Homework 3

- As announced: not due today ☺

- Due <u>Friday</u> at the beginning of class.

Cryptography
CS 555

Topic 27: Factoring Algorithm, Discrete Log Attacks + NIST Recommendations for Concrete Security Parameters

Recap

- OWFs + CRHFs from Discrete Log + Factoring
- Pollards (p-1) algorithm
  - (works when N=pq and (p-1) has only “small” prime factors)

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: $O\left(\sqrt[4]{N}\ \mathrm{pol}ylog(N)\right)$
  - Naïve Algorithm takes time $O\left(\sqrt{N}\ \mathrm{pol}ylog(N)\right)$ 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 $x = x' \bmod p$
  - 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 $x^{(1)}, \ldots, x^{(k)} \in \mathbb{Z}_p^*$ then what is the probability that we can find distinct i and j such that

$$x^{(i)} = x^{(j)} \bmod \mathrm{p}?$$

Pollard's Rho Algorithm

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

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)

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

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)

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

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

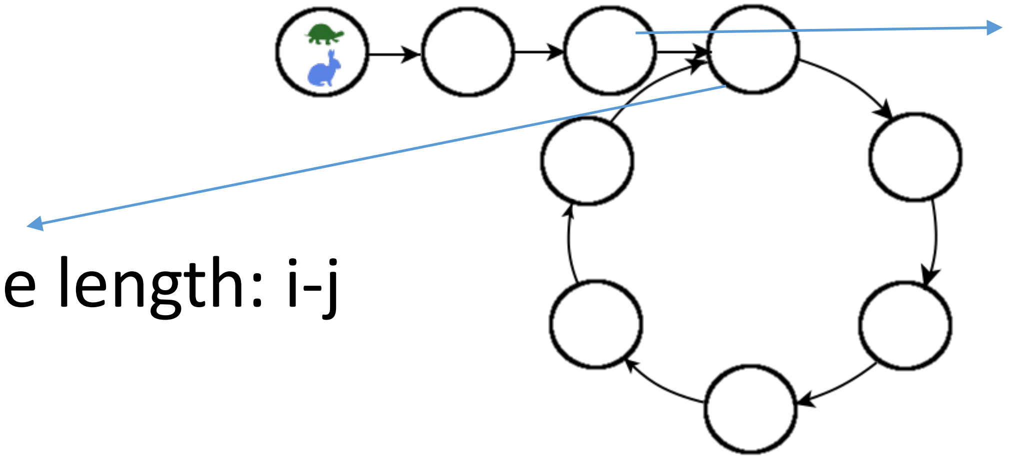
  $x \leftarrow F(x)$

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

  p = **GCD**(x-x’,N)

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

Claim: Let $x^{(i+1)} = F\left(x^{(i)}\right)$ 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

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

- Succeeds with constant probability

Quadratic Sieve Algorithm

- Runs in subexponential 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 $\mathrm{x} = \sqrt{N} + 1, \sqrt{N} + 2, \dots, \sqrt{N} + i$,…

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

Check if q is B-smooth (all prime factors of q are in $\{p_1,\dots,p_k\}$ 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 $O\left(\sqrt{q}\; pol\right.$
  - 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

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

(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 $[g^z \text{h mod p}]$ is B-smooth

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

- **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

<table>
<tr><th colspan="2">Security Strength</th><th>2011 through 2013</th><th>2014 through 2030</th><th>2031 and Beyond</th></tr>
<tr><td rowspan="2">80</td><td>Applying</td><td>Deprecated</td><td colspan="2">Disallowed</td></tr>
<tr><td>Processing</td><td colspan="3">Legacy use</td></tr>
<tr><td rowspan="2">112</td><td>Applying</td><td rowspan="2">Acceptable</td><td rowspan="2">Acceptable</td><td>Disallowed</td></tr>
<tr><td>Processing</td><td>Legacy use</td></tr>
<tr><td>128</td><td rowspan="3">Applying/Processing</td><td>Acceptable</td><td>Acceptable</td><td>Acceptable</td></tr>
<tr><td>192</td><td>Acceptable</td><td>Acceptable</td><td>Acceptable</td></tr>
<tr><td>256</td><td>Acceptable</td><td>Acceptable</td><td>Acceptable</td></tr>
</table>

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

Next Class: Key Management

- Key Management
- Read Katz and Lindell: Chapter 10