \in \mathbb{Z}_p^*\}$ is a cyclic subgroup of $\mathbb{Z}_p^*$ of order q.
  - **Closure**: $h^r g^r = (hg)^r$
  - **Inverse** of h^r is $(h^{-1})^r \in \mathbb{G}_r$
  - **Size** $(h^r)^x = h^{[rx\ mod\ rq]} = (h^r)^x = h^{r[x\ mod\ q]} = (h^r)^{[x\ mod\ q]} mod\ p$

Remark: Two known attacks on Discrete Log Problem for $\mathbb{G}_r$(Section 9.2).

- First runs in time $O(\sqrt{q}) = O(2^{\lambda/2})$
- Second runs in time $2^{O(\sqrt[3]{n}(\log n)^{2/3})}$

Can we find a cyclic group where DDH holds?

Remark: Two known attacks (Section 9.2).

- First runs in time $O\left(\sqrt{q}\right) = O\left(2^{\lambda/2}\right)$
- Second runs in time $2^{O\left(\sqrt[3]{n}(\log n)^{2/3}\right)}$, where n is bit length of p

Goal: Set λ and n to balance attacks

$$\lambda = O\left(\sqrt[3]{n}(\log n)^{2/3}\right)$$

How to sample p=rq+1?

- First sample a random λ-bit prime q and
- Repeatedly check if rq+1 is prime for a random n- λ bit value r

More groups where DDH holds?

Elliptic Curves Example: Let p be a prime (p > 3) and let A, B be constants. Consider the equation

$$y^2 = x^3 + Ax + B \; mod \; p$$

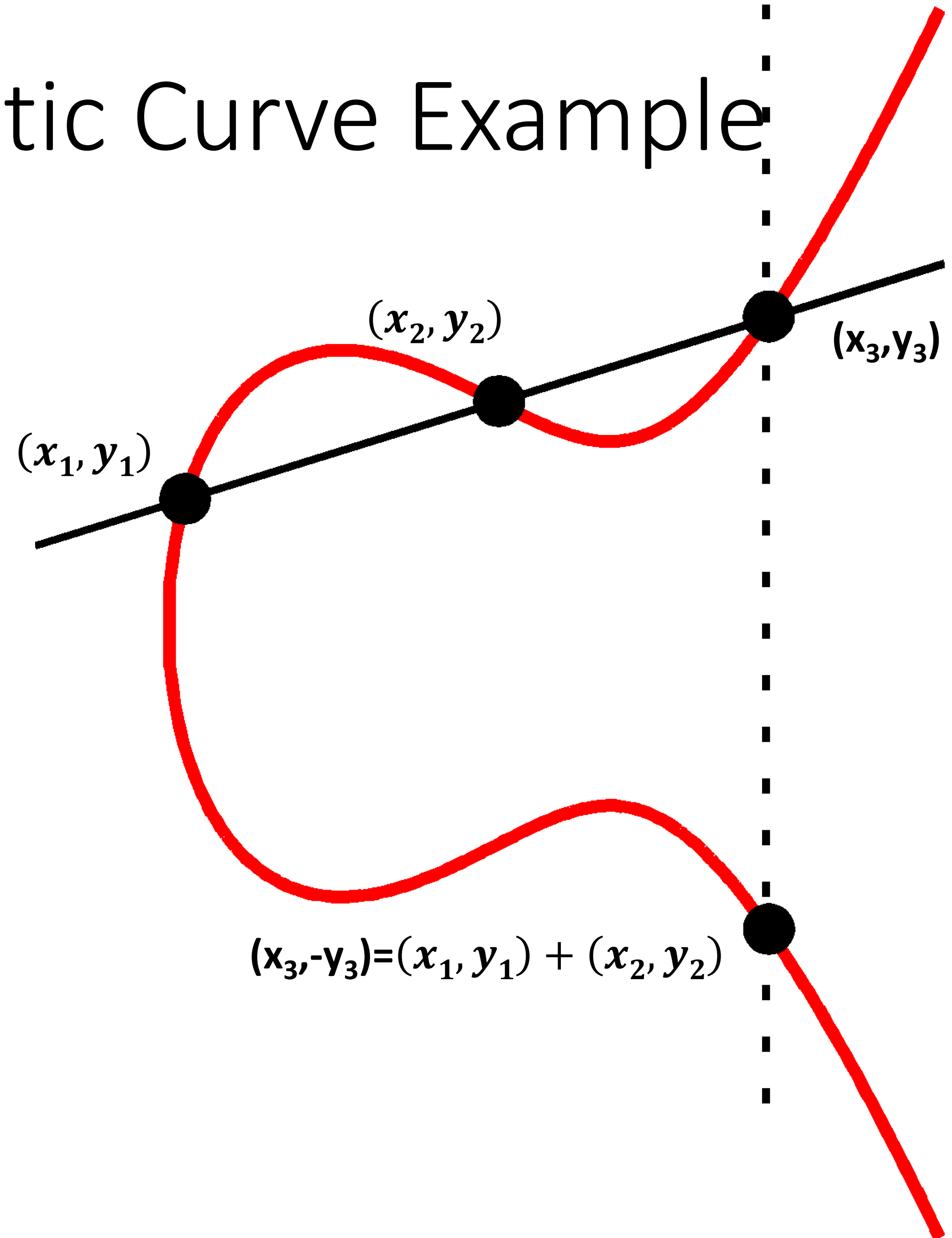
And let

$$E\left(\mathbb{Z}_p\right) = \left\{(x, y) \in \mathbb{Z}_p^2 \middle| y^2 = x^3 + Ax + B \; mod \; p \right\} \cup \{\mathcal{O}\}$$

Note: $\mathcal{O}$ is defined to be an additive identity $(x, y) + \mathcal{O} = (x, y)$

What is $(x_1, y_1) + (x_2, y_2)$?

Elliptic Curve Example



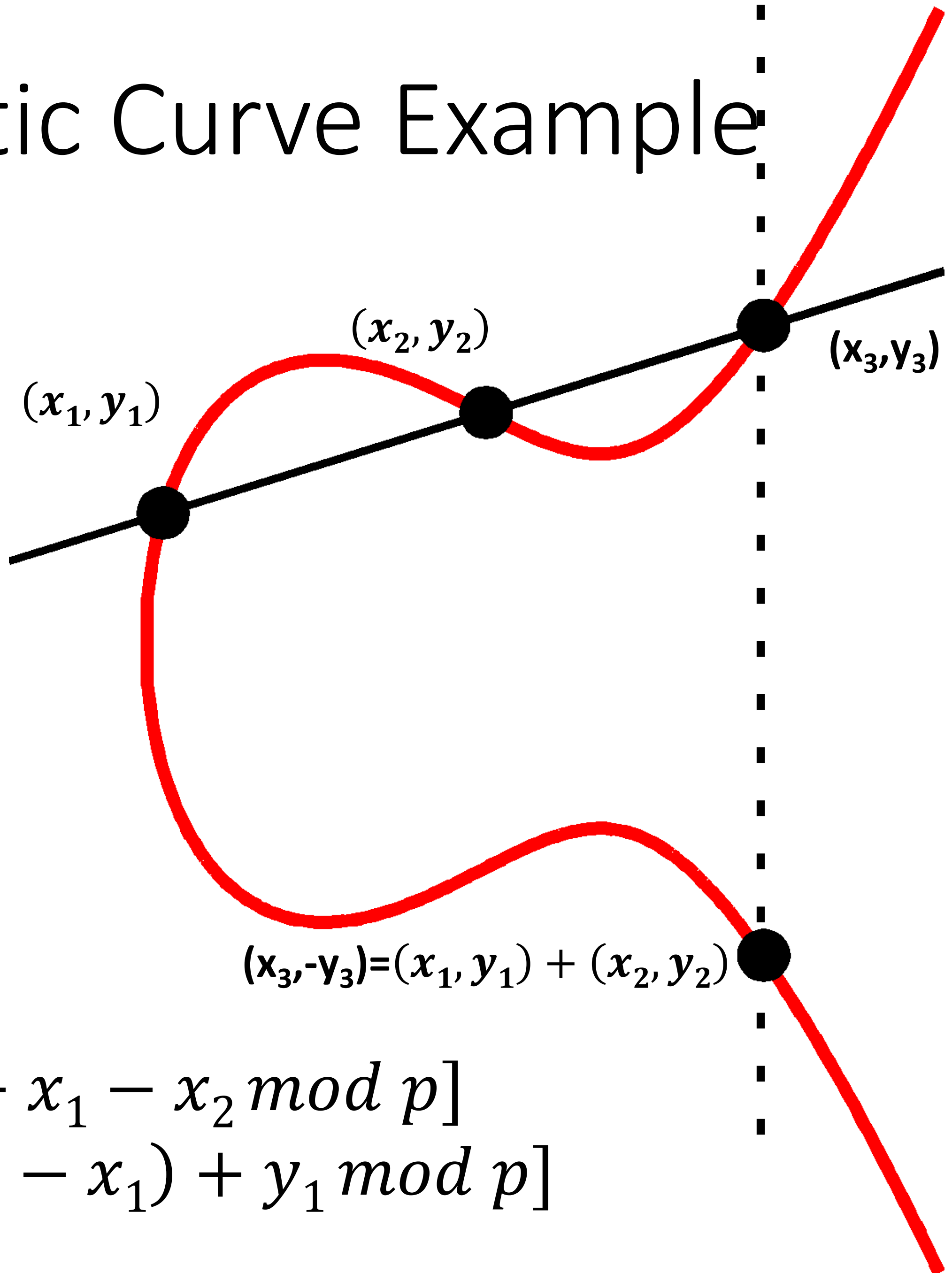
The line passing through $(\boldsymbol{x_1}, \boldsymbol{y_1})$ and $(\boldsymbol{x_2}, \boldsymbol{y_2})$ has the equation

$$y = m(x - x_1) + y_1 \, mod \, P$$

Where the slope

$$m = \left[\frac{y_1 - y_2}{x_1 - x_2} \, mod \, p\right]$$

Elliptic Curve Example



Formally, let

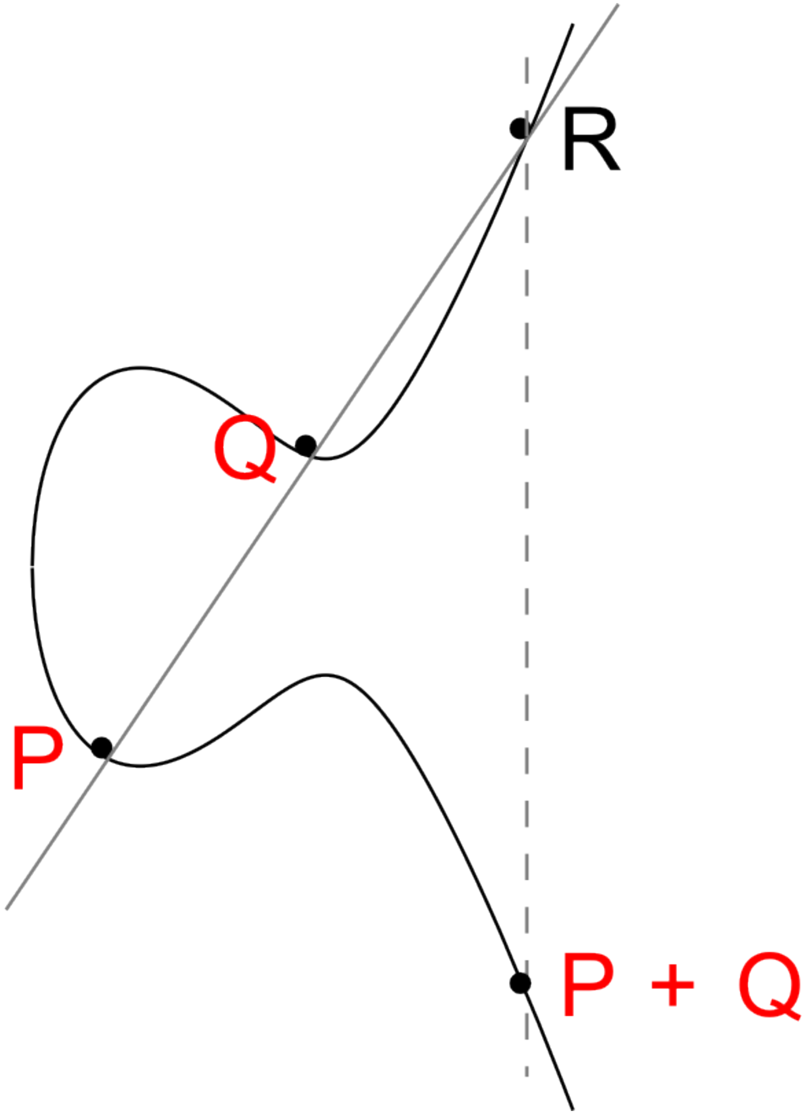
$$m = \left[\frac{y_1 - y_2}{x_1 - x_2} \; mod \; p\right]$$

Be the slope. Then the line passing through $(\boldsymbol{x_1}, \boldsymbol{y_1})$ and $(\boldsymbol{x_2}, \boldsymbol{y_2})$ has the equation

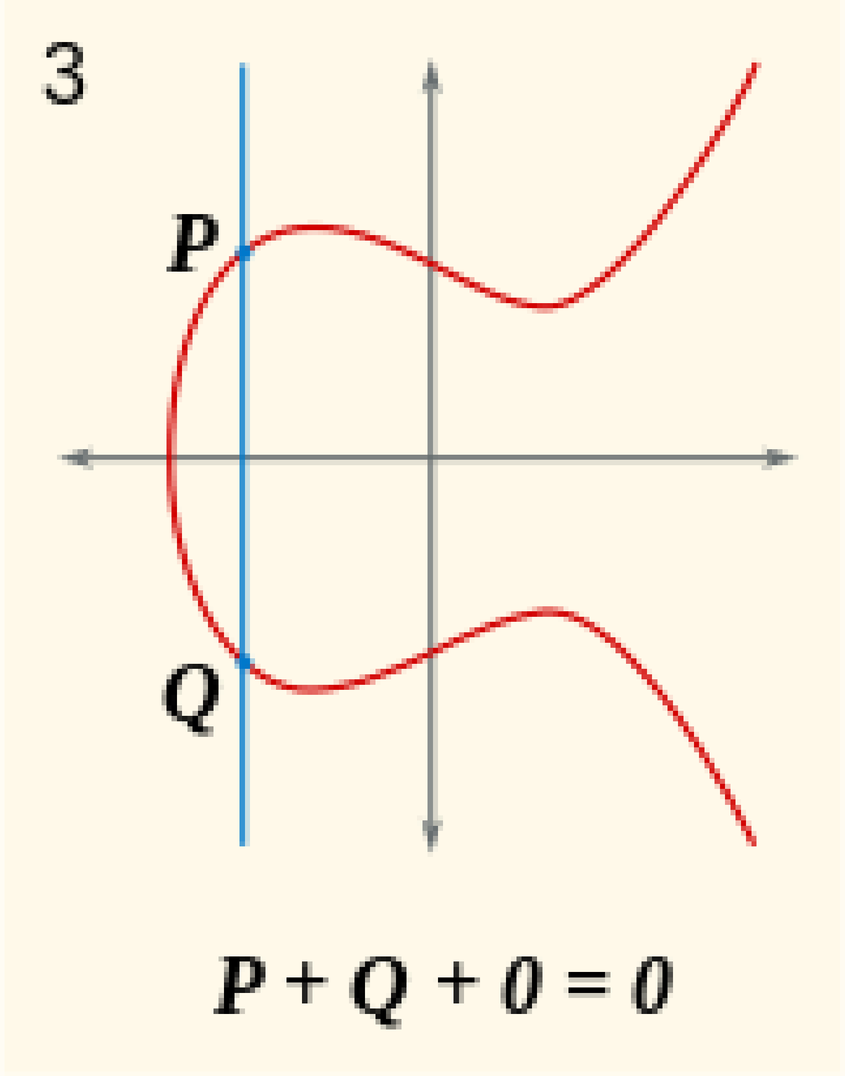
$$y = m(x - x_1) + y_1 \; mod \; P$$

$$(m(x - x_1) + y_1)^2 = x^3 + Ax + B \; mod \; p$$

$$x_3 = [m^2 - x_1 - x_2 \; mod \; p]$$
$$y_3 = [m(x_3 - x_1) + y_1 \; mod \; p]$$


R
Q
P
P + Q

Elliptic Curve Example



- No third point R on the line intersects our elliptic curve.
- Thus,
  $$P + Q = \mathcal{O}$$

Summary: Elliptic Curves

Elliptic Curves Example: Let p be a prime (p > 3) and let A, B be constants. Consider the equation

$$y^2 = x^3 + Ax + B \ mod \ p$$

And let

$$E\left(\mathbb{Z}_p\right) = \left\{(x, y) \in \mathbb{Z}_p^2 \middle| y^2 = x^3 + Ax + B \ mod \ p \ \right\} \cup \{\mathcal{O}\}$$

Fact: $E\left(\mathbb{Z}_p\right)$ defines an abelian group

- For *appropriate curves* the DDH assumption is believed to hold
- If you make up your own curve there is a good chance it is broken…
- NIST has a list of recommendations

Week 11: Topic 1: Discrete Logarithm Applications

Diffie-Hellman Key Exchange

Collision Resistant Hash Functions

Password Authenticated Key Exchange

Diffie-Hellman Key Exchange

1. Alice picks x_A and sends g^{x_A} to Bob
2. Bob picks x_B and sends g^{x_B} to Alice
3. Alice and Bob can both compute $K_{A,B} = g^{x_B\,x_A}$

Key-Exchange Experiment $KE^{eav}_{A,\Pi}(n)$:

- Two parties run Π to exchange secret messages (with security parameter 1^n).
- Let **trans** be a transcript which contains all messages sent and let k be the secret key output by each party.
- Let b be a random bit and let $\mathbf{k_b}$ = k if b=0; otherwise $\mathbf{k_b}$ is sampled uniformly at random.
- Attacker A is given **trans** and $\mathbf{k_b}$ (passive attacker).
- Attacker outputs b' ($KE^{eav}_{A,\Pi}(n)$=1 if and only if b=b')

Security of Π against an eavesdropping attacker: For all PPT A there is a negligible function **negl** such that

$$\Pr\left[KE^{eav}_{A,\Pi}(n)\right] = ½ + \mathbf{negl}(\mathrm{n}).$$

Diffie-Hellman Key-Exchange is Secure

Theorem: If the decisional Diffie-Hellman problem is hard relative to group generator $\mathcal{G}$ then the Diffie-Hellman key-exchange protocol Π is secure in the presence of a (passive) eavesdropper (*).

(*) Assuming keys are chosen uniformly at random from the cyclic group $\mathbb{G}$

Protocol Π

1. Alice picks x_A and sends g^{x_A} to Bob
2. Bob picks x_B and sends g^{x_B} to Alice
3. Alice and Bob can both compute $K_{A,B} = g^{x_B x_A}$

Diffie-Hellman Assumptions

Computational Diffie-Hellman Problem (CDH)

- Attacker is given $\mathrm{h}_1 = g^{x_1} \in \mathbb{G}$ and $\mathrm{h}_2 = g^{x_2} \in \mathbb{G}$.
- Attackers goal is to find $g^{x_1 x_2} = (\mathrm{h}_1)^{x_2} = (\mathrm{h}_2)^{x_1}$
- **CDH Assumption**: For all PPT A there is a negligible function negl upper bounding the probability that A succeeds

Decisional Diffie-Hellman Problem (DDH)

- Let $\mathrm{z}_0 = g^{x_1 x_2}$ and let $\mathrm{z}_1 = g^{r}$, where x_1,x_2 and r are random
- Attacker is given g^{x_1}, g^{x_2} and z_b (for a random bit b)
- Attackers goal is to guess b
- **DDH Assumption**: For all PPT A there is a negligible function negl such that A succeeds with probability at most ½ + negl(n).

Diffie-Hellman Key Exchange

1. Alice picks x_A and sends g^{x_A} to Bob
2. Bob picks x_B and sends g^{x_B} to Alice
3. Alice and Bob can both compute $K_{A,B} = g^{x_B x_A}$

Intuition: Decisional Diffie-Hellman assumption implies that a passive attacker who observes g^{x_A} and g^{x_B} still cannot distinguish between $K_{A,B} = g^{x_B x_A}$ and a random group element**.**

Remark: Modified protocol sets $K_{A,B} = H(g^{x_B x_A})$**.** You will prove that this protocol is secure under the weaker CDH assumption in homework 4.

Diffie-Hellman Key-Exchange is Secure

Theorem: If the decisional Diffie-Hellman problem is hard relative to group generator $\mathcal{G}$ then the Diffie-Hellman key-exchange protocol Π is secure in the presence of an eavesdropper (*).

Proof:

$$\Pr\left[KE_{A,\Pi}^{eav}(n) = 1\right]$$
$$= ½\Pr\left[KE_{A,\Pi}^{eav}(n) = 1 | b = 1\right] + ½\Pr\left[KE_{A,\Pi}^{eav}(n) = 1 | b = 0\right]$$
$$= ½\Pr[A(\mathbb{G}, g, q, g^x, g^y, g^{xy}) = 1] + ½\Pr[A(\mathbb{G}, g, q, g^x, g^y, g^z) = 1]$$
$$= ½ + ½(\Pr[A(\mathbb{G}, g, q, g^x, g^y, g^{xy}) = 1] - \Pr[A(\mathbb{G}, g, q, g^x, g^y, g^z) = 1]).$$
$$\leq ½ + ½\mathrm{negl}(n) \text{ (by DDH)}$$

(*) Assuming keys are chosen uniformly at random from the cyclic group $\mathbb{G}$

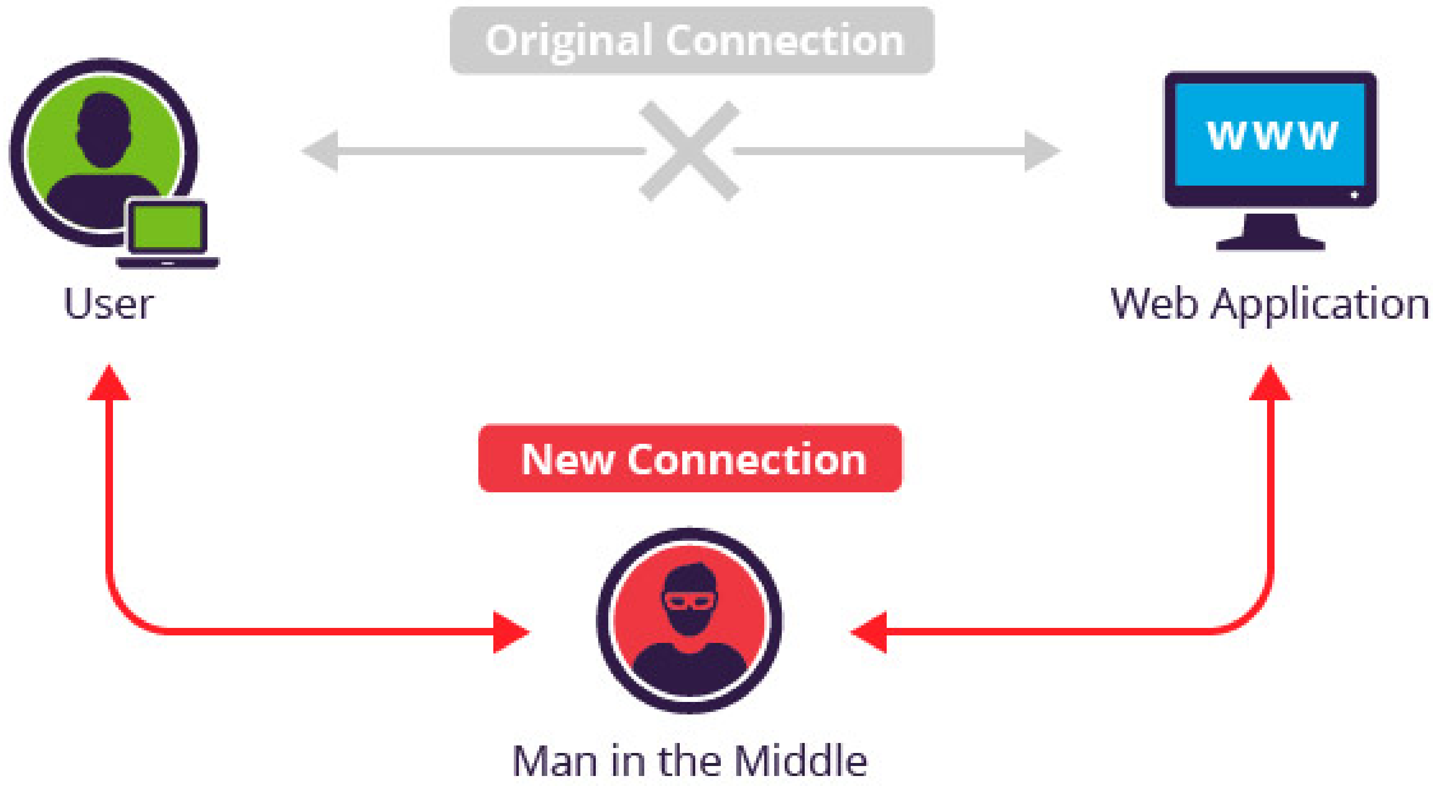
Diffie-Hellman Key Exchange

1. Alice picks x_A and sends g^{x_A} to Bob
2. Bob picks x_B and sends g^{x_B} to Alice
3. Alice and Bob can both compute $K_{A,B} = g^{x_B\, x_A}$

Intuition: Decisional Diffie-Hellman assumption implies that a passive attacker who observes g^{x_A} and g^{x_B} still cannot distinguish between $K_{A,B} = g^{x_B\, x_A}$ and a random group element**.**

Remark: The protocol is vulnerable against active attackers who can tamper with messages.

Man in the Middle Attack (MITM)



Man in the Middle Attack (MITM)

1. Alice picks x_A and sends g^{x_A} to Bob
   - Mallory intercepts g^{x_A} , picks x_E and sends g^{x_E} to Bob instead
2. Bob picks x_B and sends g^{x_B} to Alice
   1. Mallory intercepts g^{x_B}, picks $x_{E'}$ and sends $g^{x_{E'}}$ to Alice instead
3. Eve computes $g^{x_{E'} x_A}$ and $g^{x_E x_B}$
   1. Alice computes secret key $g^{x_{E'} x_A}$ (shared with Eve not Bob)
   2. Bob computes $g^{x_E x_B}$(shared with Eve not Alice)
4. Mallory forwards messages between Alice and Bob (tampering with the messages if desired)
5. Neither Alice nor Bob can detect the attack

Discrete Log Experiment $\text{DLog}_{A,G}(n)$

1. Run $\mathcal{G}(1^n)$ to obtain a cyclic group $\mathbb{G}$ of order q (with $\|q\| = n$) and a generator g such that $< \mathrm{g} >= \mathbb{G}$.
2. Select $\mathrm{h} \in \mathbb{G}$ uniformly at random.
3. Attacker A is given $\mathbb{G}$, q, g, h and outputs integer x.
4. Attacker wins ($\text{DLog}_{A,G}(n)=1$) if and only if $g^x=h$.

We say that the discrete log problem is hard relative to generator $\mathcal{G}$ if

$$\forall PPT\ A\ \exists \mu\ (\text{negligible})\ \text{s.t}\ \Pr[\text{DLog}_{A,n} = 1] \leq \mu(n)$$

Collision Resistant Hash Functions (CRHFs)

- Recall: not known how to build CRHFs from OWFs
- Can build collision resistant hash functions from Discrete Logarithm Assumption
- Let $\mathcal{G}(1^n)$ output $(\mathbb{G}, q, g)$ where $\mathbb{G}$ is a cyclic group of order q and g is a generator of the group.
- Suppose that discrete log problem is hard relative to generator $\mathcal{G}$.

$$\forall PPT\ A\ \exists \mu\ (\text{negligible})\ \text{s.t}\ \Pr[\mathsf{DLog}_{\mathsf{A,n}} = 1] \leq \mu(n)$$

Collision Resistant Hash Functions

- Let $\mathcal{G}(1^n)$ output $(\mathbb{G}, q, g)$ where $\mathbb{G}$ is a cyclic group of order q and g is a generator of the group.

Collision Resistant Hash Function (Gen,H):

- $Gen(1^n)$
  1. $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$
  2. Select random $\mathrm{h} \leftarrow \mathbb{G}$
  3. Output $\mathrm{s} = (\mathbb{G}, q, g, h)$
- $H^s(x_1, x_2) = g^{x_1} h^{x_2}$ (where, $x_1, x_2 \in \mathbb{Z}_q$)

Claim: (Gen,H) is collision resistant if the discrete log assumption holds for $\mathcal{G}$

Collision Resistant Hash Functions

- $H^s(x_1, x_2) = g^{x_1} h^{x_2}$ (where, $x_1, x_2 \in \mathbb{Z}_q$)

Claim: (Gen,H) is collision resistant

Proof: Suppose we find a collision $H^s(x_1, x_2) = H^s(y_1, y_2)$ then we have $g^{x_1} h^{x_2} = g^{y_1} h^{y_2}$ which implies

$$h^{x_2 - y_2} = g^{y_1 - x_1}$$

Use extended GCD to find $(x_2 - y_2)^{-1}$ mod q then

$$h = h^{(x_2 - y_2)(x_2 - y_2)^{-1} \, mod \, q} = g^{(y_1 - x_1)(x_2 - y_2)^{-1} \, mod \, q}$$

Which means that $(y_1 - x_1)(x_2 - y_2)^{-1} \, mod \, q$ is the discrete log of h.

Password Authenticated Key-Exchange

- Suppose Alice and Bob share a low-entropy password pwd and wish to communicate securely
  - (without using any trusted party)
  - Assuming an active attacker may try to mount a man-in-the-middle attack
- Can they do it?

Tempting Approach:

- Alice and Bob both compute K= KDF(pwd)=H^n(pwd) and communicate with using an authenticated encryption scheme.
- **Practice Midterm Exam:** Secure in random oracle model if attacker cannot query random oracle H(.) too many times.

Password Authenticated Key-Exchange

Tempting Approach:

- Alice and Bob both compute K= KDF(pwd)=H^n(pwd) and communicate with using an authenticated encryption scheme.
- **Midterm Exam:** Secure in random oracle model if attacker cannot query random oracle too many time.
- **Problems:**
  - In practice the attacker can (and will) query the random oracle many times.
  - In practice people tend to pick very weak passwords
  - Brute-force attack: Attacker enumerates over a dictionary of passwords and attempts to decrypt messages with $K_{pwd'}$=KDF(pwd') (only succeeds if $K_{pwd'}$=K).
  - An offline attack (brute-force) will almost always succeed

Attempt 2

1. Alice picks x_A and sends $g^{H(pwd)+xA}$ to Bob
2. Bob picks x_B and sends $g^{H(pwd)+x_B}$ to Alice
3. Alice and Bob can both compute $K_{A,B} = H(g^{x_B\, x_A})$
4. Alice picks random nonce r_A and sends $Enc_{K_{A,B}}(r_A)$ to Bob
    1. Enc is an authentication encryption scheme
5. Bob decrypts and sends r_A to Alice

Advantage: MITM Attacker cannot establish connection without password

Disadvantage: Mallory could mount a brute-force attack after attempted MITM attack

Attempt 2: MITM Attack

1. Alice picks x_A and sends $g^{H(pwd)+xA}$ to Bob
2. Bob picks x_B and sends $g^{H(pwd)+x_B}$ to Alice
    1. Mallory intercepts $g^{H(pwd)+xB}$, picks x_E and sends g^{x_E} to Alice instead
3. Bob can both compute $K_{A,B} = H(g^{x_B x_A})$
    1. Allice computes $K_{A,B}' = H\left(g^{(x_E - H(pwd))\, x_A}\right)$ instead
4. Alice picks random nonce r_A and sends c $= Enc_{K_{A,B}'}(r_A)$ to Bob
    1. Mallory intercepts $Enc_{K_{A,B}'}(r_A)$ and proceeds to mount brute-force attack on password
5. For each password guess y
    1. let $K_y = H\left(g^{(x_E - H(y))\, x_A}\right)$ and
    2. if $Dec_{K_{A,B}'}(c) \neq \perp$ then output y

Advantage: MITM Attacker cannot establish connection without password

Disadvantage: Mallory could mount a brute-force attack on password after attempted MITM attack

Password Authenticated Key-Exchange (PAKE)

Better Approach (PAKE):

1. Alice and Bob both compute $\mathrm{W} = g^{pwd}$
2. Alice picks x_A and sends "Alice", $X = g^{x_A}$ to Bob
3. Bob picks x_B computes $\mathrm{r} = \mathrm{H}(1, Alice, Bob, X)$ and $Y = (X \times (W)^r)^{x_B}$ and sends Alice the following message: "$Bob,$" Y
4. Alice computes $\mathrm{K} = Y^Z = g^{x_B}$ where $z = 1/\big((pwd \times r) + x_A\big)\ mod\ p$. Alice sends the message $\mathrm{V_A}$= H(2,Alice,Bob,X,Y,K) to Bob.
5. Bob verifies that $\mathrm{V_A}$== H(2,Alice,Bob,X,Y,K) where K = g^{x_B}. Bob generates $\mathrm{V_B}$= H(3,Alice,Bob,X,Y,K) and sends $\mathrm{V_B}$ to Alice.
6. Alice verifies that $\mathrm{V_B}$==H(3,Alice,Bob,X,Y, Y^Z) where $z = 1/\big((pwd \times r) + x_A\big)$.
7. If Alice and Bob don't terminate the session key is H(4,Alice,Bob,X,Y, K)

Security:

- No offline attack (brute-force) is possible. Attacker get's one password guess per instantiation of the protocol.
- If attacker is incorrect and he tampers with messages then he will cause the Alice & Bob to quit.
- If Alice and Bob accept the secret key K and the attacker did not know/guess the password then K is "just as good" as a truly random secret key.

See [RFC 6628]

Week 11: Topic 2: Factoring Algorithms, Discrete Log Attacks + NIST Recommendations for Concrete Security Parameters

Pollard's p-1 Algorithm (Factoring)

- Let $N = pq$ where (p-1) has only "small" prime factors.
- Pollard's p-1 algorithm can factor N.
  - **Remark 1**: This happens with very small probability if p is a random n bit prime.
  - **Remark 2**: One convenient/fast way to generate big primes it to multiply many small primes, add 1 and test for primality.
    - Example: $2 \times 3 \times 5 \times 7 + 1 = 211$ is prime

Claim: Suppose we are given an integer B such that (p-1) divides B but (q-1) does not divide B then we can factor N.

Pollard’s p-1 Algorithm (Factoring)

Claim: Suppose we are given an integer B such that (p-1) divides B but (q-1) does not divide B then we can factor N.

Proof: Suppose B=c(p-1) for some integer c and let

$$y = [x^B - 1\ mod\ N]$$

Applying the Chinese Remainder Theorem we have

$$y \leftrightarrow (x^B - 1 \text{ mod p}, x^B - 1 \text{ mod q})$$
$$= (0, x^B - 1 \text{ mod q})$$

This means that p divides y, but q does not divide y (unless $x^B = 1 \text{ mod q}$, which is very unlikely).

Thus, GCD(y,N) = p

Pollard's p-1 Algorithm (Factoring)

- Let $N = pq$ where (p-1) has only "small" prime factors.
- Pollard's p-1 algorithm can factor N.

Claim: Suppose we are given an integer B such that (p-1) divides B but (q-1) does not divide B then we can factor N.

- **Goal**: Find B such that (p-1) divides B but (q-1) does not divide B.
- **Remark**: This is difficult if (p-1) has a large prime factor.

$$B = \prod_{i=1}^{k} p_i^{[n/\log p_i]}$$

Pollard's p-1 Algorithm (Factoring)

- **Goal**: Find B such that (p-1) divides B but (q-1) does not divide B.
- **Remark**: This is difficult if (p-1) has a large prime factor.

$$B = \prod_{i=1}^{k} p_i^{[n/\log p_i]}$$

Here p_1=2,p_2=3,…

Fact: If (q-1) has prime factor larger than p_k then (q-1) does not divide B.

Fact: If (p-1) does not have prime factor larger than p_k then (p-1) does divide B.

Pollard's p-1 Algorithm (Factoring)

- **Option 1:** To defeat this attack we can choose strong primes p and q
  - A prime p is strong if (p-1) has a large prime factor
- **Drawback:** It takes more time to generate (provably) strong primes

- **Option 2:** A random prime is strong with high probability

- **Current Consensus:** Just pick a random prime

Pollard’s Rho Algorithm

- General Purpose Factoring Algorithm
  - Doesn’t assume (p-1) has no large prime factor
  - **Goal**: factor N=pq (product of two n-bit primes)
- Running time: $O\left(\sqrt[4]{N}\ \text{pol}ylog(N)\right)$
  - Naïve Algorithm takes time $O\left(\sqrt{N}\ \text{pol}ylog(N)\right)$ to factor

- **Core idea:** find distinct $\mathrm{x}, \mathrm{x}' \in \mathbb{Z}_N^*$ such that $x = x' \bmod p$
  - Implies that x-x’ is a multiple of p and, thus, GCD(x-x’,N)=p (whp)

Pollard's Rho Algorithm

- General Purpose Factoring Algorithm
  - Doesn't assume (p-1) has no large prime factor
- Running time: $O\left(\sqrt[4]{N}\ \ \mathrm{pol}ylog(N)\right)$
- **Core idea:** find distinct $\mathrm{x}, \mathrm{x}' \in \mathbb{Z}_N^*$ such that $x = x' \mathrm{mod}\, p$
  - Implies that x-x' is a multiple of p and, thus, GCD(x-x',N)=p (whp)
- **Question**: If we pick $\mathrm{k} = \mathrm{O}\left(\sqrt{p}\right)$ random $x^{(1)}, \dots, x^{(k)} \in \mathbb{Z}_N^*$ then what is the probability that we can find distinct i and j such that
$$x^{(i)} = x^{(j)} \mathrm{mod\ p?}$$

Pollard's Rho Algorithm

- **Question**: If we pick $\mathrm{k} = \mathrm{O}\left(\sqrt{p}\right)$ random $x^{(1)}, \ldots, x^{(k)} \in \mathbb{Z}_N^*$ then what is the probability that we can find distinct i and j such that $x^{(i)} = x^{(j)} \mathrm{mod\ p}$?
- **Answer:** $\geq {}^{1}/_{2}$
- **Proof (sketch):** Use the Chinese Remainder Theorem + Birthday Bound

$$x^{(i)} = \left(x^{(i)}\ mod\ p, x^{(i)}\ mod\ q\right)$$

Note: We will also have $x^{(i)} \neq x^{(j)}$ mod q (whp)

Pollard's Rho Algorithm

- **Question**: If we pick $\mathrm{k} = \mathrm{O}\left(\sqrt{p}\right)$ random $x^{(1)}, \dots, x^{(k)} \in \mathbb{Z}_N^*$ then what is the probability that we can find distinct i and j such that $x^{(i)} = x^{(j)} \mathrm{mod}\ \mathrm{p}$?
- **Answer:** $\geq {}^1\!/_2$
- **Challenge:** We do not know p or q so we cannot sort the $x^{(i)}$'s using the Chinese Remainder Theorem Representation

$$x^{(i)} = \left(x^{(i)}\ mod\ p, x^{(i)}\ mod\ q\right)$$

How can we identify the pair i and j such that $x^{(i)} = x^{(j)} \mathrm{mod}\ \mathrm{p}$?

Pollard's Rho Algorithm

- Pollard's Rho Algorithm is similar the low-space version of the birthday attack

Input: N (product of two n bit primes)

$x^{(0)} \leftarrow \mathbb{Z}_N^*, \mathrm{x} = \mathrm{x}' = x^{(0)}$

For i=1 to $2^{n/2}$

$x \leftarrow F(x)$

$x' \leftarrow F\big(F(x')\big)$

p = **GCD**(x-x',N)

if 1< p < N **return** p

Remark 1: F should have the property that if x=x' mod p then F(x) = F(x') mod p.

Remark 2: $F(x) = [x^2 + 1 \; mod \; N]$ will work since

$$F(x) = [x^2 + 1 \; mod \; N]$$
$$\leftrightarrow (x^2 + 1 \; mod \; p, x^2 + 1 \; mod \; q)$$
$$\leftrightarrow (F([x \bmod p]\,) \; mod \; p, F([x \bmod q]\,) mod \; q)$$

Pollard's Rho Algorithm

- Pollard's Rho Algorithm is similar the low-space version of the birthday attack

Input: N (product of two n bit primes)

$x^{(0)} \leftarrow \mathbb{Z}_N^*, \mathrm{x} = \mathrm{x}' = x^{(0)}$

For i=1 to $2^{n/2}$

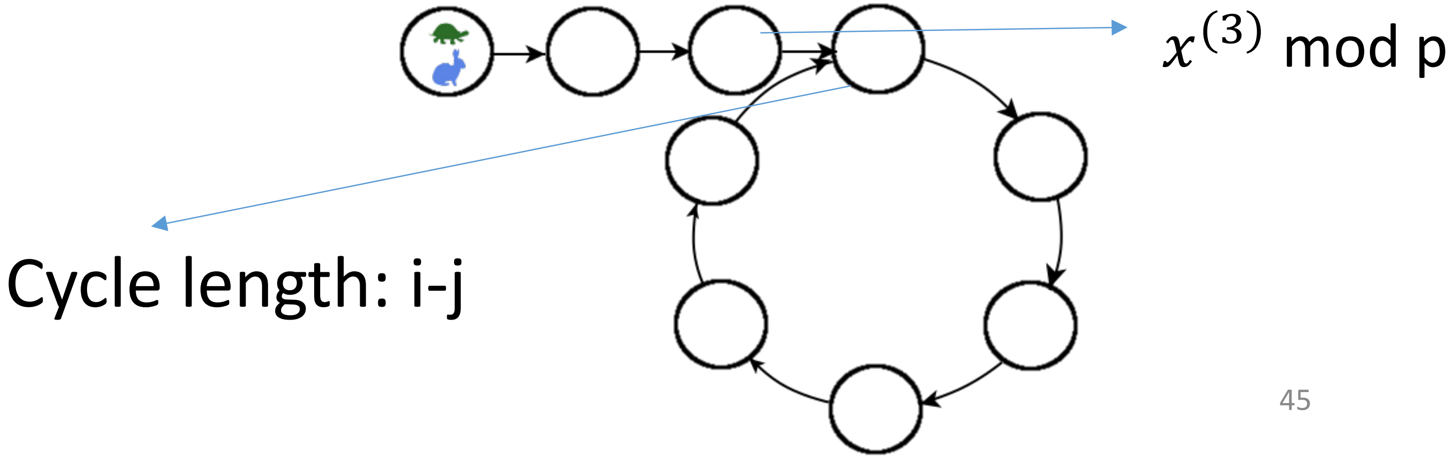
$x \leftarrow F(x)$

$x' \leftarrow F\big(F(x')\big)$

p = **GCD**(x-x',N)

if 1< p < N **return** p

Claim: Let $x^{(i+1)} = F\big(x^{(i)}\big)$ and suppose that for some distinct $\mathrm{i}, \mathrm{j} < 2^{n/2}$ we have $x^{(i)} = x^{(j)} \bmod \mathrm{p}$ but $x^{(i)} \neq x^{(j)}$. Then the algorithm will find p.



Pollard's Rho Algorithm (Summary)

- General Purpose Factoring Algorithm
  - Doesn't assume (p-1) has no large prime factor

- Expected Running Time: $O\left(\sqrt[4]{N}\ \text{pol}ylog(N)\right)$
  - (Birthday Bound)
  - (still exponential in number of bits $\sim 2^{n/4}$)
- Required Space: $O(\log(N))$

Quadratic Sieve Algorithm

- Runs in sub-exponential time $2^{O\left(\sqrt{\log N\ \log\log N}\right)} = 2^{O\left(\sqrt{n \log n}\right)}$
  - Still not polynomial time but $2^{\sqrt{n \log n}}$ grows much slower than $2^{n/4}$.

- **Core Idea**: Find $\mathrm{x}, \mathrm{y} \in \mathbb{Z}_N^*$ such that

$$x^2 = y^2 \bmod N$$

and

$$x \neq \pm y \bmod N$$

Quadratic Sieve Algorithm

- **Core Idea**: Find $\mathrm{x}, \mathrm{y} \in \mathbb{Z}_N^*$ such that

$$x^2 = y^2 \bmod N \quad (1)$$

and

$$x \neq \pm y \bmod N \quad (2)$$

Claim: gcd(x-y,N)∈ $\{p, q\}$

→N=pq divides $x^2 - y^2 = (x-y)(x+y)$. (by (1)).

→$(x-y)(x+y) \neq 0$ (by (2)).

→N does not divide $(x-y)$ (by (2)).

→N does not divide $(x+y)$. (by (2)).

→p *is a factor of exactly one of the terms* $(x-y)$ *and* $(x+y)$.

→*(q is a factor of the other term)*

Quadratic Sieve Algorithm

- **Core Idea**: Find $\mathrm{x}, \mathrm{y} \in \mathbb{Z}_N^*$ such that

$$x^2 = y^2 \bmod N$$

and

$$x \neq \pm y \bmod N$$

- **Key Question**: How to find such an $\mathrm{x}, \mathrm{y} \in \mathbb{Z}_N^*$?
- **Step 1:**

j=0;

For $\mathrm{x} = \sqrt{N} + 1, \sqrt{N} + 2, \ldots, \sqrt{N} + i, \ldots$

$$\mathrm{q} \leftarrow \left[\left(\sqrt{N} + i\right)^2 \ mod\ N\right] = \left[2i\sqrt{N} + i^2\, mod\ N\right]$$

Check if q is B-smooth (all prime factors of q are in $\{p_1,\ldots,p_k\}$ where $p_k < B$).

If q is B smooth then factor q, increment j and define

$$\mathrm{q}_\mathrm{j} \leftarrow q = \prod_{i=1}^{k} p_i^{e_{j,i}},$$

Quadratic Sieve Algorithm

- **Core Idea**: Find $\mathrm{x}, \mathrm{y} \in \mathbb{Z}_N^*$ such that

$$x^2 = y^2 \bmod N$$

and

$$x \neq \pm y \bmod N$$

- **Key Question**: How to find such an $\mathrm{x}, \mathrm{y} \in \mathbb{Z}_N^*$?
- **Step 2:** Once we have $\ell > k$ equations of the form

$$\mathrm{q_j} \leftarrow q = \prod_{i=1}^{k} p_i^{e_{j,i}},$$

We can use linear algebra to find S such that for each $i \leq k$ we have

$$\sum_{j \in S} e_{j,i} = 0 \; mod\; 2.$$

Quadratic Sieve Algorithm

- **Key Question**: How to find $x, y \in \mathbb{Z}_N^*$ such that $x^2 = y^2 \bmod N$ and $x \neq \pm y \bmod N$?
- **Step 2:** Once we have $\ell > k$ equations of the form

$$\mathrm{q_j} \leftarrow q = \prod_{i=1}^{k} p_i^{e_{j,i}},$$

We can use linear algebra to find a subset S such that for each i $\leq$ k we have

$$\sum_{j \in S} e_{j,i} = 0 \; mod \; 2.$$

Thus,

$$\prod_{j \in S} \mathrm{q_j} = \prod_{i=1}^{k} p_i^{\sum_{j \in S} e_{j,i}} = \left(\prod_{i=1}^{k} p_i^{\frac{1}{2} \sum_{j \in S} e_{j,i}} \right)^2 = y^2$$

Quadratic Sieve Algorithm

- **Key Question**: How to find $\mathrm{x}, \mathrm{y} \in \mathbb{Z}_N^*$ such that $x^2 = y^2 \bmod N$ and $x \neq \pm y \bmod N$?

Thus,

$$\prod_{j\in S} \mathrm{q_j} = \prod_{i=1}^{k} p_i^{\sum_{j\in S} e_{j,i}} = \left(\prod_{i=1}^{k} p_i^{\frac{1}{2}\sum_{j\in S} e_{j,i}}\right)^2 = y^2$$

But we also have

$$\prod_{j\in S} \mathrm{q_j} = \prod_{j\in S} (x_j{}^2) = \left(\prod_{j\in S} x_j\right)^2 = x^2 \, mod \, N$$

Quadratic Sieve Algorithm (Summary)

- Appropriate parameter tuning yields sub-exponential time algorithm $2^{O\left(\sqrt{\log N \ \log\log N}\right)} = 2^{O\left(\sqrt{n \log n}\right)}$
  - Still not polynomial time but $2^{\sqrt{n \log n}}$ grows much slower than $2^{n/4}$.

Discrete Log Attacks

- Pohlig-Hellman Algorithm
  - Given a cyclic group $\mathbb{G}$ of non-prime order q=| $\mathbb{G}$ |=rp
  - Reduce discrete log problem to discrete problem(s) for subgroup(s) of order p (or smaller).
  - Preference for prime order subgroups in cryptography
- Baby-step/Giant-Step Algorithm
  - Solve discrete logarithm in time $O\left(\sqrt{q}\, polylog(q)\right)$
- Pollard's Rho Algorithm
  - Solve discrete logarithm in time $O\left(\sqrt{q}\, polylog(q)\right)$
  - Bonus: Constant memory!
- Index Calculus Algorithm
  - Similar to quadratic sieve
  - Runs in sub-exponential time $2^{O\left(\sqrt{\log q \log \log q}\right)}$
  - Specific to the group $\mathbb{Z}_p^*$ (e.g., attack doesn't work elliptic-curves)

Discrete Log Attacks

- **Pohlig-Hellman Algorithm**
  - Given a cyclic group $\mathbb{G}$ of non-prime order q=| $\mathbb{G}$ |=rp
  - Reduce discrete log problem to discrete problem(s) for subgroup(s) of order p (or smaller).
  - Preference for prime order subgroups in cryptography
- Let $\mathbb{G} = \langle g \rangle$ and $\mathrm{h} = g^x \in \mathbb{G}$ be given. For simplicity assume that r is prime and r < p.
- Observe that $\langle g^r \rangle$ generates a subgroup of size p and that $\mathrm{h}^{\mathrm{r}} \in \langle g^r \rangle$.
  - Solve discrete log problem in subgroup $\langle g^r \rangle$ with input h^{r}.
  - Find z such that $\mathrm{h}^{\mathrm{rz}} = g^{rz}$.
- Observe that $\langle g^p \rangle$ generates a subgroup of size p and that $\mathrm{h}^{\mathrm{p}} \in \langle g^p \rangle$.
  - Solve discrete log problem in subgroup $\langle g^p \rangle$ with input h^{p}.
  - Find y such that $\mathrm{h}^{\mathrm{yp}} = g^{yp}$.
- Chinese Remainder Theorem $\mathrm{h} = g^x$ where $\mathrm{x} \leftrightarrow ([z \bmod p], [y \bmod r])$

Baby-step/Giant-Step Algorithm

- Input: $\mathbb{G} = \langle g \rangle$ of order q, generator g and $\mathrm{h} = g^x \in \mathbb{G}$
- Set $t = \lfloor \sqrt{q} \rfloor$

For i =0 to $\left\lfloor \frac{q}{t} \right\rfloor$

$$g_i \leftarrow g^{it}$$

Sort the pairs (i,g_i) by their second component

For i =0 to t

$h_i \leftarrow hg^i$

if $h_i = gk \in \{g_0, \dots, g_t\}$ then

return [kt-i mod q]

$$h_i = hg^i = g^{kt}$$

$$\rightarrow h = g^{kt-i}$$

Discrete Log Attacks

- Baby-step/Giant-Step Algorithm
  - Solve discrete logarithm in time $O\left(\sqrt{q}\; polylog(q)\right)$
  - Requires memory $O\left(\sqrt{q}\; polylog(q)\right)$
- Pollard's Rho Algorithm
  - Solve discrete logarithm in time $O\left(\sqrt{q}\; polylog(q)\right)$
  - Bonus: Constant memory!
- **Key Idea:** Low-Space Birthday Attack (*) using our collision resistant hash function

$$H_{g,h}(x_1, x_2) = g^{x_1} h^{x_2}$$

$$H_{g,h}(y_1, y_2) = H_{g,h}(x_1, x_2) \rightarrow \; h^{y_2 - x_2} = g^{x_1 - y_1}$$

$$\rightarrow h = g^{(x_1 - y_1)(y_2 - x_2)^{-1}}$$

(*) A few small technical details to address

Discrete Log Attacks

- Baby-step/Giant-Step Algorithm
  - Solve discrete logarithm in time $O\left(\sqrt{q}\, polylog(q)\right)$
  - Requires memory $O\left(\sqrt{q}\, polylog(q)\right)$
- Pollard's Rho Algorithm
  - Solve discrete logarithm in time $O\left(\sqrt{q}\, pol$
  - Bonus: Constant memory!
- **Key Idea:** Low-Space Birthday Attack (*)

$$H_{g,h}(x_1, x_2) = g^{x_1} h^{x_2}$$
$$H_{g,h}(y_1, y_2) = H_{g,h}(x_1, x_2)$$

$$\rightarrow\ h^{y_2 - x_2} = g^{x_1 - y_1}$$
$$\rightarrow h = g^{(x_1 - y_1)(y_2 - x_2)^{-1}}$$

(*) A few small technical details to address

Remark: We used discrete-log problem to construct collision resistant hash functions.

Security Reduction showed that attack on collision resistant hash function yields attack on discrete log.

→Generic attack on collision resistant hash functions (e.g., low space birthday attack) yields generic attack on discrete log.

Discrete Log Attacks

- Index Calculus Algorithm
  - Similar to quadratic sieve
  - Runs in sub-exponential time $2^{O\left(\sqrt{\log q \log\log q}\right)}$
  - Specific to the group $\mathbb{Z}_p^*$ (e.g., attack doesn't work elliptic-curves)

- As before let $\{p_1,\ldots,p_k\}$ be set of prime numbers < B.
- **Step 1.A:** Find $\ell > k$ distinct values $x_1, \ldots, x_k$ such that $g_j = [g^{x_j} \; mod \; p]$ is B-smooth for each j. That is

$$g_j = \prod_{i=1}^{k} p_i^{e_{i,j}} \,.$$

Discrete Log Attacks

- As before let $\{p_1,\ldots,p_k\}$ be set of prime numbers < B.
- **Step 1.A:** Find $\ell > k$ distinct values $x_1, \ldots, x_k$ such that $g_j = [g^{x_j} \, mod \, p]$ is B-smooth for each j. That is

$$g_j = \prod_{i=1}^{k} p_i^{e_{i,j}} .$$

- **Step 1.B:** Use linear algebra to solve the equations

$$x_j = \sum_{i=1}^{k} \left(\mathbf{log_g \, p_i} \right) \times e_{i,j} \ \text{ mod } (p-1).$$

(Note: the $\mathbf{log_g p_i}$'s are the unknowns)

Discrete Log

- As before let $\{p_1,\ldots,p_k\}$ be set of prime numbers < B.
- **Step 1 (precomputation):** Obtain $y_1,\ldots,y_k$ such that $p_i = g^{y_i} \, mod \, p$.
- **Step 2:** Given discrete log challenge $h=g^x$ mod p.
  - Find y such that $[g^y \mathrm{h \ mod \ p}]$ is B-smooth

$$[g^y \mathrm{h \ mod \ p}] = \prod_{i=1}^{k} p_i^{e_i}$$

$$= \prod_{i=1}^{k} (g^{y_i})^{e_i} = g^{\sum_i e_i y_i}$$

Discrete Log

- As before let $\{p_1,\ldots,p_k\}$ be set of prime numbers < B.
- **Step 1 (precomputation):** Obtain $y_1,\ldots,y_k$ such that $p_i = g^{y_i} \, mod \, p$.
- **Step 2:** Given discrete log challenge $h=g^x$ mod p.
  - Find z such that $[g^z \mathrm{h \ mod \ p}]$ is B-smooth

$$[g^z \mathrm{h \ mod \ p}] = g^{\sum_i e_i y_i} \rightarrow h = g^{\sum_i e_i y_i - z}$$

$$\rightarrow x = \sum_i e_i y_i - z$$

- **Remark:** Precomputation costs can be amortized over many discrete log instances
  - In practice, the same group $\mathbb{G} = \langle g \rangle$ and generator g are used repeatedly.

Reference: https://www.weakdh.org/

NIST Guidelines (Concrete Security)

Best known attack against 1024 bit RSA takes time (approximately) 2^{80}

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Table 1: NIST Recommended Key Sizes

NIST Guidelines (Concrete Security)

Diffie-Hellman uses subgroup of $\mathbb{Z}_p^*$ size q

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)		Elliptic Curve Key Size (bits)
80	1024		160
112	2048	q=224 bits	224
128	3072	q=256 bits	256
192	7680	q=384 bits	384
256	15360	q=512 bits	521

Table 1: NIST Recommended Key Sizes

Security Strength		2011 through 2013	2014 through 2030	2031 and Beyond
80	Applying	Deprecated	Disallowed	
	Processing	Legacy use		
112	Applying	Acceptable	Acceptable	Disallowed
	Processing			Legacy use
128	Applying/Processing	Acceptable	Acceptable	Acceptable
192		Acceptable	Acceptable	Acceptable
256		Acceptable	Acceptable	Acceptable

NIST's security strength guidelines, from Specialist Publication SP 800-57
Recommendation for Key Management – Part 1: General (Revision 3)