Public Key Crypto: Math Needed and Diffie-Hellman

October 9, 2019

TALK TONIGHT

Capitol One Tech Talk

Wed Sept 18, 6:00-7:15, IRB Center Room 1116

Abstract

Join us to learn more about Tech at Capital One! We'll have mini, interactive tech talks on Machine Learning, Tech Product Demos, and Tech Internship Programs from 6-7:15, followed by networking.

Private-Key Ciphers

What do the following all have in common?

1. Shift Cipher
2. Affine Cipher
3. Vig Cipher
4. General Sub
5. General 2-char sub
6. Matrix Cipher
7. Playfair Cipher
8. Rail Cipher
9. One-time Pad

Private-Key Ciphers

What do the following all have in common?

1. Shift Cipher
2. Affine Cipher
3. Vig Cipher
4. General Sub
5. General 2-char sub
6. Matrix Cipher
7. Playfair Cipher
8. Rail Cipher
9. One-time Pad

Alice and Bob need to meet! (Hence Private Key.)

Private-Key Ciphers

What do the following all have in common?

1. Shift Cipher
2. Affine Cipher
3. Vig Cipher
4. General Sub
5. General 2-char sub
6. Matrix Cipher
7. Playfair Cipher
8. Rail Cipher
9. One-time Pad

Alice and Bob need to meet! (Hence Private Key.)
Can Alice and Bob establish a key without meeting?

Private-Key Ciphers

What do the following all have in common?

1. Shift Cipher
2. Affine Cipher
3. Vig Cipher
4. General Sub
5. General 2-char sub
6. Matrix Cipher
7. Playfair Cipher
8. Rail Cipher
9. One-time Pad

Alice and Bob need to meet! (Hence Private Key.)
Can Alice and Bob establish a key without meeting?
Yes! And that is the key to public-key cryptography.

General Philosophy

A good crypto system is such that:

1. The computational task to encrypt and decrypt is easy.
2. The computational task to crack is hard.

Caveats:

1. Hard to achieve info-theoretic hardness (One-time pad).
2. Hard to achieve comp-hardness. Few problems provably hard.
3. Can use hardness assumptions (e.g. factoring is hard)

Difficulty of Problems Based on Length of Input

How hard is a problem based on the length of the input

Examples

1. SAT on a formula with n vars seems to require $2^{\Omega(n)}$ steps.
2. Polynomial vs Exp time is our notion of easy vs hard.
3. Factoring n can be done in $O(\sqrt{n})$ time: Discuss. Easy!

Difficulty of Problems Based on Length of Input

How hard is a problem based on the length of the input

Examples

1. SAT on a formula with n vars seems to require $2^{\Omega(n)}$ steps.
2. Polynomial vs Exp time is our notion of easy vs hard.
3. Factoring n can be done in $O(\sqrt{n})$ time: Discuss. Easy!
   NO!!: n is of length $\lg n + O(1)$ (henceforth just $\lg n$).
   $\sqrt{n} = 2^{(0.5)\lg n}$. Exponential. Slightly better algs known.

Upshot: For numeric problems length is $\lg n$. Encryption requires:

- Alice and Bob can Enc and Dec in time $\leq (\log n)^{O(1)}$.
- Eve needs time $\geq c^{O(\log n)}$ to crack.

What We Count: We will count arithmetic operations as taking 1 time step. This could be an issue with enormous numbers. Not our problem.

Math Needed for Both Diffie-Hellman and RSA

October 9, 2019

Notation

Let p be a prime.

1. $\mathbb{Z}_p$ is the numbers $\{0, \ldots, p-1\}$ with mod add and mult.
2. $\mathbb{Z}_p^*$ is the numbers $\{1, \ldots, p-1\}$ with mod mult.

Exponentiation Mod p

Exponentiation mod p

Problem: Given a, n, p find $a^n \pmod{p}$

First Attempt

1. $x_0 = a^0 = 1$
2. For $i = 1$ to n, $x_i = ax_{i-1}$.
3. Let $x = x_n \pmod{p}$.
4. Output x.

Is this a good idea?

Exponentiation mod p

Problem: Given a, n, p find $a^n \pmod{p}$

First Attempt

1. $x_0 = a^0 = 1$
2. For $i = 1$ to n, $x_i = ax_{i-1}$.
3. Let $x = x_n \pmod{p}$.
4. Output x.

Is this a good idea? I called it First Attempt, so no.

Exponentiation mod p

Problem: Given a, n, p find $a^n \pmod{p}$

First Attempt

1. $x_0 = a^0 = 1$
2. For $i = 1$ to n, $x_i = ax_{i-1}$.
3. Let $x = x_n \pmod{p}$.
4. Output x.

Is this a good idea? I called it First Attempt, so no.

Discuss How many steps used compute $a^n \pmod{p}$.

Exponentiation mod p

Problem: Given a, n, p find $a^n \pmod{p}$

First Attempt

1. $x_0 = a^0 = 1$
2. For $i = 1$ to n, $x_i = ax_{i-1}$.
3. Let $x = x_n \pmod{p}$.
4. Output x.

Is this a good idea? I called it First Attempt, so no.

Discuss How many steps used compute $a^n \pmod{p}$.

Answer: $\sim n$.

Exponentiation mod p

Problem: Given a, n, p find $a^n \pmod p$

First Attempt

1. $x_0 = a^0 = 1$
2. For $i = 1$ to n, $x_i = ax_{i-1}$.
3. Let $x = x_n \pmod p$.
4. Output x.

Is this a good idea? I called it First Attempt, so no.

Discuss How many steps used compute $a^n \pmod p$.

Answer: $\sim n$.

But it's worse than that. Why?

Exponentiation mod p

Problem: Given a, n, p find $a^n \pmod{p}$

First Attempt

1. $x_0 = a^0 = 1$
2. For $i = 1$ to n, $x_i = ax_{i-1}$.
3. Let $x = x_n \pmod{p}$.
4. Output x.

Is this a good idea? I called it First Attempt, so no.

Discuss How many steps used compute $a^n \pmod{p}$.

Answer: $\sim n$.

But it's worse than that. Why? x gets really large.

Exponentiation mod p

Problem: Given a, n, p find $a^n \pmod{p}$

First Attempt

1. $x_0 = a^0 = 1$
2. For $i = 1$ to n, $x_i = ax_{i-1}$.
3. Let $x = x_n \pmod{p}$.
4. Output x.

Is this a good idea? I called it First Attempt, so no.

Discuss How many steps used compute $a^n \pmod{p}$.

Answer: $\sim n$.

But it's worse than that. Why? x gets really large.

Can mod p every step so x not large. But still takes n steps.

Exponentiation mod p

Example of a Good Algorithm

Want $3^{64} \pmod{101}$. All arithmetic is mod 101.

$x_0 = 3$

$x_1 = x_0^2 \equiv 9$. This is 3^2.

$x_2 = x_1^2 \equiv 9^2 \equiv 81$. This is 3^4.

$x_3 = x_2^2 \equiv 81^2 \equiv 97$. This is 3^8.

$x_4 = x_3^2 \equiv 97^2 \equiv 16$. This is 3^{16}.

$x_5 = x_4^2 \equiv 16^2 \equiv 54$. This is 3^{32}.

$x_6 = x_5^2 \equiv 54^2 \equiv 88$. This is 3^{64}.

So in 6 steps we got the answer!

Exponentiation mod p

Example of a Good Algorithm

Want $3^{64} \pmod{101}$. All arithmetic is mod 101.

$x_0 = 3$

$x_1 = x_0^2 \equiv 9$. This is 3^2.

$x_2 = x_1^2 \equiv 9^2 \equiv 81$. This is 3^4.

$x_3 = x_2^2 \equiv 81^2 \equiv 97$. This is 3^8.

$x_4 = x_3^2 \equiv 97^2 \equiv 16$. This is 3^{16}.

$x_5 = x_4^2 \equiv 16^2 \equiv 54$. This is 3^{32}.

$x_6 = x_5^2 \equiv 54^2 \equiv 88$. This is 3^{64}.

So in 6 steps we got the answer!

Discuss How many steps used compute $a^n \pmod{p}$?

Exponentiation mod p

Example of a Good Algorithm

Want $3^{64} \pmod{101}$. All arithmetic is mod 101.

$x_0 = 3$

$x_1 = x_0^2 \equiv 9$. This is 3^2.

$x_2 = x_1^2 \equiv 9^2 \equiv 81$. This is 3^4.

$x_3 = x_2^2 \equiv 81^2 \equiv 97$. This is 3^8.

$x_4 = x_3^2 \equiv 97^2 \equiv 16$. This is 3^{16}.

$x_5 = x_4^2 \equiv 16^2 \equiv 54$. This is 3^{32}.

$x_6 = x_5^2 \equiv 54^2 \equiv 88$. This is 3^{64}.

So in 6 steps we got the answer!

Discuss How many steps used compute $a^n \pmod{p}$?

Answer: $\sim \lg n$.

Exponentiation mod p

Example of a Good Algorithm

Want $3^{64} \pmod{101}$. All arithmetic is mod 101.

$x_0 = 3$

$x_1 = x_0^2 \equiv 9$. This is 3^2.

$x_2 = x_1^2 \equiv 9^2 \equiv 81$. This is 3^4.

$x_3 = x_2^2 \equiv 81^2 \equiv 97$. This is 3^8.

$x_4 = x_3^2 \equiv 97^2 \equiv 16$. This is 3^{16}.

$x_5 = x_4^2 \equiv 16^2 \equiv 54$. This is 3^{32}.

$x_6 = x_5^2 \equiv 54^2 \equiv 88$. This is 3^{64}.

So in 6 steps we got the answer!

Discuss How many steps used compute $a^n \pmod{p}$?

Answer: $\sim \lg n$.

Discuss How we can generalize to when n is not a power of 2.

A Review of Base 2

Say we want to do $a^n \pmod{p}$.
Let's look carefully at a in binary.
$7 = (111)_2 = 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$. Note $2 = \lfloor \lg 7 \rfloor$
$8 = (1000) = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$. Note $3 = \lfloor \lg 8 \rfloor$
$9 = (1001) = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$. Note $3 = \lfloor \lg 9 \rfloor$
Upshot: If write n as a sum of powers of 2 with 0,1 coefficients then n is of the form:

$$n = n_L 2^L + \cdots + n_1 2^1 + n_0 2^0 = \sum_{i=0}^{L} n_i 2^i$$

Where $L = \lfloor \lg(n) \rfloor$ and $n_i \in \{0, 1\}$.
Note that L is one less than the number of bits needed for n.

Repeated Squaring Algorithm

All arithmetic is mod p.

1. Input (a, n, p)
2. Convert n to base 2: $n = \sum_{i=0}^{L} n_i 2^i$. ($L$ is $\lfloor \lg(n) \rfloor$)
3. $x_0 = a$
4. For $i = 1$ to L, $x_i = x_{i-1}^2$ (Note: $x_i = a^{2^i}$.)
5. (Now have $a^{n_0 2^0}, \ldots, a^{n_L 2^L}$) Answer is $a^{n_0 2^0} \times \cdots \times a^{n_L 2^L}$

Number of operations:

Number of $\times$'s in step 4: $\leq \lfloor \lg(n) \rfloor \leq \lg(n)$

Repeated Squaring Algorithm

All arithmetic is mod p.

1. Input (a, n, p)
2. Convert n to base 2: $n = \sum_{i=0}^{L} n_i 2^i$. ($L$ is $\lfloor \lg(n) \rfloor$)
3. $x_0 = a$
4. For $i = 1$ to L, $x_i = x_{i-1}^2$ (Note: $x_i = a^{2^i}$.)
5. (Now have $a^{n_0 2^0}, \ldots, a^{n_L 2^L}$) Answer is $a^{n_0 2^0} \times \cdots \times a^{n_L 2^L}$

Number of operations:
Number of $\times$'s in step 4: $\leq \lfloor \lg(n) \rfloor \leq \lg(n)$
Number of $\times$'s in step 5: $\leq L = \lfloor \lg(n) \rfloor \leq \lg(n)$

Repeated Squaring Algorithm

All arithmetic is mod p.

1. Input (a, n, p)
2. Convert n to base 2: $n = \sum_{i=0}^{L} n_i 2^i$. ($L$ is $\lfloor \lg(n) \rfloor$)
3. $x_0 = a$
4. For $i = 1$ to L, $x_i = x_{i-1}^2$ (Note: $x_i = a^{2^i}$.)
5. (Now have $a^{n_0 2^0}, \ldots, a^{n_L 2^L}$) Answer is $a^{n_0 2^0} \times \cdots \times a^{n_L 2^L}$

Number of operations:

Number of $\times$'s in step 4: $\leq \lfloor \lg(n) \rfloor \leq \lg(n)$

Number of $\times$'s in step 5: $\leq L = \lfloor \lg(n) \rfloor \leq \lg(n)$

Total number of $\times$'s: $\leq 2\lg(n)$.

Repeated Squaring Algorithm

All arithmetic is mod p.

1. Input (a, n, p)
2. Convert n to base 2: $n = \sum_{i=0}^{L} n_i 2^i$. ($L$ is $\lfloor \lg(n) \rfloor$)
3. $x_0 = a$
4. For $i = 1$ to L, $x_i = x_{i-1}^2$ (Note: $x_i = a^{2^i}$.)
5. (Now have $a^{n_0 2^0}, \ldots, a^{n_L 2^L}$) Answer is $a^{n_0 2^0} \times \cdots \times a^{n_L 2^L}$

Number of operations:
Number of $\times$'s in step 4: $\le \lfloor \lg(n) \rfloor \le \lg(n)$
Number of $\times$'s in step 5: $\le L = \lfloor \lg(n) \rfloor \le \lg(n)$
Total number of $\times$'s: $\le 2\lg(n)$.
More refined: $\lg(n) +$ number of 1's in binary rep of n -1

Repeated Squaring Algorithm

All arithmetic is mod p.

1. Input (a, n, p)
2. Convert n to base 2: $n = \sum_{i=0}^{L} n_i 2^i$. ($L$ is $\lfloor \lg(n) \rfloor$)
3. $x_0 = a$
4. For $i = 1$ to L, $x_i = x_{i-1}^2$ (Note: $x_i = a^{2^i}$.)
5. (Now have $a^{n_0 2^0}, \ldots, a^{n_L 2^L}$) Answer is $a^{n_0 2^0} \times \cdots \times a^{n_L 2^L}$

Number of operations:
Number of $\times$'s in step 4: $\leq \lfloor \lg(n) \rfloor \leq \lg(n)$
Number of $\times$'s in step 5: $\leq L = \lfloor \lg(n) \rfloor \leq \lg(n)$
Total number of $\times$'s: $\leq 2\lg(n)$.
More refined: $\lg(n) +$ number of 1's in binary rep of n -1
Example on next page

Example of Exponentiation: $17^{265} \pmod{101}$

$$265 = 2^8 + 2^3 + 2^0$$

$17^{2^0} \equiv 17$ (0 steps)
$17^{2^1} \equiv 17^2 \equiv 87$ (1 step)
$17^{2^2} \equiv 87^2 \equiv 95$ (1 step)
$17^{2^3} \equiv 95^2 \equiv 36$ (1 step)
$17^{2^4} \equiv 36^2 \equiv 84$ (1 step)
$17^{2^5} \equiv 84^2 \equiv 87$ (1 step)
$17^{2^6} \equiv 87^2 \equiv 95$ (1 step)
$17^{2^7} \equiv 95^2 \equiv 36$ (1 step)
$17^{2^8} \equiv 36^2 \equiv 84$ (1 step)
This took $8 \sim \lg(265)$ multiplications so far.

Example of Exponentiation: $17^{265} \pmod{101}$

$$265 = 2^8 + 2^3 + 2^0$$

$17^{2^0} \equiv 17$ (0 steps)
$17^{2^1} \equiv 17^2 \equiv 87$ (1 step)
$17^{2^2} \equiv 87^2 \equiv 95$ (1 step)
$17^{2^3} \equiv 95^2 \equiv 36$ (1 step)
$17^{2^4} \equiv 36^2 \equiv 84$ (1 step)
$17^{2^5} \equiv 84^2 \equiv 87$ (1 step)
$17^{2^6} \equiv 87^2 \equiv 95$ (1 step)
$17^{2^7} \equiv 95^2 \equiv 36$ (1 step)
$17^{2^8} \equiv 36^2 \equiv 84$ (1 step)
This took $8 \sim \lg(265)$ multiplications so far.
The next step takes only two:

$$17^{265} \equiv 17^{2^8} \times 17^{2^3} \times 17^{2^0} \equiv 84 \times 36 \times 17 \equiv 100$$

Example of Exponentiation: $17^{265} \pmod{101}$

$$265 = 2^8 + 2^3 + 2^0$$

$17^{2^0} \equiv 17$ (0 steps)
$17^{2^1} \equiv 17^2 \equiv 87$ (1 step)
$17^{2^2} \equiv 87^2 \equiv 95$ (1 step)
$17^{2^3} \equiv 95^2 \equiv 36$ (1 step)
$17^{2^4} \equiv 36^2 \equiv 84$ (1 step)
$17^{2^5} \equiv 84^2 \equiv 87$ (1 step)
$17^{2^6} \equiv 87^2 \equiv 95$ (1 step)
$17^{2^7} \equiv 95^2 \equiv 36$ (1 step)
$17^{2^8} \equiv 36^2 \equiv 84$ (1 step)
This took $8 \sim \lg(265)$ multiplications so far.
The next step takes only two:

$$17^{265} \equiv 17^{2^8} \times 17^{2^3} \times 17^{2^0} \equiv 84 \times 36 \times 17 \equiv 100$$

Point: Step 2 took $\ll \lg(265)$ steps since base-2 rep had few 1's.

Finding Generators

Generators mod p

Let's take powers of 3 mod 7. All arithmetic is mod 7.

$3^1 \equiv 3$
$3^2 \equiv 3 \times 3^1 \equiv 9 \equiv 2$
$3^3 \equiv 3 \times 3^2 \equiv 3 \times 2 \equiv 6$
$3^4 \equiv 3 \times 3^3 \equiv 3 \times 6 \equiv 18 \equiv 4$
$3^5 \equiv 3 \times 3^4 \equiv 3 \times 4 \equiv 12 \equiv 5$
$3^6 \equiv 3 \times 3^5 \equiv 3 \times 5 \equiv 15 \equiv 1$

$$\{3^1, 3^2, 3^3, 3^4, 3^5, 3^6\} = \{1, 2, 3, 4, 5, 6\} \text{ Not in order}$$

Generators mod p

Let's take powers of 3 mod 7. All arithmetic is mod 7.

$3^1 \equiv 3$

$3^2 \equiv 3 \times 3^1 \equiv 9 \equiv 2$

$3^3 \equiv 3 \times 3^2 \equiv 3 \times 2 \equiv 6$

$3^4 \equiv 3 \times 3^3 \equiv 3 \times 6 \equiv 18 \equiv 4$

$3^5 \equiv 3 \times 3^4 \equiv 3 \times 4 \equiv 12 \equiv 5$

$3^6 \equiv 3 \times 3^5 \equiv 3 \times 5 \equiv 15 \equiv 1$

$$\{3^1, 3^2, 3^3, 3^4, 3^5, 3^6\} = \{1, 2, 3, 4, 5, 6\} \text{ Not in order}$$

3 is a generator for $\mathbb{Z}_7^*$.

Generators mod p

Let's take powers of 3 mod 7. All arithmetic is mod 7.
$3^1 \equiv 3$
$3^2 \equiv 3 \times 3^1 \equiv 9 \equiv 2$
$3^3 \equiv 3 \times 3^2 \equiv 3 \times 2 \equiv 6$
$3^4 \equiv 3 \times 3^3 \equiv 3 \times 6 \equiv 18 \equiv 4$
$3^5 \equiv 3 \times 3^4 \equiv 3 \times 4 \equiv 12 \equiv 5$
$3^6 \equiv 3 \times 3^5 \equiv 3 \times 5 \equiv 15 \equiv 1$

$$\{3^1, 3^2, 3^3, 3^4, 3^5, 3^6\} = \{1, 2, 3, 4, 5, 6\} \text{ Not in order}$$

3 is a generator for $\mathbb{Z}_7^*$.
Definition: If p is a prime and $\{g^1, \ldots, g^{p-1}\} = \{1, \ldots, p-1\}$ then g is a generator for $\mathbb{Z}_p^*$.

Discrete Log-Example

Fact: 3 is a generator mod 101. All arithmetic is mod 101.
Discuss the following with your neighbor:

1. Find x such that $3^x \equiv 81$.

Discrete Log-Example

Fact: 3 is a generator mod 101. All arithmetic is mod 101.
Discuss the following with your neighbor:

1. Find x such that $3^x \equiv 81$. $x = 4$ obv works.

Discrete Log-Example

Fact: 3 is a generator mod 101. All arithmetic is mod 101.
Discuss the following with your neighbor:

1. Find x such that $3^x \equiv 81$. $x = 4$ obv works.
2. Find x such that $3^x \equiv 92$.

Discrete Log-Example

Fact: 3 is a generator mod 101. All arithmetic is mod 101.
Discuss the following with your neighbor:

1. Find x such that $3^x \equiv 81$. $x = 4$ obv works.
2. Find x such that $3^x \equiv 92$. Try computing $3^1, 3^2, \ldots,$ until you get 92. Might take ~ 100 steps.

Discrete Log-Example

Fact: 3 is a generator mod 101. All arithmetic is mod 101.
Discuss the following with your neighbor:

1. Find x such that $3^x \equiv 81$. $x = 4$ obv works.
2. Find x such that $3^x \equiv 92$. Try computing $3^1, 3^2, \ldots,$ until you get 92. Might take ~ 100 steps. Shortcut?

Discrete Log-Example

Fact: 3 is a generator mod 101. All arithmetic is mod 101.
Discuss the following with your neighbor:

1. Find x such that $3^x \equiv 81$. $x = 4$ obv works.
2. Find x such that $3^x \equiv 92$. Try computing $3^1, 3^2, \ldots,$ until you get 92. Might take ~ 100 steps. Shortcut?
3. Find x such that $3^x \equiv 93$.

Discrete Log-Example

Fact: 3 is a generator mod 101. All arithmetic is mod 101.
Discuss the following with your neighbor:

1. Find x such that $3^x \equiv 81$. $x = 4$ obv works.
2. Find x such that $3^x \equiv 92$. Try computing $3^1, 3^2, \ldots,$ until you get 92. Might take ~ 100 steps. Shortcut?
3. Find x such that $3^x \equiv 93$. Try computing $3^1, 3^2, \ldots,$ until you get 93. Might take ~ 100 steps.

Discrete Log-Example

Fact: 3 is a generator mod 101. All arithmetic is mod 101.
Discuss the following with your neighbor:

1. Find x such that $3^x \equiv 81$. $x = 4$ obv works.
2. Find x such that $3^x \equiv 92$. Try computing $3^1, 3^2, \ldots,$ until you get 92. Might take ~ 100 steps. Shortcut?
3. Find x such that $3^x \equiv 93$. Try computing $3^1, 3^2, \ldots,$ until you get 93. Might take ~ 100 steps. Shortcut?

The second and third problem look hard. Are they? VOTE: Both hard, both easy, one of each, unknown to science.

Discrete Log-Example

Fact: 3 is a generator mod 101. All arithmetic is mod 101.
Discuss the following with your neighbor:

1. Find x such that $3^x \equiv 81$. $x = 4$ obv works.
2. Find x such that $3^x \equiv 92$. Try computing $3^1, 3^2, \ldots,$ until you get 92. Might take ~ 100 steps. Shortcut?
3. Find x such that $3^x \equiv 93$. Try computing $3^1, 3^2, \ldots,$ until you get 93. Might take ~ 100 steps. Shortcut?

The second and third problem look hard. Are they? VOTE: Both hard, both easy, one of each, unknown to science.

$3^x \equiv 92$ easy. $3^x \equiv 93$ Not known how hard.

Discrete Log-Example: $3^x \equiv 92 \pmod{101}$

Fact: 3 is a generator mod 101. All arithmetic is mod 101.
Find x such that $3^x \equiv 92$. Easy!

1. $92 \equiv 101 - 9 \equiv (-1)(9) \equiv (-1)3^2$.
2. $3^{50} \equiv -1$ (WHAT! Really?)
3. $92 \equiv 3^{50} \times 3^2 \equiv 3^{52}$. So $x = 52$ works.

Generalize:

1. If g is a generator of $\mathbb{Z}_p^*$ then $g^{(p-1)/2} \equiv p - 1 \equiv -1$.
2. So finding x such that $g^x \equiv p - g^a \equiv -g^a$ is easy:

$$x = \frac{p-1}{2} + a : \qquad g^{\frac{p-1}{2}+a} = g^{\frac{p-1}{2}} g^a \equiv -g^a$$

Discrete Log-Example: $3^x \equiv 93 \pmod{101}$

Fact: 3 is a generator mod 101. All arithmetic is mod 101.
Is there a trick for $g^x \equiv 93 \pmod{101}$? Not that I know of.

What is known about complexity of discrete log?
Given g, a, p find x such that $g^x \equiv a \pmod{p}$.

1. Naive algorithm is $O(p)$ time.
2. Exists a $O(\sqrt{p})$ Time, $O(\sqrt{p})$ space alg. Space makes it not useable.
3. Exists a $O(\sqrt{p})$ Time, $(\log p)^{O(1)}$ space alg. Useable!
4. Not much progress on theory front since 1985.
5. DL is in QuantumP.

My Opinion on DL. Also applies to Factoring

1. Fact: DL in in QuantumP.
2. Opinion: Quantum computers that can do DL fast won't happen

My Opinion on DL. Also applies to Factoring

1. Fact: DL in in QuantumP.
2. Opinion: Quantum computers that can do DL fast won't happen in my lifetime.

My Opinion on DL. Also applies to Factoring

1. Fact: DL in in QuantumP.
2. Opinion: Quantum computers that can do DL fast won't happen in my lifetime. In your lifetime.

My Opinion on DL. Also applies to Factoring

1. Fact: DL in in QuantumP.
2. Opinion: Quantum computers that can do DL fast won't happen in my lifetime. In your lifetime. Ever.

My Opinion on DL. Also applies to Factoring

1. Fact: DL in in QuantumP.
2. Opinion: Quantum computers that can do DL fast won't happen in my lifetime. In your lifetime. Ever.
3. Fact: Classical algorithms that are *better* than naive, using hard number theory, have been discovered and implemented. Still exponential but low constants, possibly good in practice. Some are amenable to parallelism.

My Opinion on DL. Also applies to Factoring

1. Fact: DL in in QuantumP.
2. Opinion: Quantum computers that can do DL fast won't happen in my lifetime. In your lifetime. Ever.
3. Fact: Classical algorithms that are *better* than naive, using hard number theory, have been discovered and implemented. Still exponential but low constants, possibly good in practice. Some are amenable to parallelism.
4. Opinion: The biggest threat to crypto is from hard math combined with special purpose parallel hardware.

My Opinion on DL. Also applies to Factoring

1. Fact: DL in in QuantumP.
2. Opinion: Quantum computers that can do DL fast won't happen in my lifetime. In your lifetime. Ever.
3. Fact: Classical algorithms that are *better* than naive, using hard number theory, have been discovered and implemented. Still exponential but low constants, possibly good in practice. Some are amenable to parallelism.
4. Opinion: The biggest threat to crypto is from hard math combined with special purpose parallel hardware.
5. Fact: If computers do DL much better (e.g., $O(n^{1/10})$) then humans need to triple the length of their numbers. Still, Eve has made Alice and Bob work harder.

My Opinion on DL. Also applies to Factoring

1. Fact: DL in in QuantumP.
2. Opinion: Quantum computers that can do DL fast won't happen in my lifetime. In your lifetime. Ever.
3. Fact: Classical algorithms that are *better* than naive, using hard number theory, have been discovered and implemented. Still exponential but low constants, possibly good in practice. Some are amenable to parallelism.
4. Opinion: The biggest threat to crypto is from hard math combined with special purpose parallel hardware.
5. Fact: If computers do DL much better (e.g., $O(n^{1/10})$) then humans need to triple the length of their numbers. Still, Eve has made Alice and Bob work harder.
6. Opinion: When people really really need to up their parameters

My Opinion on DL. Also applies to Factoring

1. Fact: DL in in QuantumP.
2. Opinion: Quantum computers that can do DL fast won't happen in my lifetime. In your lifetime. Ever.
3. Fact: Classical algorithms that are *better* than naive, using hard number theory, have been discovered and implemented. Still exponential but low constants, possibly good in practice. Some are amenable to parallelism.
4. Opinion: The biggest threat to crypto is from hard math combined with special purpose parallel hardware.
5. Fact: If computers do DL much better (e.g., $O(n^{1/10})$) then humans need to triple the length of their numbers. Still, Eve has made Alice and Bob work harder.
6. Opinion: When people really really need to up their parameters they don't.

My Opinion on DL. Also applies to Factoring

1. Fact: DL in in QuantumP.
2. Opinion: Quantum computers that can do DL fast won't happen in my lifetime. In your lifetime. Ever.
3. Fact: Classical algorithms that are *better* than naive, using hard number theory, have been discovered and implemented. Still exponential but low constants, possibly good in practice. Some are amenable to parallelism.
4. Opinion: The biggest threat to crypto is from hard math combined with special purpose parallel hardware.
5. Fact: If computers do DL much better (e.g., $O(n^{1/10})$) then humans need to triple the length of their numbers. Still, Eve has made Alice and Bob work harder.
6. Opinion: When people really really need to up their parameters they don't.They say

   It won't happen to me Until it does.

Discrete Log-General

Definition Let p be a prime and g be a generator mod p.
The Discrete Log Problem is:
Given $y \in \{1, \ldots, p\}$, find x such that $g^x \equiv y \pmod{p}$. We call this $DL_{p,g}(y)$.

1. If g is small then $DL(g^a)$ or $DL(p - g^a)$ might be easy.
   Example: $DL_{1009,7}(49) = 2$ since $7^2 \equiv 49 \pmod{1009}$.
   Example: $DL_{1009,7}(1009 - 49) = 506$ since $7^{504}7^2 \equiv -7^2 \equiv 1009 - 49 \pmod{1009}$.
2. If $g, a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$ then problem suspected hard.
3. Tradeoff: By restricting a we are cutting down search space for Eve. Even so, in this case we need to since she REALLY can recognize when DL is easy.

Consider What We Already Have Here

- Exponentiation is Easy.
- Discrete Log is thought to be Hard.

Consider What We Already Have Here

- Exponentiation is Easy.
- Discrete Log is thought to be Hard.

Can we come up with a crypto system where Alice and Bob do Exponentiation to encrypt and decrypt, while Eve has to do Discrete Log to crack it?

Consider What We Already Have Here

- Exponentiation is Easy.
- Discrete Log is thought to be Hard.

Can we come up with a crypto system where Alice and Bob do Exponentiation to encrypt and decrypt, while Eve has to do Discrete Log to crack it?

No. But we'll come close.

Convention

For the rest of the slides on Diffie-Hellman Key Exchange there will always be a prime p that we are considering.

ALL arithmetic done from that point on is $\pmod p$.

ALL numbers are in $\{1, \ldots, p-1\}$.

Finding Gens: First Attempt

Given prime p, find a gen for $\mathbb{Z}_p^*$

1. Input p
2. For $g = \left\lceil \frac{p}{3} \right\rceil$ to $\left\lfloor \frac{2p}{3} \right\rfloor$

   Compute $g^1, g^2, \ldots, g^{p-1}$ until either hit a repeat or finish. If repeats then g is NOT a generator, so goto the next g. If finishes then output g and stop.

PRO: many g's are gen's so $O(1)$ iterations.

Finding Gens: First Attempt

Given prime p, find a gen for $\mathbb{Z}_p^*$

1. Input p
2. For $g = \lceil \frac{p}{3} \rceil$ to $\left\lfloor \frac{2p}{3} \right\rfloor$

   Compute $g^1, g^2, \ldots, g^{p-1}$ until either hit a repeat or finish. If repeats then g is NOT a generator, so goto the next g. If finishes then output g and stop.

PRO: many g's are gen's so $O(1)$ iterations.
CON: Computing $g^1, \ldots, g^{p-1}$ is $O(p)$ operations.

Finding Gens: First Attempt

Given prime p, find a gen for $\mathbb{Z}_p^*$

1. Input p
2. For $g = \lceil \frac{p}{3} \rceil$ to $\lfloor \frac{2p}{3} \rfloor$
   Compute $g^1, g^2, \ldots, g^{p-1}$ until either hit a repeat or finish. If repeats then g is NOT a generator, so goto the next g. If finishes then output g and stop.

PRO: many g's are gen's so $O(1)$ iterations.
CON: Computing $g^1, \ldots, g^{p-1}$ is $O(p)$ operations.
Bad! Recall need poly on $\log p$.

Finding Gens: Second Attempt

Theorem: If g is not a generator then there exists x that (1) x divides $p-1$, (2) $x \neq p-1$, and (3) $g^x \equiv 1$.

Given prime p, find a gen for $\mathbb{Z}_p^*$

1. Input p
2. Factor $p-1$. Let F be the set of its factors except $p-1$.
3. For $g = \lceil \frac{p}{3} \rceil$ to $\left\lfloor \frac{2p}{3} \right\rfloor$
   Compute g^x for all $x \in F$. If any $= 1$ then g not generator.
   If none are 1 then output g and stop.

Is this a good algorithm?

Finding Gens: Second Attempt

Theorem: If g is not a generator then there exists x that (1) x divides $p-1$, (2) $x \neq p-1$, and (3) $g^x \equiv 1$.

Given prime p, find a gen for $\mathbb{Z}_p^*$

1. Input p
2. Factor $p-1$. Let F be the set of its factors except $p-1$.
3. For $g = \lceil \frac{p}{3} \rceil$ to $\left\lfloor \frac{2p}{3} \right\rfloor$
   Compute g^x for all $x \in F$. If any $= 1$ then g not generator.
   If none are 1 then output g and stop.

Is this a good algorithm?
PRO: As noted before, $O(1)$ iterations.

Finding Gens: Second Attempt

Theorem: If g is not a generator then there exists x that (1) x divides $p-1$, (2) $x \neq p-1$, and (3) $g^x \equiv 1$.

Given prime p, find a gen for $\mathbb{Z}_p^*$

1. Input p
2. Factor $p-1$. Let F be the set of its factors except $p-1$.
3. For $g = \lceil \frac{p}{3} \rceil$ to $\left\lfloor \frac{2p}{3} \right\rfloor$
   Compute g^x for all $x \in F$. If any $= 1$ then g not generator.
   If none are 1 then output g and stop.

Is this a good algorithm?
PRO: As noted before, $O(1)$ iterations.
PRO: Every iter – $O(|F|(\log p))$ ops. $|F|$ might be huge! So no good. But wait for next slide. . ..

Finding Gens: Second Attempt

Theorem: If g is not a generator then there exists x that (1) x divides $p-1$, (2) $x \neq p-1$, and (3) $g^x \equiv 1$.

Given prime p, find a gen for $\mathbb{Z}_p^*$

1. Input p
2. Factor $p-1$. Let F be the set of its factors except $p-1$.
3. For $g = \lceil \frac{p}{3} \rceil$ to $\left\lfloor \frac{2p}{3} \right\rfloor$
   Compute g^x for all $x \in F$. If any $=1$ then g not generator.
   If none are 1 then output g and stop.

Is this a good algorithm?
PRO: As noted before, $O(1)$ iterations.
PRO: Every iter – $O(|F|(\log p))$ ops. $|F|$ might be huge! So no good. But wait for next slide. . ..
BIG CON: Factoring $p-1$? Really? Darn!

Finding Gens: Prep for Third Attempt

Example: $p = 1009$ which is prime. All math is mod 1009.

$$p - 1 = 1008 = 2^4 \times 3^2 \times 7^1 \text{ which has } 5 \times 3 \times 2 = 30 \text{ factors}$$

Finding Gens: Prep for Third Attempt

Example: $p = 1009$ which is prime. All math is mod 1009.

$$p - 1 = 1008 = 2^4 \times 3^2 \times 7^1 \text{ which has } 5 \times 3 \times 2 = 30 \text{ factors}$$

Don't count 1008, so 29 factors. F is set of 29 factors.

Finding Gens: Prep for Third Attempt

Example: $p = 1009$ which is prime. All math is mod 1009.

$$p - 1 = 1008 = 2^4 \times 3^2 \times 7^1 \text{ which has } 5 \times 3 \times 2 = 30 \text{ factors}$$

Don't count 1008, so 29 factors. F is set of 29 factors.
Last Page Method: $(\forall x \in F)$ compute g^x. If any are 1 then g NOT a gen. 29 x's to check! Can we shorten? Discuss.

Finding Gens: Prep for Third Attempt

Example: $p = 1009$ which is prime. All math is mod 1009.

$$p - 1 = 1008 = 2^4 \times 3^2 \times 7^1 \text{ which has } 5 \times 3 \times 2 = 30 \text{ factors}$$

Don't count 1008, so 29 factors. F is set of 29 factors.
Last Page Method: $(\forall x \in F)$ compute g^x. If any are 1 then g NOT a gen. 29 x's to check! Can we shorten? Discuss.

Thought Experiment: Want to test if g is a generator. We find $g^{2^3 \times 3^1} = 1$ so g is not a generator. Note that $g^{2^3 \times 3^2 \times 7^1} = 1$.

Finding Gens: Prep for Third Attempt

Example: $p = 1009$ which is prime. All math is mod 1009.

$$p - 1 = 1008 = 2^4 \times 3^2 \times 7^1 \text{ which has } 5 \times 3 \times 2 = 30 \text{ factors}$$

Don't count 1008, so 29 factors. F is set of 29 factors.
Last Page Method: $(\forall x \in F)$ compute g^x. If any are 1 then g NOT a gen. 29 x's to check! Can we shorten? Discuss.

Thought Experiment: Want to test if g is a generator. We find $g^{2^3 \times 3^1} = 1$ so g is not a generator. Note that $g^{2^3 \times 3^2 \times 7^1} = 1$.
Key: Assume $g^{2^a 3^b 7^c} = 1$ with $(a \le 3) \vee (b \le 1) \vee (c \le 0)$.

- If $a \le 3$ then $g^{2^3 3^2 7^1} = 1$.
- If $b \le 1$ then $g^{2^4 3^1 7^1} = 1$.
- If $c \le 0$ then $g^{2^4 3^2 7^0} = 1$.

So, need only test those THREE values of x.

Finding Gens: Theorem needed for Third Attempt

Theorem

Let $p - 1 = p_1^{a_1} \cdots p_n^{a_n}$ *Let* $g \in \mathbb{Z}_p^*$. *Then* g *is not a gen IFF*

- $g^{(p_1^{a_1} \cdots p_n^{a_n})/p_1} = 1$ *OR*
- $g^{(p_1^{a_1} \cdots p_n^{a_n})/p_2} = 1$ *OR*
- $\vdots$
- $g^{(p_1^{a_1} \cdots p_n^{a_n})/p_n} = 1.$

Finding Gens: Theorem needed for Third Attempt

Theorem

Let $p-1 = p_1^{a_1} \cdots p_n^{a_n}$ *Let* $g \in \mathbb{Z}_p^*$. *Then* g *is not a gen IFF*

- $g^{(p_1^{a_1} \cdots p_n^{a_n})/p_1} = 1$ *OR*
- $g^{(p_1^{a_1} \cdots p_n^{a_n})/p_2} = 1$ *OR*
- $\vdots$
- $g^{(p_1^{a_1} \cdots p_n^{a_n})/p_n} = 1$.

Definition

A number of the form $(p_1^{a_1} \cdots p_n^{a_n})/p_i$ is called a maximal factor of $p-1$. Maxfac for short.

Finding Gens: Theorem needed for Third Attempt

Theorem

Let $p-1 = p_1^{a_1} \cdots p_n^{a_n}$ *Let* $g \in \mathbb{Z}_p^*$. *Then* g *is not a gen IFF*

- $g^{(p_1^{a_1} \cdots p_n^{a_n})/p_1} = 1$ *OR*
- $g^{(p_1^{a_1} \cdots p_n^{a_n})/p_2} = 1$ *OR*
- $\vdots$
- $g^{(p_1^{a_1} \cdots p_n^{a_n})/p_n} = 1.$

Definition

A number of the form $(p_1^{a_1} \cdots p_n^{a_n})/p_i$ is called a maximal factor of $p-1$. Maxfac for short.

How many maxfac's are there as a function of p? Discuss.

Finding Gens: Theorem needed for Third Attempt

Theorem

Let $p - 1 = p_1^{a_1} \cdots p_n^{a_n}$ *Let* $g \in \mathbb{Z}_p^*$. *Then* g *is not a gen IFF*

- $g^{(p_1^{a_1} \cdots p_n^{a_n})/p_1} = 1$ *OR*
- $g^{(p_1^{a_1} \cdots p_n^{a_n})/p_2} = 1$ *OR*
- $\vdots$
- $g^{(p_1^{a_1} \cdots p_n^{a_n})/p_n} = 1.$

Definition

A number of the form $(p_1^{a_1} \cdots p_n^{a_n})/p_i$ is called a maximal factor of $p - 1$. Maxfac for short.

How many maxfac's are there as a function of p? Discuss.
The number of maxfacs is maximized when exp are all 1.
$p - 1 = p_1 \cdots p_n \geq 2^n$, so $\lg(p - 1) \geq n$, so $n \leq \lg(p - 1) \leq \lg(p)$.

Finding Gens: Third Attempt

Theorem: If g is not a generator then there exists x that (1) x is a maxfac of $p-1$ and (2) $g^x = 1$.

Given prime p, find a gen for $\mathbb{Z}_p^*$

1. Input p
2. Factor $p-1$. Let MF be the set of its maxfacs.
3. For $g = \lceil \frac{p}{3} \rceil$ to $\left\lfloor \frac{2p}{3} \right\rfloor$
   Compute g^x for all $x \in MF$. If any $= 1$ then g not generator. If none are 1 then output g and stop.

Is this a good algorithm?

Finding Gens: Third Attempt

Theorem: If g is not a generator then there exists x that (1) x is a maxfac of $p-1$ and (2) $g^x = 1$.

Given prime p, find a gen for $\mathbb{Z}_p^*$

1. Input p
2. Factor $p-1$. Let MF be the set of its maxfacs.
3. For $g = \lceil \frac{p}{3} \rceil$ to $\left\lfloor \frac{2p}{3} \right\rfloor$
   Compute g^x for all $x \in MF$. If any $= 1$ then g not generator. If none are 1 then output g and stop.

Is this a good algorithm?
PRO: As noted before, $O(1)$ iterations.

Finding Gens: Third Attempt

Theorem: If g is not a generator then there exists x that (1) x is a maxfac of $p-1$ and (2) $g^x = 1$.

Given prime p, find a gen for $\mathbb{Z}_p^*$

1. Input p
2. Factor $p-1$. Let MF be the set of its maxfacs.
3. For $g = \lceil \frac{p}{3} \rceil$ to $\left\lfloor \frac{2p}{3} \right\rfloor$
   Compute g^x for all $x \in MF$. If any $= 1$ then g not generator. If none are 1 then output g and stop.

Is this a good algorithm?
PRO: As noted before, $O(1)$ iterations.
PRO: Every iter – $O(|MF|(\log p))$ ops. $|MF| = O(\log p)$. GREAT, this improves over Attempt 2.

Finding Gens: Third Attempt

Theorem: If g is not a generator then there exists x that (1) x is a maxfac of $p-1$ and (2) $g^x = 1$.

Given prime p, find a gen for $\mathbb{Z}_p^*$

1. Input p
2. Factor $p-1$. Let MF be the set of its maxfacs.
3. For $g = \left\lceil \frac{p}{3} \right\rceil$ to $\left\lfloor \frac{2p}{3} \right\rfloor$
   Compute g^x for all $x \in MF$. If any $= 1$ then g not generator. If none are 1 then output g and stop.

Is this a good algorithm?
PRO: As noted before, $O(1)$ iterations.
PRO: Every iter – $O(|MF|(\log p))$ ops. $|MF| = O(\log p)$. GREAT, this improves over Attempt 2.
BIG CON: We still need to factor $p-1$? Really? Darn!
Now what? See the next lecture for the exciting conclusion!