Mathematical Cryptography

Diffie Hellman, Discrete Log Problem, Collision Algorithms

Mentor: Tao Song , Mentee: Lisette del Pino

University of Pennsylvania Directed Reading Program

May 16, 2020

You and your friend want to exchange secret messages, in this case, these numbers translate to words. You and your friend live very far away from each other and can't meet up to share secrets. What do you do?

You and your friend want to exchange secret messages, in this case, these numbers translate to words. You and your friend live very far away from each other and can't meet up to share secrets. What do you do?

1. First, pick prime p, and a nonzero integer g mod p.

You and your friend want to exchange secret messages, in this case, these numbers translate to words. You and your friend live very far away from each other and can't meet up to share secrets. What do you do?

1. First, pick prime p, and a nonzero integer g mod p.
2. You: pick a secret integer a. Compute $A \equiv g^a \bmod p$

You and your friend want to exchange secret messages, in this case, these numbers translate to words. You and your friend live very far away from each other and can't meet up to share secrets. What do you do?

1. First, pick prime p, and a nonzero integer g mod p.
2. You: pick a secret integer a. Compute $A \equiv g^a \mod p$
3. Your friend: pick a secret integer b, Compute $B \equiv g^b \mod p$

Diffie Hellman Public Key Exchange

A Privacy Dillema

You and your friend want to exchange secret messages, in this case, these numbers translate to words. You and your friend live very far away from each other and can't meet up to share secrets. What do you do?

1. First, pick prime p, and a nonzero integer g mod p.
2. You: pick a secret integer a. Compute $A \equiv g^a \bmod p$
3. Your friend: pick a secret integer b, Compute $B \equiv g^b \bmod p$
4. Send A to your friend. Your friend sends B to you.

You and your friend want to exchange secret messages, in this case, these numbers translate to words. You and your friend live very far away from each other and can't meet up to share secrets. What do you do?

1. First, pick prime p, and a nonzero integer g mod p.
2. You: pick a secret integer a. Compute $A \equiv g^a \mod p$
3. Your friend: pick a secret integer b, Compute $B \equiv g^b \mod p$
4. Send A to your friend. Your friend sends B to you.
5. In private, you compute $B' \equiv B^a \mod p$

You and your friend want to exchange secret messages, in this case, these numbers translate to words. You and your friend live very far away from each other and can't meet up to share secrets. What do you do?

1. First, pick prime p, and a nonzero integer g mod p.
2. You: pick a secret integer a. Compute $A \equiv g^a \mod p$
3. Your friend: pick a secret integer b, Compute $B \equiv g^b \mod p$
4. Send A to your friend. Your friend sends B to you.
5. In private, you compute $B' \equiv B^a \mod p$
6. In private, your friend computes $A' \equiv A^b \mod p$

You and your friend want to exchange secret messages, in this case, these numbers translate to words. You and your friend live very far away from each other and can't meet up to share secrets. What do you do?

1. First, pick prime p, and a nonzero integer g mod p.
2. You: pick a secret integer a. Compute $A \equiv g^a \mod p$
3. Your friend: pick a secret integer b, Compute $B \equiv g^b \mod p$
4. Send A to your friend. Your friend sends B to you.
5. In private, you compute $B' \equiv B^a \mod p$
6. In private, your friend computes $A' \equiv A^b \mod p$
7. This is the shared value. B' and A' are the same.

You and your friend want to exchange secret messages, in this case, these numbers translate to words. You and your friend live very far away from each other and can't meet up to share secrets. What do you do?

1. First, pick prime p, and a nonzero integer g mod p.
2. You: pick a secret integer a. Compute $A \equiv g^a \bmod p$
3. Your friend: pick a secret integer b, Compute $B \equiv g^b \bmod p$
4. Send A to your friend. Your friend sends B to you.
5. In private, you compute $B' \equiv B^a \bmod p$
6. In private, your friend computes $A' \equiv A^b \bmod p$
7. This is the shared value. B' and A' are the same.
8. Proof: $A' \equiv B^a \equiv g^{ba} \equiv A^b \equiv B' \bmod p$

A Numerical Example of the DHKE

Let's try an explicit example with relatively small numbers. In practice, you should pick primes about 4000 bits long.

A Numerical Example of the DHKE

Let's try an explicit example with relatively small numbers. In practice, you should pick primes about 4000 bits long.

1. First, pick prime $p = 941$, and a nonzero integer $g = 627 \bmod 941$.

Recall eavesdroppers know these values : A, B, g^a, g^b, g, p And they need to find: g^{ab} This problem is no harder than the Discrete Logarithm Problem

A Numerical Example of the DHKE

Let's try an explicit example with relatively small numbers. In practice, you should pick primes about 4000 bits long.

1. First, pick prime $p = 941$, and a nonzero integer $g = 627 \bmod 941$.
2. You: pick a secret integer $a = 347$. Compute $A = 390$

Recall eavesdroppers know these values : A, B, g^a, g^b, g, p And they need to find: g^{ab} This problem is no harder than the Discrete Logarithm Problem

A Numerical Example of the DHKE

Let's try an explicit example with relatively small numbers. In practice, you should pick primes about 4000 bits long.

1. First, pick prime $p = 941$, and a nonzero integer $g = 627 \bmod 941$.
2. You: pick a secret integer $a = 347$. Compute $A = 390$
3. Your friend: pick a secret integer $b = 781$, Compute $B = 691$

Recall eavesdroppers know these values : A, B, g^a, g^b, g, p And they need to find: g^{ab} This problem is no harder than the Discrete Logarithm Problem

A Numerical Example of the DHKE

Let's try an explicit example with relatively small numbers. In practice, you should pick primes about 4000 bits long.

1. First, pick prime $p = 941$, and a nonzero integer $g = 627 \bmod 941$.
2. You: pick a secret integer $a = 347$. Compute $A = 390$
3. Your friend: pick a secret integer $b = 781$, Compute $B = 691$
4. Send A to your friend. Your friend sends B to you.

Recall eavesdroppers know these values : A, B, g^a, g^b, g, p And they need to find: g^{ab} This problem is no harder than the Discrete Logarithm Problem

A Numerical Example of the DHKE

Let's try an explicit example with relatively small numbers. In practice, you should pick primes about 4000 bits long.

1. First, pick prime $p = 941$, and a nonzero integer $g = 627$ mod 941.
2. You: pick a secret integer $a = 347$. Compute $A = 390$
3. Your friend: pick a secret integer $b = 781$, Compute $B = 691$
4. Send A to your friend. Your friend sends B to you.
5. The shared value is $470 \equiv 627^{347*781}$ mod 941

Recall eavesdroppers know these values : A, B, g^a, g^b, g, p And they need to find: g^{ab} This problem is no harder than the Discrete Logarithm Problem

Discrete Logarithm Problem

Recall that computing the shared value g^{ab} is no harder than solving the Discrete Log Problem.

Discrete Logarithm Problem

Recall that computing the shared value g^{ab} is no harder than solving the Discrete Log Problem.

Fermat's Little Theorem: if p is prime, any integer g gives us

$$g^{p-1} \equiv 1 \mod p$$

Discrete Logarithm Problem

Recall that computing the shared value g^{ab} is no harder than solving the Discrete Log Problem.
Fermat's Little Theorem: if p is prime, any integer g gives us

$$g^{p-1} \equiv 1 \mod p$$

recall elements of a finite multiplicative group $\mathbb{F}_p^*$ with a generator g are:

$$1, g^1, g^2, ..., g^{p-2}$$

Discrete Logarithm Problem

Recall that computing the shared value g^{ab} is no harder than solving the Discrete Log Problem.
Fermat's Little Theorem: if p is prime, any integer g gives us

$$g^{p-1} \equiv 1 \mod p$$

recall elements of a finite multiplicative group $\mathbb{F}_p^*$ with a generator g are:

$$1, g^1, g^2, ..., g^{p-2}$$

where $g^{p-1} \equiv 1 \mod p$ by Fermat's Little Theorem.

Discrete Logarithm Problem

Recall that computing the shared value g^{ab} is no harder than solving the Discrete Log Problem.

Discrete Logarithm Problem

Recall that computing the shared value g^{ab} is no harder than solving the Discrete Log Problem.

Discrete Log Problem: Given a primitive root (generator) g of a finite group $G = \mathbb{F}_p^*$

Discrete Logarithm Problem

Recall that computing the shared value g^{ab} is no harder than solving the Discrete Log Problem.

Discrete Log Problem: Given a primitive root (generator) g of a finite group $G = \mathbb{F}_p^*$ and $h \neq 0 \in G = \mathbb{F}_{|}{}^{*}$

Discrete Logarithm Problem

Recall that computing the shared value g^{ab} is no harder than solving the Discrete Log Problem.
Discrete Log Problem: Given a primitive root (generator) g of a finite group $G = \mathbb{F}_p^*$ and $h \neq 0 \in G = \mathbb{F}_l^*$
find an x such that $g^x \equiv h \mod p$

Discrete Logarithm Problem

Discrete Log Problem: find an x such that $g^x \equiv h \bmod p$

Discrete Logarithm Problem

Discrete Log Problem: find an x such that $g^x \equiv h \bmod p$
Notice that if there is one such x, there are many. The solution is not unique!

Discrete Logarithm Problem

Discrete Log Problem: find an x such that $g^x \equiv h \bmod p$
Notice that if there is one such x, there are many. The solution is not unique!
Proof: if x solves $g^x \equiv h \mod p$, then so does $x + k(p-1) \forall k$

Discrete Logarithm Problem

Discrete Log Problem: find an x such that $g^x \equiv h \bmod p$
Notice that if there is one such x, there are many. The solution is not unique!
Proof: if x solves $g^x \equiv h \mod p$, then so does $x + k(p-1) \forall k$ we have:

$$g^{x+k(p-1)} = g^x g^{p-1^k} = h * 1^k \equiv h \mod p$$

Discrete Logarithm Problem

Discrete Log Problem: find an x such that $g^x \equiv h \bmod p$

Discrete Logarithm Problem

Discrete Log Problem: find an x such that $g^x \equiv h \bmod p$
We can also restate the D.L.P in terms of Group Theory:

Discrete Logarithm Problem

Discrete Log Problem: find an x such that $g^x \equiv h \bmod p$
We can also restate the D.L.P in terms of Group Theory:
Let $g \in G$, G is a finite group. x is a positive integer and star is the group operation.

$$g^x = g * g * g * ... * g$$

Order of a Group, Order of an Element

The order of a group is its cardinality.

Order of a Group, Order of an Element

The order of a group is its cardinality.
The order of an element a of that group is:

Order of a Group, Order of an Element

The order of a group is its cardinality.
The order of an element a of that group is:
a postive integer d s.t $a^d = e$, where e is the identity element.
otherwise, infinite order.

Big O

We need to quantify difficulty of the Discrete Logarithm Problem

Big O

We need to quantify difficulty of the Discrete Logarithm Problem
Big O: $f(x) = O(g(x))$ if there exist positive constants c and C s.t.

$$f(x) \leq cg(x), \forall x \geq C$$

Big O

We need to quantify difficulty of the Discrete Logarithm Problem
Big O: $f(x) = O(g(x))$ if there exist positive constants c and C s.t.

$$f(x) \leq cg(x), \forall x \geq C$$

also,

$$\lim_{x \to +\infty} \frac{f(x)}{g(x)}$$

exists and is finite

Brute Force the Discrete Log

We need to quantify difficulty of the Discrete Logarithm Problem

Brute Force the Discrete Log

We need to quantify difficulty of the Discrete Logarithm Problem
Since our group has order p and we are applying group operations at most p times, D.L.P runtime is $O(p)$

Brute Force the Discrete Log

We need to quantify difficulty of the Discrete Logarithm Problem
Since our group has order p and we are applying group operations at most p times, D.L.P runtime is $O(p)$
we chose our prime p to be a k-bit number (so a binary number), so it's approximately 2^k

We need to quantify difficulty of the Discrete Logarithm Problem
Since our group has order p and we are applying group operations at most p times, D.L.P runtime is $O(p)$
we chose our prime p to be a k-bit number (so a binary number), so it's approximately 2^k
For our computer then, the runtime is

$$O(2^k)$$

for the trial and error method. Pretty awful.

Fast Exponentiation

Computers brute forcing the D.L.P use the Fast Exponentiation Method

Fast Exponentiation

Computers brute forcing the D.L.P use the Fast Exponentiation Method

$$(A^2 \mod C = A * A \mod C = A \mod C * A \mod C (\mod C)$$

Fast Exponentiation

Computers brute forcing the D.L.P use the Fast Exponentiation Method

$$(A^2 \mod C = A*A \mod C = A \mod C*A \mod C(\mod C)$$

For our computer then, the runtime is

$$O(2^k * k)$$

for the trial and error method. Still exponential time.

Shanks Baby-Step Giant-Step Algorithm

SBSGS runtime: $O(\sqrt{}(N))$

Shanks Baby-Step Giant-Step Algorithm

SBSGS runtime: $O(\sqrt(N))$
Let G be a finite group.

Shanks Baby-Step Giant-Step Algorithm

SBSGS runtime: $O(\sqrt{(}N))$
Let G be a finite group.
Let $g \in G$ is an element with order $N \geq 2$.

Shanks Baby-Step Giant-Step Algorithm

SBSGS runtime: $O(\sqrt{(}N))$
Let G be a finite group.
Let $g \in G$ is an element with order $N \geq 2$.
Then we can find x, where $g^x \equiv h \mod p$, in at most $O(\sqrt{N} logN)$ steps

Shanks Baby-Step Giant-Step Algorithm

SBSGS runtime: $O(\sqrt(N))$
Let G be a finite group.
Let $g \in G$ is an element with order $N \geq 2$.
Then we can find x, where $g^x \equiv h \mod p$, in at most $O(\sqrt{N} logN)$ steps

1. Let $n = 1 + floor(\sqrt{N})$, so that $n \geq \sqrt{N}$

Shanks Baby-Step Giant-Step Algorithm

SBSGS runtime: $O(\sqrt(N))$
Let G be a finite group.
Let $g \in G$ is an element with order $N \geq 2$.
Then we can find x, where $g^x \equiv h \mod p$, in at most $O(\sqrt{N} logN)$ steps

1. Let $n = 1 + floor(\sqrt{N})$, so that $n \geq \sqrt{N}$
2. Create two lists (use a hash table for efficient lookup!)

Shanks Baby-Step Giant-Step Algorithm

SBSGS runtime: $O(\sqrt{}(N))$
Let G be a finite group.
Let $g \in G$ is an element with order $N \geq 2$.
Then we can find x, where $g^x \equiv h \mod p$, in at most $O(\sqrt{N} logN)$ steps

1. Let $n = 1 + floor(\sqrt{N})$, so that $n \geq \sqrt{N}$
2. Create two lists (use a hash table for efficient lookup!)
3. List 1: $e, g, g^2, ...g^n$ (recall $n \geq \sqrt{N}$)

Shanks Baby-Step Giant-Step Algorithm

SBSGS runtime: $O(\sqrt(N))$
Let G be a finite group.
Let $g \in G$ is an element with order $N \geq 2$.
Then we can find x, where $g^x \equiv h \mod p$, in at most $O(\sqrt{N} logN)$ steps

1. Let $n = 1 + floor(\sqrt{N})$, so that $n \geq \sqrt{N}$
2. Create two lists (use a hash table for efficient lookup!)
3. List 1: $e, g, g^2, ...g^n$ (recall $n \geq \sqrt{N}$)
4. List 2: $h, h * g^{-n}, h * g^{-2n}, ..., h * g^{n^2}$

Shanks Baby-Step Giant-Step Algorithm

SBSGS runtime: $O(\sqrt{(}N))$
Let G be a finite group.
Let $g \in G$ is an element with order $N \geq 2$.
Then we can find x, where $g^x \equiv h \mod p$, in at most $O(\sqrt{N} logN)$ steps

1. Let $n = 1 + floor(\sqrt{N})$, so that $n \geq \sqrt{N}$
2. Create two lists (use a hash table for efficient lookup!)
3. List 1: $e, g, g^2, ...g^n$ (recall $n \geq \sqrt{N}$)
4. List 2: $h, h * g^{-n}, h * g^{-2n}, ..., h * g^{n^2}$
5. Find a match between your two lists. If it exists, it's $g^i = hg^{-jn}$, i, j indices

Shanks Baby-Step Giant-Step Algorithm

SBSGS runtime: $O(\sqrt{(}N))$
Let G be a finite group.
Let $g \in G$ is an element with order $N \geq 2$.
Then we can find x, where $g^x \equiv h \mod p$, in at most $O(\sqrt{N} logN)$ steps

1. Let $n = 1 + floor(\sqrt{N})$, so that $n \geq \sqrt{N}$
2. Create two lists (use a hash table for efficient lookup!)
3. List 1: $e, g, g^2, ...g^n$ (recall $n \geq \sqrt{N}$)
4. List 2: $h, h * g^{-n}, h * g^{-2n}, ..., h * g^{n^2}$
5. Find a match between your two lists. If it exists, it's $g^i = hg^{-jn}$, i, j indices
6. Then $x = i + jn$ is a solution to $g^x \equiv h \mod p$

Shanks Baby-Step Giant-Step Algorithm Example

1. Use order of the group as estimate. Solve $3^x \equiv 19 mod 59$

Shanks Baby-Step Giant-Step Algorithm Example

1. Use order of the group as estimate. Solve $3^x \equiv 19 mod 59$
2. Set $n = 8$, so $n \geq \sqrt{p-1}$

Shanks Baby-Step Giant-Step Algorithm Example

1. Use order of the group as estimate. Solve $3^x \equiv 19 mod 59$
2. Set $n = 8$, so $n \geq \sqrt{p-1}$

$$\begin{aligned} 3^1 &\equiv 3 \mod 59 \\ 3^2 &\equiv 9 \mod 59 \\ 3^3 &\equiv 27 \mod 59 \\ 3^4 &\equiv 22 \mod 59 \\ 3^5 &\equiv 7 \mod 59 \\ 3^6 &\equiv 21 \mod 59 \\ 3^7 &\equiv 4 \mod 59 \\ 3^8 &\equiv 12 \mod 59 \end{aligned}$$

$$\begin{aligned} 3^{-1} &\equiv 20 \mod 59 \\ 3^{-8} &\equiv 20^8 \mod 59 \equiv 5 \mod 59 \\ 19(3^{-8}) &\equiv 19(20^8) \equiv 19(5) \equiv 36 \mod 59 \\ 19(3^{-16}) &\equiv 19(20^{16}) \equiv 19(25) \equiv 3 \mod 59 \end{aligned}$$

Shanks Baby-Step Giant-Step Algorithm Example

1. Use order of the group as estimate. Solve $3^x \equiv 19 mod 59$
2. Set $n = 8$, so $n \geq \sqrt{p-1}$

$$
\begin{aligned}
3^1 &\equiv 3 \mod 59 \\
3^2 &\equiv 9 \mod 59 \\
3^3 &\equiv 27 \mod 59 \\
3^4 &\equiv 22 \mod 59 \\
3^5 &\equiv 7 \mod 59 \\
3^6 &\equiv 21 \mod 59 \\
3^7 &\equiv 4 \mod 59 \\
3^8 &\equiv 12 \mod 59
\end{aligned}
$$

$$
\begin{aligned}
3^{-1} &\equiv 20 \mod 59 \\
3^{-8} &\equiv 20^8 \mod 59 \equiv 5 \mod 59 \\
19(3^{-8}) &\equiv 19(20^8) \equiv 19(5) \equiv 36 \mod 59 \\
19(3^{-16}) &\equiv 19(20^{16}) \equiv 19(25) \equiv 3 \mod 59
\end{aligned}
$$

3. We're done! We found a match!

Shanks Baby-Step Giant-Step Algorithm Example

1. Use order of the group as estimate. Solve $3^x \equiv 19 mod 59$
2. Set $n = 8$, so $n \geq \sqrt{p-1}$

$$\begin{array}{cc}
3^1 \equiv 3 \mod 59 & 3^{-1} \equiv 20 \mod 59 \\
3^2 \equiv 9 \mod 59 & 3^{-8} \equiv 20^8 \mod 59 \equiv 5 \mod 59 \\
3^3 \equiv 27 \mod 59 & 19(3^{-8}) \equiv 19(20^8) \equiv 19(5) \equiv 36 \mod 59 \\
3^4 \equiv 22 \mod 59 & 19(3^{-16}) \equiv 19(20^{16}) \equiv 19(25) \equiv 3 \mod 59 \\
3^5 \equiv 7 \mod 59 & \\
3^6 \equiv 21 \mod 59 & \\
3^7 \equiv 4 \mod 59 & \\
3^8 \equiv 12 \mod 59 &
\end{array}$$

3. We're done! We found a match!
4. then, $3^1 \equiv 19(20^{16}) \mod 59$

Shanks Baby-Step Giant-Step Algorithm Example

1. Use order of the group as estimate. Solve $3^x \equiv 19 mod 59$
2. Set $n = 8$, so $n \geq \sqrt{p-1}$

$$\begin{aligned} 3^1 &\equiv 3 \mod 59 \\ 3^2 &\equiv 9 \mod 59 \\ 3^3 &\equiv 27 \mod 59 \\ 3^4 &\equiv 22 \mod 59 \\ 3^5 &\equiv 7 \mod 59 \\ 3^6 &\equiv 21 \mod 59 \\ 3^7 &\equiv 4 \mod 59 \\ 3^8 &\equiv 12 \mod 59 \end{aligned}$$

$$\begin{aligned} 3^{-1} &\equiv 20 \mod 59 \\ 3^{-8} &\equiv 20^8 \mod 59 \equiv 5 \mod 59 \\ 19(3^{-8}) &\equiv 19(20^8) \equiv 19(5) \equiv 36 \mod 59 \\ 19(3^{-16}) &\equiv 19(20^{16}) \equiv 19(25) \equiv 3 \mod 59 \end{aligned}$$

3. We're done! We found a match!
4. then, $3^1 \equiv 19(20^{16}) \mod 59$
5. and according to our algorithm, $x = 1 + 16 = 17$

Shanks Baby-Step Giant-Step Algorithm Example

1. Use order of the group as estimate. Solve $3^x \equiv 19 mod 59$
2. Set $n = 8$, so $n \geq \sqrt{p-1}$

$$
\begin{array}{ll}
3^1 \equiv 3 \mod 59 & 3^{-1} \equiv 20 \mod 59 \\
3^2 \equiv 9 \mod 59 & 3^{-8} \equiv 20^8 \mod 59 \equiv 5 \mod 59 \\
3^3 \equiv 27 \mod 59 & 19(3^{-8}) \equiv 19(20^8) \equiv 19(5) \equiv 36 \mod 59 \\
3^4 \equiv 22 \mod 59 & 19(3^{-16}) \equiv 19(20^{16}) \equiv 19(25) \equiv 3 \mod 59 \\
3^5 \equiv 7 \mod 59 & \\
3^6 \equiv 21 \mod 59 & \\
3^7 \equiv 4 \mod 59 & \\
3^8 \equiv 12 \mod 59 &
\end{array}
$$

3. We're done! We found a match!
4. then, $3^1 \equiv 19(20^{16}) \mod 59$
5. and according to our algorithm, $x = 1 + 16 = 17$
6. Thus, $3^{17} \equiv 19 \mod 59$

Chinese Remainder Theorem

Let $m_1, m_2, ...m_k$ be a collection of pairwise relatively prime integers. Just means:

Chinese Remainder Theorem

Let $m_1, m_2, ...m_k$ be a collection of pairwise relatively prime integers. Just means:

$$gcd(m_i, m_j) = 1, \forall i \neq j$$

Chinese Remainder Theorem

Let $m_1, m_2, ...m_k$ be a collection of pairwise relatively prime integers. Just means:

$$gcd(m_i, m_j) = 1, \forall i \neq j$$

Let $a_1, ..., a_k$ be arbitrary integers

Chinese Remainder Theorem

Let $m_1, m_2, ...m_k$ be a collection of pairwise relatively prime integers. Just means:

$$gcd(m_i, m_j) = 1, \forall i \neq j$$

Let $a_1, ..., a_k$ be arbitrary integers
Then the system of congruences:

$$x \equiv a_1 \mod m_1, x \equiv a_2 \mod m_2, ..., x \equiv a_k \mod m_k$$

has a solution $x = c \mod m_1 * ... * m_k$ that is unique.

Chinese Remainder Theorem

Let $m_1, m_2, ...m_k$ be a collection of pairwise relatively prime integers. Just means:

$$gcd(m_i, m_j) = 1, \forall i \neq j$$

Let $a_1, ..., a_k$ be arbitrary integers
Then the system of congruences:

$$x \equiv a_1 \mod m_1, x \equiv a_2 \mod m_2, ..., x \equiv a_k \mod m_k$$

has a solution $x = c \text{ mod } m_1 * ... * m_k$ that is unique.
The C.R.T allows us to solve systems of modular congruences.

Pohlig Hellman Algorithm Procedure

$G = F_p^*, g \in G$ has a prime power order.

Pohlig Hellman Algorithm Procedure

$G = F_p^*, g \in G$ has a prime power order.
$g \in G$ has order q^e, so $g^{q^e} \equiv e \mod p$

Pohlig Hellman Algorithm Procedure

$G = F_p^*, g \in G$ has a prime power order.
$g \in G$ has order q^e, so $g^{q^e} \equiv e \mod p$
Then this algorithm lets us solve D.L.P in $O(Sq^e)$ steps.

Pohlig Hellman Algorithm Procedure

$G = F_p^*, g \in G$ has a prime power order.
$g \in G$ has order q^e, so $g^{q^e} \equiv e \mod p$
Then this algorithm lets us solve D.L.P in $O(Sq^e)$ steps.
In our worst case, where we can't decompose p into small primes, we use Shanks instead, so $q^e = g^{e1/2} = \sqrt{N}$

Pohlig Hellman Algorithm Procedure

if our order factors into primes, then:

$$N = q_2^{e_2}, q_2^{e_2}, ..., q_t^{e_t}$$

Pohlig Hellman Algorithm Procedure

if our order factors into primes, then:

$$N = q_2^{e_2}, q_2^{e_2}, ..., q_t^{e_t}$$

Then we solve D.L.P in $O(\sum_{i=1}^{t} Sq_i e_i + logN)$

Pohlig Hellman Algorithm Procedure

if our order factors into primes, then:

$$N = q_2^{e_2}, q_2^{e_2}, ..., q_t^{e_t}$$

Then we solve D.L.P in $O(\sum_{i=1}^{t} Sq_i e_i + logN)$
Let's look at the procedure

Pohlig Hellman Algorithm Procedure

1. $\forall 1 \leq i \leq t$, let $g_i = g^{N/q_i^{e_i}}$ and let $h_i = h^{N/q_i^{e_i}}$

Pohlig Hellman Algorithm Procedure

1. $\forall 1 \leq i \leq t$, let $g_i = g^{N/q_i^{e_i}}$ and let $h_i = h^{N/q_i^{e_i}}$
2. since g_i has prime power order $q_i^{e_{is}}$, we meed to solve each Discrete Log Problem $g_i^y = h_i$

Pohlig Hellman Algorithm Procedure

1. $\forall 1 \leq i \leq t$, let $g_i = g^{N/q_i^{e_i}}$ and let $h_i = h^{N/q_i^{e_i}}$
2. since g_i has prime power order $q_i^{e_{is}}$, we meed to solve each Discrete Log Problem $g_i^y = h_i$
3. Use the Chinese Remainder Theorem to solve each modular congruence

$$x_1 \equiv y_1 \mod q_1^{e_1}, ..., x_t \equiv y_t \mod q_t^{e_t}$$

Pohlig Hellman Algorithm Runtime

Why that runtime?

Pohlig Hellman Algorithm Runtime

Why that runtime?
Step 1, solving each Discrete Log, takes at most as long as it takes to solve that Discrete Log using Shanks, but it's a very small finite group, since our order factored into small primes.

Pohlig Hellman Algorithm Runtime

Why that runtime?
Step 1, solving each Discrete Log, takes at most as long as it takes to solve that Discrete Log using Shanks, but it's a very small finite group, since our order factored into small primes.
In reality, we might be able to get a much smaller runtime than Shanks, which is why we write that step 1 takes:

$$O(Sqi^{e_1} + \ldots + Sq^{e}_{t\ t})$$

Pohlig Hellman Algorithm Runtime

Why that runtime?
Step 1, solving each Discrete Log, takes at most as long as it takes to solve that Discrete Log using Shanks, but it's a very small finite group, since our order factored into small primes.
In reality, we might be able to get a much smaller runtime than Shanks, which is why we write that step 1 takes:

$$O(Sqi^{e_1} + ... + Sq_t^e{}_t)$$

And Step 2 has a neglifible computation time. Solving modular congrunces using C.R.T tajes only $O(logN)$ steps

Pohlig Hellman Algorithm Example

1. Solve $3^x \equiv 22 \mod 31$

Pohlig Hellman Algorithm Example

1. Solve $3^x \equiv 22 \mod 31$
2. Find relatively prime factors of 30, $5 * 6$

Pohlig Hellman Algorithm Example

1. Solve $3^x \equiv 22 \mod 31$
2. Find relatively prime factors of 30, $5 * 6$
3. set first equation using first factor

$$x = 5^0 a_0 + 5^1 a_1 = a_0 + 5a_1$$

4. raise to second factor

$$(3^{a_0+5a_1})^6 \equiv 22^6 \mod 31$$

Pohlig Hellman Algorithm Example

1. Then,

$$3^{6a_0+30a_1} \equiv 22^6 \mod 31$$

Pohlig Hellman Algorithm Example

1. Then,

$$3^{6a_0+30a_1} \equiv 22^6 \mod 31$$

$$3^{6a_0}(3^{a_1})^{30} \equiv 3^{6a_0} * 1 \equiv 8 \mod 31$$

by F.L.T

Pohlig Hellman Algorithm Example

1. Then,

$$3^{6a_0+30a_1} \equiv 22^6 \mod 31$$

$$3^{6a_0}(3^{a_1})^{30} \equiv 3^{6a_0} * 1 \equiv 8 \mod 31$$

by F.L.T

$$(3^6)^{a_0} = 229^{a_0} = 16^{a_0} \equiv 8 \mod 31$$

Pohlig Hellman Algorithm Example

1. Then,

$$3^{6a_0+30a_1} \equiv 22^6 \mod 31$$

$$3^{6a_0}(3^{a_1})^{30} \equiv 3^{6a_0} * 1 \equiv 8 \mod 31$$

by F.L.T

$$(3^6)^{a_0} = 229^{a_0} = 16^{a_0} \equiv 8 \mod 31$$

2. Trial and error/ Shanks gives $16^2 \equiv 8 \mod 31$, so $a_0 = 2$, our first equation is then

$$x = 2 + 5a_1$$

Pohlig Hellman Algorithm Example

1. set second equation using second factor

$$x = 6^0 b_0 + 6^1 b_1 = b_0 + 6 b_1$$

Pohlig Hellman Algorithm Example

1. set second equation using second factor

$$x = 6^0 b_0 + 6^1 b_1 = b_0 + 6b_1$$

2. raise to first factor

$$(3^{b_0+6b_1})^5 \equiv 22^5 \mod 31$$

Pohlig Hellman Algorithm Example

1. Then,

$$3^{5b_0+30b_1} \equiv 22^5 \mod 31$$

Pohlig Hellman Algorithm Example

1. Then,

$$3^{5b_0+30b_1} \equiv 22^5 \mod 31$$

$$3^{5b_0}(3^{b_1})^{30} \equiv 3^{5b_0} * 1 \equiv 6 \mod 31$$

by F.L.T

Pohlig Hellman Algorithm Example

1. Then,

$$3^{5b_0+30b_1} \equiv 22^5 \mod 31$$

$$3^{5b_0}(3^{b_1})^{30} \equiv 3^{5b_0} * 1 \equiv 6 \mod 31$$

by F.L.T

$$(3^5)^{b_0} = 243^{b_0} = 26^{b_0} \equiv 6 \mod 31$$

Pohlig Hellman Algorithm Example

1. Then,

$$3^{5b_0+30b_1} \equiv 22^5 \mod 31$$

$$3^{5b_0}(3^{b_1})^{30} \equiv 3^{5b_0} * 1 \equiv 6 \mod 31$$

by F.L.T

$$(3^5)^{b_0} = 243^{b_0} = 26^{b_0} \equiv 6 \mod 31$$

2. Trial and error/ Shanks gives $26^5 \equiv 6 \mod 31$, so $b_0 = 5$, our second equation is then

$$x = 5 + 6b_1$$

Pohlig Hellman Algorithm Example

1. Then, our two moduar congrunces are

$$x \equiv 2 \mod 5, x \equiv 5 \mod 6$$

3. So $3^{17} \equiv 22 \mod 31$

Pohlig Hellman Algorithm Example

1. Then, our two moduar congrunces are

$$x \equiv 2 \mod 5, x \equiv 5 \mod 6$$

2. so now just use C.R.T to solve :

$$x = 17$$

solves both.

3. So $3^{17} \equiv 22 \mod 31$

Conclusion

Use multiplicative groups of large (at least 4000 bit) prime order to encrypt information!

Conclusion

Use multiplicative groups of large (at least 4000 bit) prime order to encrypt information!
In cryptography, you're always designing against the best known decryption algorithm and its runtime

Conclusion

Use multiplicative groups of large (at least 4000 bit) prime order to encrypt information!

In cryptography, you're always designing against the best known decryption algorithm and its runtime

Longer decryption times with groups defined on elliptic curves