BILL, RECORD LECTURE!!!!

BILL RECORD LECTURE!!!

Math Needed for Both Diffie-Hellman and RSA

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.

Convention By **prime** we will always mean a large prime, so in particular, NOT 2. Hence we can assume $\frac{p-1}{2}$ is in $\mathbb{N}$.

Repeated Squaring Algorithm

All math is mod p.

Repeated Squaring Algorithm

All math is mod p.

1. Input (a, n, p).

Repeated Squaring Algorithm

All math 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 $\lfloor \lg(n) \rfloor$)

Repeated Squaring Algorithm

All math 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. $x_0 = a$.

Repeated Squaring Algorithm

All math 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, $x_i = x_{i-1}^2$

Repeated Squaring Algorithm

All math 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, $x_i = x_{i-1}^2$
5. (Now have $a^{n_0 2^0}, \ldots, a^{n_L 2^L}$) Answer is $a^{n_0 2^0} \times \cdots \times a^{n_L 2^L}$

Repeated Squaring Algorithm

All math 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, $x_i = x_{i-1}^2$
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:

Repeated Squaring Algorithm

All math 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, $x_i = x_{i-1}^2$
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 **MULTS** in step 4: $\leq \lfloor \lg(n) \rfloor \leq \lg(n)$

Repeated Squaring Algorithm

All math 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, $x_i = x_{i-1}^2$
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 **MULTS** in step 4: $\leq \lfloor \lg(n) \rfloor \leq \lg(n)$

Number of **MULTS** in step 5: $\leq L = \lfloor \lg(n) \rfloor \leq \lg(n)$

Repeated Squaring Algorithm

All math 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, $x_i = x_{i-1}^2$
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 **MULTS** in step 4: $\le \lfloor \lg(n) \rfloor \le \lg(n)$

Number of **MULTS** in step 5: $\le L = \lfloor \lg(n) \rfloor \le \lg(n)$

Total number of **MULTS** $\le 2\lg(n)$.

Repeated Squaring Algorithm

All math 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, $x_i = x_{i-1}^2$
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 **MULTS** in step 4: $\le \lfloor \lg(n) \rfloor \le \lg(n)$
Number of **MULTS** in step 5: $\le L = \lfloor \lg(n) \rfloor \le \lg(n)$
Total number of **MULTS** $\le 2\lg(n)$.
More refined: $\lg(n) +$ (number of 1's in binary rep of n) -1

Repeated Squaring Algorithm

All math 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, $x_i = x_{i-1}^2$
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 **MULTS** in step 4: $\leq \lfloor \lg(n) \rfloor \leq \lg(n)$
Number of **MULTS** in step 5: $\leq L = \lfloor \lg(n) \rfloor \leq \lg(n)$
Total number of **MULTS** $\leq 2\lg(n)$.
More refined: $\lg(n) +$ (number of 1's in binary rep of n) -1
Example on next page

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

$$265 = 2^8 + 2^3 + 2^0 = (100001001)_2$$

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

$$265 = 2^8 + 2^3 + 2^0 = (100001001)_2$$

$17^{2^0} \equiv 17$ (0 steps)

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

$$265 = 2^8 + 2^3 + 2^0 = (100001001)_2$$

$17^{2^0} \equiv 17$ (0 steps)
$17^{2^1} \equiv 17^2 \equiv 87$ (1 step)

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

$$265 = 2^8 + 2^3 + 2^0 = (100001001)_2$$

$17^{2^0} \equiv 17$ (0 steps)
$17^{2^1} \equiv 17^2 \equiv 87$ (1 step)
$17^{2^2} \equiv 87^2 \equiv 95$ (1 step)

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

$$265 = 2^8 + 2^3 + 2^0 = (100001001)_2$$

$17^{2^0} \equiv 17$ (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)

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

$$265 = 2^8 + 2^3 + 2^0 = (100001001)_2$$

$17^{2^0} \equiv 17$ (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)

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

$$265 = 2^8 + 2^3 + 2^0 = (100001001)_2$$

$17^{2^0} \equiv 17$ (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)

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

$$265 = 2^8 + 2^3 + 2^0 = (100001001)_2$$

$$17^{2^0} \equiv 17 \text{ (0 steps)}$$
$$17^{2^1} \equiv 17^2 \equiv 87 \text{ (1 step)}$$
$$17^{2^2} \equiv 87^2 \equiv 95 \text{ (1 step)}$$
$$17^{2^3} \equiv 95^2 \equiv 36 \text{ (1 step)}$$
$$17^{2^4} \equiv 36^2 \equiv 84 \text{ (1 step)}$$
$$17^{2^5} \equiv 84^2 \equiv 87 \text{ (1 step)}$$
$$17^{2^6} \equiv 87^2 \equiv 95 \text{ (1 step)}$$

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

$$265 = 2^8 + 2^3 + 2^0 = (100001001)_2$$

$$17^{2^0} \equiv 17 \text{ (0 steps)}$$
$$17^{2^1} \equiv 17^2 \equiv 87 \text{ (1 step)}$$
$$17^{2^2} \equiv 87^2 \equiv 95 \text{ (1 step)}$$
$$17^{2^3} \equiv 95^2 \equiv 36 \text{ (1 step)}$$
$$17^{2^4} \equiv 36^2 \equiv 84 \text{ (1 step)}$$
$$17^{2^5} \equiv 84^2 \equiv 87 \text{ (1 step)}$$
$$17^{2^6} \equiv 87^2 \equiv 95 \text{ (1 step)}$$
$$17^{2^7} \equiv 95^2 \equiv 36 \text{ (1 step)}$$

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

$$265 = 2^8 + 2^3 + 2^0 = (100001001)_2$$

$$\begin{aligned}
17^{2^0} &\equiv 17 \text{ (0 steps)} \\
17^{2^1} &\equiv 17^2 \equiv 87 \text{ (1 step)} \\
17^{2^2} &\equiv 87^2 \equiv 95 \text{ (1 step)} \\
17^{2^3} &\equiv 95^2 \equiv 36 \text{ (1 step)} \\
17^{2^4} &\equiv 36^2 \equiv 84 \text{ (1 step)} \\
17^{2^5} &\equiv 84^2 \equiv 87 \text{ (1 step)} \\
17^{2^6} &\equiv 87^2 \equiv 95 \text{ (1 step)} \\
17^{2^7} &\equiv 95^2 \equiv 36 \text{ (1 step)} \\
17^{2^8} &\equiv 36^2 \equiv 84 \text{ (1 step)}
\end{aligned}$$

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

$$265 = 2^8 + 2^3 + 2^0 = (100001001)_2$$

$17^{2^0} \equiv 17$ (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: $17^{265} \pmod{101}$

$$265 = 2^8 + 2^3 + 2^0 = (100001001)_2$$

$17^{2^0} \equiv 17$ (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 multiplications:

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

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

$$265 = 2^8 + 2^3 + 2^0 = (100001001)_2$$

$17^{2^0} \equiv 17$ (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 multiplications:

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

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

An Important Point That Some Students Missed

An Important Point That Some Students Missed

Bill Kunal, many students on piazza seem to say their programs are taking too long. Whats up with that?

An Important Point That Some Students Missed

Bill Kunal, many students on piazza seem to say their programs are taking too long. Whats up with that?

Kunal In repeated squaring you are supposed to do the MOD p at EVERY STEP. Half of the students who emailed were doing the exponentiation and THEN doing the MOD p, so they had really large intermediary numbers which slowed things down.

An Important Point That Some Students Missed

Bill Kunal, many students on piazza seem to say their programs are taking too long. Whats up with that?

Kunal In repeated squaring you are supposed to do the MOD p at EVERY STEP. Half of the students who emailed were doing the exponentiation and THEN doing the MOD p, so they had really large intermediary numbers which slowed things down.

Bill I will emphasize that in class when I do the review.

Generators and Discrete Logarithms

Formally Discrete Log is. . .

Def The **Discrete Log (DL)** problem is a follows:

Formally Discrete Log is...

Def The **Discrete Log (DL)** problem is a follows:

1. Input g, a, p. With $1 \le g, a \le p-1$. g is a gen for $\mathbb{Z}_p^*$.

Formally Discrete Log is...

Def The **Discrete Log (DL)** problem is a follows:

1. Input g, a, p. With $1 \le g, a \le p-1$. g is a gen for $\mathbb{Z}_p^*$.
2. Output x such that $g^x \equiv a \pmod{p}$.

Formally Discrete Log is...

Def The **Discrete Log (DL)** problem is a follows:

1. Input g, a, p. With $1 \le g, a \le p - 1$. g is a gen for $\mathbb{Z}_p^*$.
2. Output x such that $g^x \equiv a \pmod{p}$.

Recall

Formally Discrete Log is...

Def The **Discrete Log (DL)** problem is a follows:

1. Input g, a, p. With $1 \le g, a \le p-1$. g is a gen for $\mathbb{Z}_p^*$.
2. Output x such that $g^x \equiv a \pmod{p}$.

Recall

- A **good** alg would be time $(\log p)^{O(1)}$.

Formally Discrete Log is...

Def The **Discrete Log (DL)** problem is a follows:

1. Input g, a, p. With $1 \le g, a \le p-1$. g is a gen for $\mathbb{Z}_p^*$.
2. Output x such that $g^x \equiv a \pmod{p}$.

Recall

- A **good** alg would be time $(\log p)^{O(1)}$.
- A **bad** alg would be time $p^{O(1)}$.

Formally Discrete Log is. . .

Def The **Discrete Log (DL)** problem is a follows:

1. Input g, a, p. With $1 \le g, a \le p - 1$. g is a gen for $\mathbb{Z}_p^*$.
2. Output x such that $g^x \equiv a \pmod{p}$.

Recall

- A **good** alg would be time $(\log p)^{O(1)}$.
- A **bad** alg would be time $p^{O(1)}$.
- If an algorithm is in time (say) $p^{1/10}$ still not efficient but will force Alice and Bob to up their game.

The Complexity of Discrete Log?

Input is (g, a, p).

The Complexity of Discrete Log?

Input is (g, a, p).

1. Naive algorithm is $O(p)$ time.

The Complexity of Discrete Log?

Input is (g, a, p).

1. Naive algorithm is $O(p)$ time.
2. Exists an $O(\sqrt{p})$ time, $O(\sqrt{p})$ space alg. Time and Space makes it NOT practical.

The Complexity of Discrete Log?

Input is (g, a, p).

1. Naive algorithm is $O(p)$ time.
2. Exists an $O(\sqrt{p})$ time, $O(\sqrt{p})$ space alg. Time and Space makes it NOT practical.
3. Exists an $O(\sqrt{p})$ time, $(\log p)^{O(1)}$ space alg. Space fine, but time still a problem.

The Complexity of Discrete Log?

Input is (g, a, p).

1. Naive algorithm is $O(p)$ time.
2. Exists an $O(\sqrt{p})$ time, $O(\sqrt{p})$ space alg. Time and Space makes it NOT practical.
3. Exists an $O(\sqrt{p})$ time, $(\log p)^{O(1)}$ space alg. Space fine, but time still a problem.
4. Not much progress on theory front since 1985.

The Complexity of Discrete Log?

Input is (g, a, p).

1. Naive algorithm is $O(p)$ time.
2. Exists an $O(\sqrt{p})$ time, $O(\sqrt{p})$ space alg. Time and Space makes it NOT practical.
3. Exists an $O(\sqrt{p})$ time, $(\log p)^{O(1)}$ space alg. Space fine, but time still a problem.
4. Not much progress on theory front since 1985.
5. Discrete Log is in QuantumP.

The Complexity of Discrete Log?

Input is (g, a, p).

1. Naive algorithm is $O(p)$ time.
2. Exists an $O(\sqrt{p})$ time, $O(\sqrt{p})$ space alg. Time and Space makes it NOT practical.
3. Exists an $O(\sqrt{p})$ time, $(\log p)^{O(1)}$ space alg. Space fine, but time still a problem.
4. Not much progress on theory front since 1985.
5. Discrete Log is in QuantumP.

Good Candidate for a hard problem for Eve.

Discrete Log-General

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

Discrete Log-General

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

1. If g is small then $DL(g^a)$ might be easy: $DL_{1009,7}(49) = 2$ since $7^2 \equiv 49 \pmod{1009}$.

Discrete Log-General

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

1. If g is small then $DL(g^a)$ might be easy: $DL_{1009,7}(49) = 2$ since $7^2 \equiv 49 \pmod{1009}$.
2. If g is small then $DL(p - g^a)$ might be easy: $DL_{1009,7}(1009 - 49) = 506$ since $7^{504}7^2 \equiv -7^2 \equiv 1009 - 49 \pmod{1009}$.

Discrete Log-General

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

1. If g is small then $DL(g^a)$ might be easy: $DL_{1009,7}(49) = 2$ since $7^2 \equiv 49 \pmod{1009}$.
2. If g is small then $DL(p - g^a)$ might be easy: $DL_{1009,7}(1009 - 49) = 506$ since $7^{504}7^2 \equiv -7^2 \equiv 1009 - 49 \pmod{1009}$.
3. If $g, a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$ then problem suspected hard.

Discrete Log-General

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

1. If g is small then $DL(g^a)$ might be easy: $DL_{1009,7}(49) = 2$ since $7^2 \equiv 49 \pmod{1009}$.
2. If g is small then $DL(p - g^a)$ might be easy: $DL_{1009,7}(1009 - 49) = 506$ since $7^{504}7^2 \equiv -7^2 \equiv 1009 - 49 \pmod{1009}$.
3. If $g, a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$ then problem suspected hard.
4. **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

Consider What We Already Have Here

- **Exponentiation mod p** is Easy.

Consider What We Already Have Here

- **Exponentiation mod p** is Easy.
- **Discrete Log** is thought to be Hard.

Consider What We Already Have Here

- **Exponentiation mod p** is Easy.
- **Discrete Log** is thought to be Hard.

We want a crypto system where:

Consider What We Already Have Here

- **Exponentiation mod p** is Easy.
- **Discrete Log** is thought to be Hard.

We want a crypto system where:

- Alice and Bob do **Exponentiation mod p** to encrypt and decrypt.

Consider What We Already Have Here

- **Exponentiation mod p** is Easy.
- **Discrete Log** is thought to be Hard.

We want a crypto system where:

- Alice and Bob do **Exponentiation mod p** to encrypt and decrypt.
- Eve has to do **Discrete Log** to crack it.

Consider What We Already Have Here

- **Exponentiation mod p** is Easy.
- **Discrete Log** is thought to be Hard.

We want a crypto system where:

- Alice and Bob do **Exponentiation mod p** to encrypt and decrypt.
- Eve has to do **Discrete Log** to crack it.

Do we have this?

Consider What We Already Have Here

- **Exponentiation mod p** is Easy.
- **Discrete Log** is thought to be Hard.

We want a crypto system where:

- Alice and Bob do **Exponentiation mod p** to encrypt and decrypt.
- Eve has to do **Discrete Log** to crack it.

Do we have this?

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 math done from that point on is mod p.

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

Finding Generators

Finding Gens; How Many Gens Are There?

Problem Given p, find g such that

- g generates $\mathbb{Z}_p^*$.
- $g \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. (We ignore floors and ceilings for notational convenience.)

Finding Gens; How Many Gens Are There?

Problem Given p, find g such that

- g generates $\mathbb{Z}_p^*$.
- $g \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. (We ignore floors and ceilings for notational convenience.)

We could test $\frac{p}{3}$, then $\frac{p}{3} + 1$, etc. Will we hit a generator soon?

Finding Gens; How Many Gens Are There?

Problem Given p, find g such that

- g generates $\mathbb{Z}_p^*$.
- $g \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. (We ignore floors and ceilings for notational convenience.)

We could test $\frac{p}{3}$, then $\frac{p}{3}+1$, etc. Will we hit a generator soon?

How many elts of $\{1, \ldots, p-1\}$ are gens?

Finding Gens; How Many Gens Are There?

Problem Given p, find g such that

- g generates $\mathbb{Z}_p^*$.
- $g \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. (We ignore floors and ceilings for notational convenience.)

We could test $\frac{p}{3}$, then $\frac{p}{3}+1$, etc. Will we hit a generator soon?

How many elts of $\{1, \ldots, p-1\}$ are gens? $\Theta(\frac{p}{\log \log p})$

Finding Gens; How Many Gens Are There?

Problem Given p, find g such that

- g generates $\mathbb{Z}_p^*$.
- $g \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. (We ignore floors and ceilings for notational convenience.)

We could test $\frac{p}{3}$, then $\frac{p}{3} + 1$, etc. Will we hit a generator soon?

How many elts of $\{1, \ldots, p-1\}$ are gens? $\Theta(\frac{p}{\log \log p})$

Hence if you just look for a gen you will find one soon.

Finding Gens: Useful Theorem

Theorem: If g is **not** a generator then there exists x that

Finding Gens: Useful Theorem

Theorem: If g is **not** a generator then there exists x that

1. x divides $p-1$, $x \neq 1$, $x \neq -1$.

Finding Gens: Useful Theorem

Theorem: If g is **not** a generator then there exists x that

1. x divides $p-1$, $x \neq 1$, $x \neq -1$.
2. $g^x \equiv 1$.

So want to use a prime p such that $p-1$ is easy to factor.

Finding Gens: Final Attempt

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

Finding Gens: Final Attempt

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

1. Input p a prime such that $p - 1 = 2q$ where q is prime. (We later explore how we can find such a prime.)

Finding Gens: Final Attempt

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

1. Input p a prime such that $p - 1 = 2q$ where q is prime. (We later explore how we can find such a prime.)
2. Factor $p - 1$. Let F be the set of its factors except $p - 1$. That's EASY: $F = \{2, q\}$.

Finding Gens: Final Attempt

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

1. Input p a prime such that $p - 1 = 2q$ where q is prime. (We later explore how we can find such a prime.)
2. Factor $p - 1$. Let F be the set of its factors except $p - 1$. That's EASY: $F = \{2, q\}$.
3. For $g = \frac{p}{3}$ to $\frac{2p}{3}$:
   Compute g^x for all $x \in F$. If any $= 1$ then g NOT generator. If none are 1 then output g and stop.

Finding Gens: Final Attempt

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

1. Input p a prime such that $p-1=2q$ where q is prime. (We later explore how we can find such a prime.)
2. Factor $p-1$. Let F be the set of its factors except $p-1$. That's EASY: $F=\{2,q\}$.
3. For $g=\frac{p}{3}$ to $\frac{2p}{3}$:
   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 Every iteration does $O(\log p)$ operations.

Finding Gens: Final Attempt

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

1. Input p a prime such that $p-1=2q$ where q is prime. (We later explore how we can find such a prime.)
2. Factor $p-1$. Let F be the set of its factors except $p-1$. That's EASY: $F=\{2,q\}$.
3. For $g=\frac{p}{3}$ to $\frac{2p}{3}$:
   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 Every iteration does $O(\log p)$ operations.
CON: Need both p and $\frac{p-1}{2}$ are primes.

Finding Gens: Final Attempt

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

1. Input p a prime such that $p-1=2q$ where q is prime. (We later explore how we can find such a prime.)
2. Factor $p-1$. Let F be the set of its factors except $p-1$. That's EASY: $F=\{2,q\}$.
3. For $g=\frac{p}{3}$ to $\frac{2p}{3}$:
   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 Every iteration does $O(\log p)$ operations.
CON: Need both p and $\frac{p-1}{2}$ are primes.
CAVEAT We need to pick certain kinds of primes. **Can** do that!

Primality Testing

RECAP

We seek a protocol where Alice and Bob do easy computations and Eve has to do a hard one in order to crack the code.

RECAP

We seek a protocol where Alice and Bob do easy computations and Eve has to do a hard one in order to crack the code.

1. Exponentiation mod p is easy.

RECAP

We seek a protocol where Alice and Bob do easy computations and Eve has to do a hard one in order to crack the code.

1. Exponentiation mod p is easy.
2. Discrete Log is hard.

RECAP

We seek a protocol where Alice and Bob do easy computations and Eve has to do a hard one in order to crack the code.

1. Exponentiation mod p is easy.
2. Discrete Log is hard.
3. In order for Discrete Log to be used we need a prime p and a generator g.

RECAP

We seek a protocol where Alice and Bob do easy computations and Eve has to do a hard one in order to crack the code.

1. Exponentiation mod p is easy.
2. Discrete Log is hard.
3. In order for Discrete Log to be used we need a prime p and a generator g.
4. If p is a safe prime then it is easy to find a generator.

RECAP

We seek a protocol where Alice and Bob do easy computations and Eve has to do a hard one in order to crack the code.

1. Exponentiation mod p is easy.
2. Discrete Log is hard.
3. In order for Discrete Log to be used we need a prime p and a generator g.
4. If p is a safe prime then it is easy to find a generator.
5. **Goal One** Primality Testing. Can easily use this to test if a number is a prime and also if a number is a safe prime.

RECAP

We seek a protocol where Alice and Bob do easy computations and Eve has to do a hard one in order to crack the code.

1. Exponentiation mod p is easy.
2. Discrete Log is hard.
3. In order for Discrete Log to be used we need a prime p and a generator g.
4. If p is a safe prime then it is easy to find a generator.
5. **Goal One** Primality Testing. Can easily use this to test if a number is a prime and also if a number is a safe prime.
6. **Goal Two** Finding a Safe Prime: Given L we will want to quickly generate a safe prime of bit-length L.

Primality Testing

Warning We give a primality that may FAIL.

Primality Testing

Warning We give a primality that may FAIL.

It is **not** used.

Primality Testing

Warning We give a primality that may FAIL.

It is **not** used.

But the **ideas** are used in real algorithms.

A Primality Testing Algorithm

Prior Slides If p is prime and $a \in \mathbb{N}$ then $a^p \equiv a \pmod{p}$.

A Primality Testing Algorithm

Prior Slides If p is prime and $a \in \mathbb{N}$ then $a^p \equiv a \pmod{p}$.
What has been observed If p is **not** prime then **usually** for **most** a, $a^p \not\equiv a \pmod{p}$.

A Primality Testing Algorithm

Prior Slides If p is prime and $a \in \mathbb{N}$ then $a^p \equiv a \pmod{p}$.
What has been observed If p is **not** prime then **usually** for **most** a, $a^p \not\equiv a \pmod{p}$.
Primality Algorithm

A Primality Testing Algorithm

Prior Slides If p is prime and $a \in \mathbb{N}$ then $a^p \equiv a \pmod{p}$.
What has been observed If p is **not** prime then **usually** for **most** a, $a^p \not\equiv a \pmod{p}$.
Primality Algorithm

1. Input p. (In algorithm all arithmetic is mod p.)

A Primality Testing Algorithm

Prior Slides If p is prime and $a \in \mathbb{N}$ then $a^p \equiv a \pmod{p}$.
What has been observed If p is **not** prime then **usually** for **most** a, $a^p \not\equiv a \pmod{p}$.
Primality Algorithm

1. Input p. (In algorithm all arithmetic is mod p.)
2. Form rand $R \subseteq \{2, \ldots, p-1\}$ of size $\sim \lg p$.

A Primality Testing Algorithm

Prior Slides If p is prime and $a \in \mathbb{N}$ then $a^p \equiv a \pmod{p}$.
What has been observed If p is **not** prime then **usually** for **most** a, $a^p \not\equiv a \pmod{p}$.
Primality Algorithm

1. Input p. (In algorithm all arithmetic is mod p.)
2. Form rand $R \subseteq \{2, \ldots, p-1\}$ of size $\sim \lg p$.
3. For each $a \in R$ compute a^p.

A Primality Testing Algorithm

Prior Slides If p is prime and $a \in \mathbb{N}$ then $a^p \equiv a \pmod{p}$.
What has been observed If p is **not** prime then **usually** for **most** a, $a^p \not\equiv a \pmod{p}$.
Primality Algorithm

1. Input p. (In algorithm all arithmetic is mod p.)
2. Form rand $R \subseteq \{2, \ldots, p-1\}$ of size $\sim \lg p$.
3. For each $a \in R$ compute a^p.
   3.1 If ever get $a^p \not\equiv a$ then p **not prime**(we are sure).

A Primality Testing Algorithm

Prior Slides If p is prime and $a \in \mathbb{N}$ then $a^p \equiv a \pmod{p}$.
What has been observed If p is **not** prime then **usually** for **most** a, $a^p \not\equiv a \pmod{p}$.
Primality Algorithm

1. Input p. (In algorithm all arithmetic is mod p.)
2. Form rand $R \subseteq \{2, \ldots, p-1\}$ of size $\sim \lg p$.
3. For each $a \in R$ compute a^p.
   3.1 If ever get $a^p \not\equiv a$ then p **not prime**(we are sure).
   3.2 If for all a, $a^p \equiv a$ then PRIME (we are not sure).

Two reasons for our uncertainty:

A Primality Testing Algorithm

Prior Slides If p is prime and $a \in \mathbb{N}$ then $a^p \equiv a \pmod{p}$.
What has been observed If p is **not** prime then **usually** for **most** a, $a^p \not\equiv a \pmod{p}$.
Primality Algorithm

1. Input p. (In algorithm all arithmetic is mod p.)
2. Form rand $R \subseteq \{2, \ldots, p-1\}$ of size $\sim \lg p$.
3. For each $a \in R$ compute a^p.
   3.1 If ever get $a^p \not\equiv a$ then p **not prime**(we are sure).
   3.2 If for all a, $a^p \equiv a$ then PRIME (we are not sure).

Two reasons for our uncertainty:

- p is composite but we were unlucky with R.

A Primality Testing Algorithm

Prior Slides If p is prime and $a \in \mathbb{N}$ then $a^p \equiv a \pmod{p}$.
What has been observed If p is **not** prime then **usually** for **most** a, $a^p \not\equiv a \pmod{p}$.
Primality Algorithm

1. Input p. (In algorithm all arithmetic is mod p.)
2. Form rand $R \subseteq \{2, \ldots, p-1\}$ of size $\sim \lg p$.
3. For each $a \in R$ compute a^p.
   3.1 If ever get $a^p \not\equiv a$ then p **not prime**(we are sure).
   3.2 If for all a, $a^p \equiv a$ then PRIME (we are not sure).

Two reasons for our uncertainty:

- p is composite but we were unlucky with R.
- There are some composite p such that **for all** a, $a^p \equiv a$.

Primality Testing – What is Really True

Primality Testing – What is Really True

1. Exists algorithm that only has first problem, possible bad luck.

Primality Testing – What is Really True

1. Exists algorithm that only has first problem, possible bad luck.
2. That algorithm has prob of failure $\leq \frac{1}{2^p}$. Good enough!

Primality Testing – What is Really True

1. Exists algorithm that only has first problem, possible bad luck.
2. That algorithm has prob of failure $\leq \frac{1}{2^p}$. Good enough!
3. Exists deterministic poly time algorithm but is much slower.

Primality Testing – What is Really True

1. Exists algorithm that only has first problem, possible bad luck.
2. That algorithm has prob of failure $\leq \frac{1}{2^p}$. Good enough!
3. Exists deterministic poly time algorithm but is much slower.
4. n is a **Carmichael Number** if it is composite and, for all a, $a^n \equiv a$. These are the numbers my algorithm FAILS on.

Primality Testing – What is Really True

1. Exists algorithm that only has first problem, possible bad luck.
2. That algorithm has prob of failure $\leq \frac{1}{2^p}$. Good enough!
3. Exists deterministic poly time algorithm but is much slower.
4. n is a **Carmichael Number** if it is composite and, for all a, $a^n \equiv a$. These are the numbers my algorithm FAILS on. The first few are

Primality Testing – What is Really True

1. Exists algorithm that only has first problem, possible bad luck.
2. That algorithm has prob of failure $\leq \frac{1}{2^p}$. Good enough!
3. Exists deterministic poly time algorithm but is much slower.
4. n is a **Carmichael Number** if it is composite and, for all a, $a^n \equiv a$. These are the numbers my algorithm FAILS on. The first few are

$$561, 1105, 1729, 2465, 2821, 6601, 8911.$$

Primality Testing – What is Really True

1. Exists algorithm that only has first problem, possible bad luck.
2. That algorithm has prob of failure $\leq \frac{1}{2^p}$. Good enough!
3. Exists deterministic poly time algorithm but is much slower.
4. n is a **Carmichael Number** if it is composite and, for all a, $a^n \equiv a$. These are the numbers my algorithm FAILS on. The first few are

   $$561, 1105, 1729, 2465, 2821, 6601, 8911.$$

   There are an infinite number of Carmichael numbers, but they are rare.

Generating Primes

- We just gave a fast algorithm for **testing** if p is prime.

Generating Primes

- We just gave a fast algorithm for **testing** if p is prime.
- We want to **generate** primes.

Generating Primes

- We just gave a fast algorithm for **testing** if p is prime.
- We want to **generate** primes.

New Problem Given L, return an L-bit prime.

Generating Primes

- We just gave a fast algorithm for **testing** if p is prime.
- We want to **generate** primes.

New Problem Given L, return an L-bit prime.
Clarification An L-bit prime has a 1 as left most bit.

Alg for Generating Primes

Given L, generating a prime of length L.

Alg for Generating Primes

Given L, generating a prime of length L.

1. Input(L).

Alg for Generating Primes

Given L, generating a prime of length L.

1. Input(L).
2. Pick $y \in \{0,1\}^{L-1}$ at rand.

Alg for Generating Primes

Given L, generating a prime of length L.

1. Input(L).
2. Pick $y \in \{0,1\}^{L-1}$ at rand.
3. $x = 1y$ (so x is a L-bit number).

Alg for Generating Primes

Given L, generating a prime of length L.

1. Input(L).
2. Pick $y \in \{0,1\}^{L-1}$ at rand.
3. $x = 1y$ (so x is a L-bit number).
4. Test if x is prime.

Alg for Generating Primes

Given L, generating a prime of length L.

1. Input(L).
2. Pick $y \in \{0,1\}^{L-1}$ at rand.
3. $x = 1y$ (so x is a L-bit number).
4. Test if x is prime.
5. If x is prime then output x and stop, else goto step 2.

Alg for Generating Primes

Given L, generating a prime of length L.

1. Input(L).
2. Pick $y \in \{0,1\}^{L-1}$ at rand.
3. $x = 1y$ (so x is a L-bit number).
4. Test if x is prime.
5. If x is prime then output x and stop, else goto step 2.

Is this a good algorithm?

Alg for Generating Primes

Given L, generating a prime of length L.

1. Input(L).
2. Pick $y \in \{0,1\}^{L-1}$ at rand.
3. $x = 1y$ (so x is a L-bit number).
4. Test if x is prime.
5. If x is prime then output x and stop, else goto step 2.

Is this a good algorithm?
PRO Math: returns a prime $\sim 3L^2$ tries with high prob.

Alg for Generating Primes

Given L, generating a prime of length L.

1. Input(L).
2. Pick $y \in \{0,1\}^{L-1}$ at rand.
3. $x = 1y$ (so x is a L-bit number).
4. Test if x is prime.
5. If x is prime then output x and stop, else goto step 2.

Is this a good algorithm?
PRO Math: returns a prime $\sim 3L^2$ tries with high prob.
CON Tests lots of numbers that are obv not prime—e.g, evens.

Alg for Generating Primes

Given L, generating a prime of length L.

1. Input(L).
2. Pick $y \in \{0,1\}^{L-1}$ at rand.
3. $x = 1y$ (so x is a L-bit number).
4. Test if x is prime.
5. If x is prime then output x and stop, else goto step 2.

Is this a good algorithm?
PRO Math: returns a prime $\sim 3L^2$ tries with high prob.
CON Tests lots of numbers that are obv not prime—e.g, evens.
CAVEAT Can make sure never test even numbers. Won't do that in review.

Generating Safe Primes

Definition p is a *safe prime* if p is prime and $\frac{p-1}{2}$ is prime.

Generating Safe Primes

Definition p is a *safe prime* if p is prime and $\frac{p-1}{2}$ is prime.

1. Input(L).

Generating Safe Primes

Definition p is a *safe prime* if p is prime and $\frac{p-1}{2}$ is prime.

1. Input(L).
2. Pick $y \in \{0,1\}^{L-1}$ at rand.

Generating Safe Primes

Definition p is a *safe prime* if p is prime and $\frac{p-1}{2}$ is prime.

1. Input(L).
2. Pick $y \in \{0,1\}^{L-1}$ at rand.
3. $x = 1y$ (note that x is a L-bit number).

Generating Safe Primes

Definition p is a *safe prime* if p is prime and $\frac{p-1}{2}$ is prime.

1. Input(L).
2. Pick $y \in \{0,1\}^{L-1}$ at rand.
3. $x = 1y$ (note that x is a L-bit number).
4. Test if x and $\frac{x-1}{2}$ are prime.

Generating Safe Primes

Definition p is a *safe prime* if p is prime and $\frac{p-1}{2}$ is prime.

1. Input(L).
2. Pick $y \in \{0,1\}^{L-1}$ at rand.
3. $x = 1y$ (note that x is a L-bit number).
4. Test if x and $\frac{x-1}{2}$ are prime.
5. If they both are then output x and stop, else goto step 2.

Generating Safe Primes

Definition p is a *safe prime* if p is prime and $\frac{p-1}{2}$ is prime.

1. Input(L).
2. Pick $y \in \{0,1\}^{L-1}$ at rand.
3. $x = 1y$ (note that x is a L-bit number).
4. Test if x and $\frac{x-1}{2}$ are prime.
5. If they both are then output x and stop, else goto step 2.

Is this a good algorithm?

Generating Safe Primes

Definition p is a *safe prime* if p is prime and $\frac{p-1}{2}$ is prime.

1. Input(L).
2. Pick $y \in \{0,1\}^{L-1}$ at rand.
3. $x = 1y$ (note that x is a L-bit number).
4. Test if x and $\frac{x-1}{2}$ are prime.
5. If they both are then output x and stop, else goto step 2.

Is this a good algorithm?

PRO Math: returns prime quickly with high prob.

Generating Safe Primes

Definition p is a *safe prime* if p is prime and $\frac{p-1}{2}$ is prime.

1. Input(L).
2. Pick $y \in \{0,1\}^{L-1}$ at rand.
3. $x = 1y$ (note that x is a L-bit number).
4. Test if x and $\frac{x-1}{2}$ are prime.
5. If they both are then output x and stop, else goto step 2.

Is this a good algorithm?

PRO Math: returns prime quickly with high prob.

CON Tests lots of numbers that are obv not prime—e.g, evens.

Generating Safe Primes

Definition p is a *safe prime* if p is prime and $\frac{p-1}{2}$ is prime.

1. Input(L).
2. Pick $y \in \{0,1\}^{L-1}$ at rand.
3. $x = 1y$ (note that x is a L-bit number).
4. Test if x and $\frac{x-1}{2}$ are prime.
5. If they both are then output x and stop, else goto step 2.

Is this a good algorithm?

PRO Math: returns prime quickly with high prob.

CON Tests lots of numbers that are obv not prime—e.g, evens.

CAVEAT Can make sure never test evens. Won't do that in this rev.