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 $g = \left\lceil \frac{p}{3} \right\rceil$ 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.

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 $g = \lceil \frac{p}{3} \rceil$ 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.
BIG CON: We still need to factor $p-1$? Really? Darn!

Factoring is Hard. Or is it?

Problem for third method: factoring is hard. But:

1. If Alice picks p, q large and gives Eve pq, then factoring pq seems to be hard.
2. If a Random process generates n then factoring n probably easy (e.g., half the time it's even!).

Factoring is Hard. Or is it?

Problem for third method: factoring is hard. But:

1. If Alice picks p, q large and gives Eve pq, then factoring pq seems to be hard.
2. If a Random process generates n then factoring n probably easy (e.g., half the time it's even!).

We want $p - 1$ to be easy to factor. We don't have an adversary like Alice, but we also don't have a random process either.

Factoring is Hard. Or is it?

Problem for third method: factoring is hard. But:

1. If Alice picks p, q large and gives Eve pq, then factoring pq seems to be hard.
2. If a Random process generates n then factoring n probably easy (e.g., half the time it's even!).

We want $p - 1$ to be easy to factor. We don't have an adversary like Alice, but we also don't have a random process either.

There are three kinds of people in the world:

Factoring is Hard. Or is it?

Problem for third method: factoring is hard. But:

1. If Alice picks p, q large and gives Eve pq, then factoring pq seems to be hard.
2. If a Random process generates n then factoring n probably easy (e.g., half the time it's even!).

We want $p - 1$ to be easy to factor. We don't have an adversary like Alice, but we also don't have a random process either.

There are three kinds of people in the world:

1. Those who make things happen.

Factoring is Hard. Or is it?

Problem for third method: factoring is hard. But:

1. If Alice picks p, q large and gives Eve pq, then factoring pq seems to be hard.
2. If a Random process generates n then factoring n probably easy (e.g., half the time it's even!).

We want $p - 1$ to be easy to factor. We don't have an adversary like Alice, but we also don't have a random process either.

There are three kinds of people in the world:

1. Those who make things happen.
2. Those who watch things happen.

Factoring is Hard. Or is it?

Problem for third method: factoring is hard. But:

1. If Alice picks p, q large and gives Eve pq, then factoring pq seems to be hard.
2. If a Random process generates n then factoring n probably easy (e.g., half the time it's even!).

We want $p - 1$ to be easy to factor. We don't have an adversary like Alice, but we also don't have a random process either.

There are three kinds of people in the world:

1. Those who make things happen.
2. Those who watch things happen.
3. Those who wonder what happened.

Factoring is Hard. Or is it?

Problem for third method: factoring is hard. But:

1. If Alice picks p, q large and gives Eve pq, then factoring pq seems to be hard.
2. If a Random process generates n then factoring n probably easy (e.g., half the time it's even!).

We want $p - 1$ to be easy to factor. We don't have an adversary like Alice, but we also don't have a random process either.

There are three kinds of people in the world:

1. Those who make things happen.
2. Those who watch things happen.
3. Those who wonder what happened.

We need to make things happen. We need to make $p - 1$ easy to factor.

Finding Gens: Fourth Attempt

Idea: Pick p such that $p-1=2q$ where q is prime.
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 $g=\lceil \frac{p}{3} \rceil$ 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: Fourth Attempt

Idea: Pick p such that $p-1=2q$ where q is prime.
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 $g=\lceil \frac{p}{3} \rceil$ 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 above $O(1)$ iterations.

Finding Gens: Fourth Attempt

Idea: Pick p such that $p-1=2q$ where q is prime.
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 $g=\lceil \frac{p}{3} \rceil$ 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 above $O(1)$ iterations.
PRO: Every iteration does $O(|F|(\log p)) = O(\log p)$ operations.

Finding Gens: Fourth Attempt

Idea: Pick p such that $p-1=2q$ where q is prime.
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 $g=\lceil\frac{p}{3}\rceil$ 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 above $O(1)$ iterations.
PRO: Every iteration does $O(|F|(\log p))=O(\log p)$ operations.
CON: Need both p and $\frac{p-1}{2}$ are primes.

Finding Gens: Fourth Attempt

Idea: Pick p such that $p-1=2q$ where q is prime.
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 $g=\lceil \frac{p}{3} \rceil$ 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 above $O(1)$ iterations.
PRO: Every iteration does $O(|F|(\log p)) = O(\log p)$ operations.
CON: Need both p and $\frac{p-1}{2}$ are primes.
We need to pick certain kinds of primes. Can do that!

Primality Testing

Primality Testing

Warning: The next few slides will culminate in a test for primality that may FAIL. It is NOT used. But ideas are used in real algorithm.

Lemma

p prime, $1 \le i \le p-1$*, then* $\frac{p!}{i!(p-i)!} \in \mathbb{N}$ *and is divisible by p.*

Primality Testing

Warning: The next few slides will culminate in a test for primality that may FAIL. It is NOT used. But ideas are used in real algorithm.

Lemma

p prime, $1 \le i \le p-1$, then $\frac{p!}{i!(p-i)!} \in \mathbb{N}$ and is divisible by p.

Proof.

Why is $\frac{p!}{i!(p-i)!} \in \mathbb{N}$?

Primality Testing

Warning: The next few slides will culminate in a test for primality that may FAIL. It is NOT used. But ideas are used in real algorithm.

Lemma

p prime, $1 \le i \le p-1$, then $\frac{p!}{i!(p-i)!} \in \mathbb{N}$ and is divisible by p.

Proof.

Why is $\frac{p!}{i!(p-i)!} \in \mathbb{N}$?

$\frac{p!}{i!(p-i)!}$? is the answer to a question that has a $\mathbb{N}$ solution:

How many ways can you choose i items out of p?

Primality Testing

Warning: The next few slides will culminate in a test for primality that may FAIL. It is NOT used. But ideas are used in real algorithm.

Lemma

p prime, $1 \le i \le p-1$, then $\frac{p!}{i!(p-i)!} \in \mathbb{N}$ and is divisible by p.

Proof.

Why is $\frac{p!}{i!(p-i)!} \in \mathbb{N}$?

$\frac{p!}{i!(p-i)!}$? is the answer to a question that has a $\mathbb{N}$ solution:

How many ways can you choose i items out of p?

Why does p divide $\frac{p!}{i!(p-i)!}$?

Primality Testing

Warning: The next few slides will culminate in a test for primality that may FAIL. It is NOT used. But ideas are used in real algorithm.

Lemma

p prime, $1 \le i \le p-1$, then $\frac{p!}{i!(p-i)!} \in \mathbb{N}$ and is divisible by p.

Proof.

Why is $\frac{p!}{i!(p-i)!} \in \mathbb{N}$?

$\frac{p!}{i!(p-i)!}$? is the answer to a question that has a $\mathbb{N}$ solution:

How many ways can you choose i items out of p?

Why does p divide $\frac{p!}{i!(p-i)!}$?

p divides the numerator, p does not divide the denominator, and p is prime. Hence p divides the number.

$\square$

Note: $\binom{p}{i} = \frac{p!}{(p-i)!i!}$.

Primality Testing

Lemma

(Binomial Theorem) For any $n \in \mathbb{N}$, $(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} x^i y^{n-i}$

Lemma

(Fermat's Little Theorem) If p *prime,* $a \in \mathbb{N}$ *then* $a^p \equiv a \pmod{p}$*.*

Proof.

Fix prime p. By induction on a. Base Case: $1^p \equiv 1$.

Ind Hyp: $a^p \equiv a \pmod{p}$

Ind Step: $(a+1)^p = \binom{p}{p} a^p + \binom{p}{p-1} a^{p-1} + \cdots + \binom{p}{1} a^1 + \binom{p}{0} a^0$.

By previous lemma $\binom{p}{1} \equiv \binom{p}{2} \equiv \cdots \equiv \binom{p}{p-1} \equiv 0$. Hence

$$(a+1)^p \equiv \binom{p}{0} a^p + \binom{p}{p} a^0 \equiv a^p + 1 \equiv a + 1 \text{ Last } \equiv \text{ by Ind Hyp}$$

$\square$

Fermat's Little Theorem Special Case

RECALL:

Lemma

(Fermat's Little Theorem) If p prime, $a \in \mathbb{N}$ then $a^p \equiv a \pmod{p}$.

We want to divide both sides by a to get $a^{p-1} \equiv 1 \pmod{p}$. Can we always do this? Discuss.

Fermat's Little Theorem Special Case

RECALL:

Lemma

(Fermat's Little Theorem) If p prime, $a \in \mathbb{N}$ then $a^p \equiv a \pmod{p}$.

We want to divide both sides by a to get $a^{p-1} \equiv 1 \pmod{p}$. Can we always do this? Discuss.

No. Need $a \not\equiv 0 \pmod{p}$. Okay, make that a premise:

Lemma

If p prime, $a \in \mathbb{N}$, $a \not\equiv 0 \pmod{p}$, $a^{p-1} \equiv 1 \pmod{p}$.

Primality Testing

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

- If 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

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

   561, 1105, 1729, 2465, 2821, 6601, 8911
6. Carmichael numbers are rare.

Generating Primes (also needed for RSA)

Take as given: Primality Testing is FAST.

First Attempt at, given L, generate 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 true 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?

Generating Primes (also needed for RSA)

Take as given: Primality Testing is FAST.

First Attempt at, given L, generate 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 true 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 tells us returns a prime within $3L^2$ tries with high prob.

Generating Primes (also needed for RSA)

Take as given: Primality Testing is FAST.

First Attempt at, given L, generate 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 true 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 tells us returns a prime within $3L^2$ tries 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.

First Attempt at, given L, generate a safe prime of length L

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

First Attempt at, given L, generate a safe prime of length L

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

First Attempt at, given L, generate a safe prime of length L

1. Input(L)
2. Pick $y \in \{0,1\}^{L-1}$ at rand.
3. $x = 1y$ (note that x is a true 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 tells us returns prime quickly with high prob.

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

Speed Prime-Finding: $n \not\equiv 0$ mod 2

We picked *any* $L-1$-bit string, including ones that end in 0, so even which we know we don't want.
IDEA: Pick $L-2$ bit string, put 1 on its right and on its left.
Is this a good idea? Vote

Speed Prime-Finding: $n \not\equiv 0$ mod 2

We picked *any* $L-1$-bit string, including ones that end in 0, so even which we know we don't want.
IDEA: Pick $L-2$ bit string, put 1 on its right and on its left.
Is this a good idea? Vote

PRO: Do not waste time testing even numbers.

Speed Prime-Finding: $n \not\equiv 0 \bmod 2$

We picked *any* $L-1$-bit string, including ones that end in 0, so even which we know we don't want.
IDEA: Pick $L-2$ bit string, put 1 on its right and on its left.
Is this a good idea? Vote

PRO: Do not waste time testing even numbers.
CON: Does it really save that much time?

Speed Prime-Finding: $n \not\equiv 0$ mod 2

We picked *any* $L-1$-bit string, including ones that end in 0, so even which we know we don't want.
IDEA: Pick $L-2$ bit string, put 1 on its right and on its left.
Is this a good idea? Vote

PRO: Do not waste time testing even numbers.
CON: Does it really save that much time?
CAVEAT: Can we extend so we don't test numbers div by 3?
Discuss

Speed Prime-Finding: $n \not\equiv 0 \bmod 2$

We picked *any* $L-1$-bit string, including ones that end in 0, so even which we know we don't want.
IDEA: Pick $L-2$ bit string, put 1 on its right and on its left.
Is this a good idea? Vote

PRO: Do not waste time testing even numbers.
CON: Does it really save that much time?
CAVEAT: Can we extend so we don't test numbers div by 3?
Discuss
Yes

Speed Up Prime-Finding: $\not\equiv 0$ mod 2 or 3

2 divides n iff $(\exists k)[n = 2k]$
2 does not divide n iff $(\exists k)[n = 2k + 1]$

Speed Up Prime-Finding: $\neq 0$ mod 2 or 3

2 divides n iff $(\exists k)[n = 2k]$
2 does not divide n iff $(\exists k)[n = 2k + 1]$

3 divides n iff $(\exists k)[n = 3k]$
3 does not divide n iff $(\exists k)(\exists i \in \{1, 2\})[n = 3k + i]$

Speed Up Prime-Finding: $\not\equiv 0$ mod 2 or 3

2 divides n iff $(\exists k)[n = 2k]$
2 does not divide n iff $(\exists k)[n = 2k + 1]$

3 divides n iff $(\exists k)[n = 3k]$
3 does not divide n iff $(\exists k)(\exists i \in \{1, 2\})[n = 3k + i]$

How to get both?

Speed Up Prime-Finding: $\not\equiv 0$ mod 2 or 3

2 divides n iff $(\exists k)[n = 2k]$
2 does not divide n iff $(\exists k)[n = 2k + 1]$

3 divides n iff $(\exists k)[n = 3k]$
3 does not divide n iff $(\exists k)(\exists i \in \{1, 2\})[n = 3k + i]$

How to get both?
Neither 2 nor 3 divides n iff $(\exists k)(\exists i \in \{1, 5\})[n = 6k + i]$

Speed Up Prime-Finding: $\neq 0$ mod 2 or 3

2 divides n iff $(\exists k)[n = 2k]$
2 does not divide n iff $(\exists k)[n = 2k + 1]$

3 divides n iff $(\exists k)[n = 3k]$
3 does not divide n iff $(\exists k)(\exists i \in \{1, 2\})[n = 3k + i]$

How to get both?
Neither 2 nor 3 divides n iff $(\exists k)(\exists i \in \{1, 5\})[n = 6k + i]$

So need to generate numbers of the form $6k + 1$ and $6k + 5$.
Caveat: Might not get a prime *of length L*. We ignore this for now.

Speed Up Prime-Finding: $\not\equiv 0$ mod 2,3

ALGORITHM: Pick an $L-3$ bit string, add a 1 to the left, mult by 6, add 1: get L-bit string which is of form $6k+1$. (Might be $L+1$ long.)

Is this a good idea? Vote

Speed Up Prime-Finding: $\not\equiv 0$ mod 2,3

ALGORITHM: Pick an $L-3$ bit string, add a 1 to the left, mult by 6, add 1: get L-bit string which is of form $6k+1$. (Might be $L+1$ long.)
Is this a good idea? Vote

PRO: Do not waste time testing numbers $\equiv 0$ mod 2 or 3.

Speed Up Prime-Finding: $\not\equiv 0$ mod 2,3

ALGORITHM: Pick an $L-3$ bit string, add a 1 to the left, mult by 6, add 1: get L-bit string which is of form $6k+1$. (Might be $L+1$ long.)
Is this a good idea? Vote

PRO: Do not waste time testing numbers $\equiv 0$ mod 2 or 3.
CON: Only use primes of form $6k+1$. Who knows, maybe such primes are easy to deal with for Eve?

Speed Up Prime-Finding: $\not\equiv 0$ mod 2,3

ALGORITHM: Pick an $L-3$ bit string, add a 1 to the left, mult by 6, add 1: get L-bit string which is of form $6k+1$. (Might be $L+1$ long.)
Is this a good idea? Vote

PRO: Do not waste time testing numbers $\equiv 0 \bmod 2$ or 3.
CON: Only use primes of form $6k+1$. Who knows, maybe such primes are easy to deal with for Eve?
CAVEAT: Can we modify to avoid this problem?

Speed Up Prime-Finding: $\not\equiv 0$ mod 2,3

ALGORITHM: Pick an $L-3$ bit string, add a 1 to the left, mult by 6, add 1: get L-bit string which is of form $6k+1$. (Might be $L+1$ long.)
Is this a good idea? Vote

PRO: Do not waste time testing numbers $\equiv 0$ mod 2 or 3.
CON: Only use primes of form $6k+1$. Who knows, maybe such primes are easy to deal with for Eve?
CAVEAT: Can we modify to avoid this problem?
Yes

Speed Up Alg Prime-Finding: $\not\equiv 0$ mod 2,3

IDEA: Pick an $L-3$ bit string, add a 1 to the left, mult by 6, add $i \in \{1,5\}$ picked at rand: get L-bit string which is of form $6k+1$ OR $6k+5$. (Might be $L+1$ long.)

Is this a good idea? Vote

Speed Up Alg Prime-Finding: $\not\equiv 0$ mod 2,3

IDEA: Pick an $L-3$ bit string, add a 1 to the left, mult by 6, add $i \in \{1,5\}$ picked at rand: get L-bit string which is of form $6k+1$ OR $6k+5$. (Might be $L+1$ long.)
Is this a good idea? Vote

PRO: Do not waste time testing numbers $\equiv 0$ mod 2 or 3.

Speed Up Alg Prime-Finding: $\not\equiv 0$ mod 2,3

IDEA: Pick an $L-3$ bit string, add a 1 to the left, mult by 6, add $i \in \{1,5\}$ picked at rand: get L-bit string which is of form $6k+1$ OR $6k+5$. (Might be $L+1$ long.)
Is this a good idea? Vote

PRO: Do not waste time testing numbers $\equiv 0$ mod 2 or 3.
PRO: Do not get a prime of a certain form.

Speed Up Alg Prime-Finding: $\not\equiv 0$ mod 2,3

IDEA: Pick an $L-3$ bit string, add a 1 to the left, mult by 6, add $i \in \{1,5\}$ picked at rand: get L-bit string which is of form $6k+1$ OR $6k+5$. (Might be $L+1$ long.)
Is this a good idea? Vote

PRO: Do not waste time testing numbers $\equiv 0$ mod 2 or 3.
PRO: Do not get a prime of a certain form.
CON: Getting more complicated. Is it worth it? Do not know.

Speed Up Alg Prime-Finding: $\not\equiv 0$ mod 2,3

IDEA: Pick an $L - 3$ bit string, add a 1 to the left, mult by 6, add $i \in \{1, 5\}$ picked at rand: get L-bit string which is of form $6k + 1$ OR $6k + 5$. (Might be $L + 1$ long.)
Is this a good idea? Vote

PRO: Do not waste time testing numbers $\equiv 0$ mod 2 or 3.
PRO: Do not get a prime of a certain form.
CON: Getting more complicated. Is it worth it? Do not know.
CAVEAT: Can we extend to 2,3,5? 2,3,5,7? etc.

Speed Up Prime-Finding: $\not\equiv 0$ mod 2,3,5

2 divides n iff $(\exists k)[n = 2k]$
2 does not divide n iff $(\exists k)[n = 2k + 1]$

Speed Up Prime-Finding: $\neq 0$ mod 2,3,5

2 divides n iff $(\exists k)[n = 2k]$
2 does not divide n iff $(\exists k)[n = 2k + 1]$

3 divides n iff $(\exists k)[n = 3k]$
3 does not divide n iff $(\exists k)(\exists i \in \{1, 2\})[n = 3k + i]$

Speed Up Prime-Finding: $\not\equiv 0$ mod 2,3,5

2 divides n iff $(\exists k)[n = 2k]$
2 does not divide n iff $(\exists k)[n = 2k + 1]$

3 divides n iff $(\exists k)[n = 3k]$
3 does not divide n iff $(\exists k)(\exists i \in \{1, 2\})[n = 3k + i]$
5 divides n iff $(\exists k)[n = 5k]$
5 does not divide n iff $(\exists k)(\exists i \in \{1, 2, 3, 4\})[n = 5k + i]$

How to get all three?

Speed Up Prime-Finding: $\not\equiv 0$ mod 2,3,5

2 divides n iff $(\exists k)[n = 2k]$
2 does not divide n iff $(\exists k)[n = 2k + 1]$

3 divides n iff $(\exists k)[n = 3k]$
3 does not divide n iff $(\exists k)(\exists i \in \{1, 2\})[n = 3k + i]$
5 divides n iff $(\exists k)[n = 5k]$
5 does not divide n iff $(\exists k)(\exists i \in \{1, 2, 3, 4\})[n = 5k + i]$

How to get all three? We use mod 30. We only want numbers of the form

$$\{30k + i : i \in \{1, 7, 11, 13, 17, 19, 23, 29\}\}$$

Speed Up Prime-Finding: $\neq 0$ mod 2,3, or 5

Let $X = \{1, 7, 11, 13, 17, 19, 23, 29\}\}$.
ALGORITHM: Pick an $L - 7$ bit string, add a 1 to the left, mult by 30, add one of $\{1, 7, 11, 13, 17, 19, 23, 29\}$.
Is this a good idea? Vote

Speed Up Prime-Finding: $\not\equiv 0$ mod 2,3, or 5

Let $X = \{1, 7, 11, 13, 17, 19, 23, 29\}\}$.
ALGORITHM: Pick an $L - 7$ bit string, add a 1 to the left, mult by 30, add one of $\{1, 7, 11, 13, 17, 19, 23, 29\}$.
Is this a good idea? Vote

PRO: Do not waste time testing numbers $\equiv 0$ mod 2 or 3 or 5.

Speed Up Prime-Finding: $\not\equiv 0 \bmod 2,3,$ or 5

Let $X = \{1, 7, 11, 13, 17, 19, 23, 29\}\}$.
ALGORITHM: Pick an $L - 7$ bit string, add a 1 to the left, mult by 30, add one of $\{1, 7, 11, 13, 17, 19, 23, 29\}$.
Is this a good idea? Vote

PRO: Do not waste time testing numbers $\equiv 0$ mod 2 or 3 or 5.
CON: Number of bits around L, but could be off by a few.

Speed Up Prime-Finding: $\not\equiv 0$ mod 2,3, or 5

Let $X = \{1, 7, 11, 13, 17, 19, 23, 29\}\}$.
ALGORITHM: Pick an $L - 7$ bit string, add a 1 to the left, mult by 30, add one of $\{1, 7, 11, 13, 17, 19, 23, 29\}$.
Is this a good idea? Vote

PRO: Do not waste time testing numbers $\equiv 0$ mod 2 or 3 or 5.
CON: Number of bits around L, but could be off by a few.
CON: Too much trouble.

Summary of Where We Are

1. Finding primes p such that $p-1=2q$, q a prime, EASY
2. Given such a p, finding generator g, EASY.
3. Given such a p, finding generator $g \in \{\frac{p}{3}, \frac{2p}{3}\}$ EASY.
4. Given p, g, a finding $g^a \pmod{p}$ EASY.
5. The following problem thought to be hard:
   Input: prime p, generator g, Number a $a, g \in \{\frac{p}{3}, \frac{2p}{3}\}$
   Output: The x such that $g^x \equiv a \pmod{p}$

The problem thought to be hard is essentially the discrete log problem, though we have safeguarded against easy instances.

Summary of Where We Are

1. Finding primes p such that $p-1=2q$, q a prime, EASY
2. Given such a p, finding generator g, EASY.
3. Given such a p, finding generator $g \in \{\frac{p}{3}, \frac{2p}{3}\}$ EASY.
4. Given p, g, a finding $g^a \pmod p$ EASY.
5. The following problem thought to be hard:
   Input: prime p, generator g, Number a $a, g \in \{\frac{p}{3}, \frac{2p}{3}\}$
   Output: The x such that $g^x \equiv a \pmod p$

The problem thought to be hard is essentially the discrete log problem, though we have safeguarded against easy instances. We hope.

Convention (Possibly Repeated)

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 $\{1, \ldots, p-1\}$.

The Diffie-Hellman Key Exchange

Alice and Bob will share a secret s. Security parameter L.

1. Alice finds a (p, g), p of length L, g gen for $\mathbb{Z}_p^*$.
   $g \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$.

The Diffie-Hellman Key Exchange

Alice and Bob will share a secret s. Security parameter L.

1. Alice finds a (p, g), p of length L, g gen for $\mathbb{Z}_p^*$.
   $g \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$.
2. Alice sends (p, g) to Bob in the clear (Eve can see it).

The Diffie-Hellman Key Exchange

Alice and Bob will share a secret s. Security parameter L.

1. Alice finds a (p, g), p of length L, g gen for $\mathbb{Z}_p^*$.
   $g \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$.
2. Alice sends (p, g) to Bob in the clear (Eve can see it).
3. Alice picks rand $a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Alice computes $g^a \pmod{p}$ and sends it to Bob in the clear (Eve can see it).

The Diffie-Hellman Key Exchange

Alice and Bob will share a secret s. Security parameter L.

1. Alice finds a (p, g), p of length L, g gen for $\mathbb{Z}_p^*$.
   $g \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$.
2. Alice sends (p, g) to Bob in the clear (Eve can see it).
3. Alice picks rand $a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Alice computes $g^a \pmod{p}$ and sends it to Bob in the clear (Eve can see it).
4. Bob picks rand $b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Bob computes $g^b \pmod{p}$ and sends it to Alice in the clear (Eve can see it).

The Diffie-Hellman Key Exchange

Alice and Bob will share a secret s. Security parameter L.

1. Alice finds a (p,g), p of length L, g gen for $\mathbb{Z}_p^*$.
   $g \in \{\frac{p}{3},\ldots,\frac{2p}{3}\}$.
2. Alice sends (p,g) to Bob in the clear (Eve can see it).
3. Alice picks rand $a \in \{\frac{p}{3},\ldots,\frac{2p}{3}\}$. Alice computes $g^a \pmod{p}$ and sends it to Bob in the clear (Eve can see it).
4. Bob picks rand $b \in \{\frac{p}{3},\ldots,\frac{2p}{3}\}$. Bob computes $g^b \pmod{p}$ and sends it to Alice in the clear (Eve can see it).
5. Alice computes $(g^b)^a = g^{ab} \pmod{p}$.

The Diffie-Hellman Key Exchange

Alice and Bob will share a secret s. Security parameter L.

1. Alice finds a (p, g), p of length L, g gen for $\mathbb{Z}_p^*$. $g \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$.
2. Alice sends (p, g) to Bob in the clear (Eve can see it).
3. Alice picks rand $a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Alice computes $g^a \pmod{p}$ and sends it to Bob in the clear (Eve can see it).
4. Bob picks rand $b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Bob computes $g^b \pmod{p}$ and sends it to Alice in the clear (Eve can see it).
5. Alice computes $(g^b)^a = g^{ab} \pmod{p}$.
6. Bob computes $(g^a)^b = g^{ab} \pmod{p}$.

The Diffie-Hellman Key Exchange

Alice and Bob will share a secret s. Security parameter L.

1. Alice finds a (p,g), p of length L, g gen for $\mathbb{Z}_p^*$. $g \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$.
2. Alice sends (p,g) to Bob in the clear (Eve can see it).
3. Alice picks rand $a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Alice computes $g^a \pmod{p}$ and sends it to Bob in the clear (Eve can see it).
4. Bob picks rand $b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Bob computes $g^b \pmod{p}$ and sends it to Alice in the clear (Eve can see it).
5. Alice computes $(g^b)^a = g^{ab} \pmod{p}$.
6. Bob computes $(g^a)^b = g^{ab} \pmod{p}$.
7. g^{ab} is the shared secret.

The Diffie-Hellman Key Exchange

Alice and Bob will share a secret s. Security parameter L.

1. Alice finds a (p, g), p of length L, g gen for $\mathbb{Z}_p^*$. $g \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$.
2. Alice sends (p, g) to Bob in the clear (Eve can see it).
3. Alice picks rand $a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Alice computes $g^a \pmod{p}$ and sends it to Bob in the clear (Eve can see it).
4. Bob picks rand $b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Bob computes $g^b \pmod{p}$ and sends it to Alice in the clear (Eve can see it).
5. Alice computes $(g^b)^a = g^{ab} \pmod{p}$.
6. Bob computes $(g^a)^b = g^{ab} \pmod{p}$.
7. g^{ab} is the shared secret.

PRO: Alice and Bob can execute the protocol easily.

The Diffie-Hellman Key Exchange

Alice and Bob will share a secret s. Security parameter L.

1. Alice finds a (p, g), p of length L, g gen for $\mathbb{Z}_p^*$.
   $g \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$.
2. Alice sends (p, g) to Bob in the clear (Eve can see it).
3. Alice picks rand $a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Alice computes $g^a \pmod{p}$ and sends it to Bob in the clear (Eve can see it).
4. Bob picks rand $b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Bob computes $g^b \pmod{p}$ and sends it to Alice in the clear (Eve can see it).
5. Alice computes $(g^b)^a = g^{ab} \pmod{p}$.
6. Bob computes $(g^a)^b = g^{ab} \pmod{p}$.
7. g^{ab} is the shared secret.

PRO: Alice and Bob can execute the protocol easily.
Biggest PRO: Alice and Bob never had to meet!

The Diffie-Hellman Key Exchange

Alice and Bob will share a secret s. Security parameter L.

1. Alice finds a (p, g), p of length L, g gen for $\mathbb{Z}_p^*$.
   $g \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$.
2. Alice sends (p, g) to Bob in the clear (Eve can see it).
3. Alice picks rand $a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Alice computes $g^a \pmod{p}$ and sends it to Bob in the clear (Eve can see it).
4. Bob picks rand $b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Bob computes $g^b \pmod{p}$ and sends it to Alice in the clear (Eve can see it).
5. Alice computes $(g^b)^a = g^{ab} \pmod{p}$.
6. Bob computes $(g^a)^b = g^{ab} \pmod{p}$.
7. g^{ab} is the shared secret.

PRO: Alice and Bob can execute the protocol easily.
Biggest PRO: Alice and Bob never had to meet!
Question: Can Eve find out s?

Have Students DO The DH Key Exchange

Pick out two students who I will call Alice and Bob.

Have Students DO The DH Key Exchange

Pick out two students who I will call Alice and Bob.

Have Students DO The DH Key Exchange

Pick out two students who I will call Alice and Bob.

1. ALICE: Pick safe prime $256 \leq p \leq 511$ (so length 9).

Have Students DO The DH Key Exchange

Pick out two students who I will call Alice and Bob.

1. ALICE: Pick safe prime $256 \le p \le 511$ (so length 9).
2. ALICE: Find a generator for $\mathbb{Z}_p^*$ that is not too big or small.

Have Students DO The DH Key Exchange

Pick out two students who I will call Alice and Bob.

1. ALICE: Pick safe prime $256 \le p \le 511$ (so length 9).
2. ALICE: Find a generator for $\mathbb{Z}_p^*$ that is not too big or small.
3. ALICE: Yell out (p, g).

Have Students DO The DH Key Exchange

Pick out two students who I will call Alice and Bob.

1. ALICE: Pick safe prime $256 \le p \le 511$ (so length 9).
2. ALICE: Find a generator for $\mathbb{Z}_p^*$ that is not too big or small.
3. ALICE: Yell out (p, g).
4. ALICE: Pick a rand $a \in \mathbb{Z}_p^*$ that is not too big or small. Write it down for later verification.

Have Students DO The DH Key Exchange

Pick out two students who I will call Alice and Bob.

1. ALICE: Pick safe prime $256 \le p \le 511$ (so length 9).
2. ALICE: Find a generator for $\mathbb{Z}_p^*$ that is not too big or small.
3. ALICE: Yell out (p, g).
4. ALICE: Pick a rand $a \in \mathbb{Z}_p^*$ that is not too big or small. Write it down for later verification.
5. ALICE: Compute $g^a \pmod{p}$. YELL IT OUT.

Have Students DO The DH Key Exchange

Pick out two students who I will call Alice and Bob.

1. ALICE: Pick safe prime $256 \le p \le 511$ (so length 9).
2. ALICE: Find a generator for $\mathbb{Z}_p^*$ that is not too big or small.
3. ALICE: Yell out (p, g).
4. ALICE: Pick a rand $a \in \mathbb{Z}_p^*$ that is not too big or small. Write it down for later verification.
5. ALICE: Compute $g^a \pmod{p}$. YELL IT OUT.
6. BOB: Pick a rand $b \in \mathbb{Z}_p^*$ that is not too big or small. Write it down for later verification.

Have Students DO The DH Key Exchange

Pick out two students who I will call Alice and Bob.

1. ALICE: Pick safe prime $256 \le p \le 511$ (so length 9).
2. ALICE: Find a generator for $\mathbb{Z}_p^*$ that is not too big or small.
3. ALICE: Yell out (p, g).
4. ALICE: Pick a rand $a \in \mathbb{Z}_p^*$ that is not too big or small. Write it down for later verification.
5. ALICE: Compute $g^a \pmod{p}$. YELL IT OUT.
6. BOB: Pick a rand $b \in \mathbb{Z}_p^*$ that is not too big or small. Write it down for later verification.
7. BOB: Compute $g^b \pmod{p}$. YELL IT OUT.

Have Students DO The DH Key Exchange

Pick out two students who I will call Alice and Bob.

1. ALICE: Pick safe prime $256 \le p \le 511$ (so length 9).
2. ALICE: Find a generator for $\mathbb{Z}_p^*$ that is not too big or small.
3. ALICE: Yell out (p, g).
4. ALICE: Pick a rand $a \in \mathbb{Z}_p^*$ that is not too big or small. Write it down for later verification.
5. ALICE: Compute $g^a \pmod{p}$. YELL IT OUT.
6. BOB: Pick a rand $b \in \mathbb{Z}_p^*$ that is not too big or small. Write it down for later verification.
7. BOB: Compute $g^b \pmod{p}$. YELL IT OUT.
8. ALICE: Compute $(g^b)^a \pmod{p}$.

Have Students DO The DH Key Exchange

Pick out two students who I will call Alice and Bob.

1. ALICE: Pick safe prime $256 \le p \le 511$ (so length 9).
2. ALICE: Find a generator for $\mathbb{Z}_p^*$ that is not too big or small.
3. ALICE: Yell out (p, g).
4. ALICE: Pick a rand $a \in \mathbb{Z}_p^*$ that is not too big or small. Write it down for later verification.
5. ALICE: Compute $g^a \pmod{p}$. YELL IT OUT.
6. BOB: Pick a rand $b \in \mathbb{Z}_p^*$ that is not too big or small. Write it down for later verification.
7. BOB: Compute $g^b \pmod{p}$. YELL IT OUT.
8. ALICE: Compute $(g^b)^a \pmod{p}$.
9. BOB: Compute $(g^a)^b \pmod{p}$.

Have Students DO The DH Key Exchange

Pick out two students who I will call Alice and Bob.

1. ALICE: Pick safe prime $256 \le p \le 511$ (so length 9).
2. ALICE: Find a generator for $\mathbb{Z}_p^*$ that is not too big or small.
3. ALICE: Yell out (p, g).
4. ALICE: Pick a rand $a \in \mathbb{Z}_p^*$ that is not too big or small. Write it down for later verification.
5. ALICE: Compute $g^a \pmod{p}$. YELL IT OUT.
6. BOB: Pick a rand $b \in \mathbb{Z}_p^*$ that is not too big or small. Write it down for later verification.
7. BOB: Compute $g^b \pmod{p}$. YELL IT OUT.
8. ALICE: Compute $(g^b)^a \pmod{p}$.
9. BOB: Compute $(g^a)^b \pmod{p}$.
10. At the count of 3 both yell out your number at the same time.

What Do We Really Know about Diffie-Hellman?

If Eve can compute Discrete Log quickly then she can crack DH:

1. Eve sees g^a, g^b.
2. Eve computes Discrete Log to find a, b.
3. Eve computes $g^{ab} \pmod{p}$.

Question: If Eve can crack DH then Eve can compute Discrete Log. VOTE: Y, N, UNKNOWN TO SCIENCE.

What Do We Really Know about Diffie-Hellman?

If Eve can compute Discrete Log quickly then she can crack DH:

1. Eve sees g^a, g^b.
2. Eve computes Discrete Log to find a, b.
3. Eve computes $g^{ab} \pmod{p}$.

Question: If Eve can crack DH then Eve can compute Discrete Log. VOTE: Y, N, UNKNOWN TO SCIENCE.

Unknown to Science

What Do We Really Know about Diffie-Hellman?

If Eve can compute Discrete Log quickly then she can crack DH:

1. Eve sees g^a, g^b.
2. Eve computes Discrete Log to find a, b.
3. Eve computes $g^{ab} \pmod{p}$.

Question: If Eve can crack DH then Eve can compute Discrete Log. VOTE: Y, N, UNKNOWN TO SCIENCE.

Unknown to Science

Question: If Eve can crack DH then Eve can compute ???.

Hardness Assumption

Definition

Let DHF be the following function:

Input: p, g, g^a, g^b (note that a, b are not the input)

Outputs: g^{ab}.

Obvious Theorem: If Alice can crack Diffie-Hellman quickly then Alice can compute DHF quickly.

Hardness Assumption

Definition

Let DHF be the following function:

Input: p, g, g^a, g^b (note that a, b are not the input)

Outputs: g^{ab}.

Obvious Theorem: If Alice can crack Diffie-Hellman quickly then Alice can compute DHF quickly.

Hardness assumption: DHF is hard to compute.

Possible Futures

1. DL found to be easy, so DH is cracked
2. DHF found to be easy, so DH is cracked
3. Slightly better but still exp algorithms for DHF are found so Alice and Bob need to up their game, but DH still secure. (JNIP this is the most likely.

Possible Futures

1. DL found to be easy, so DH is cracked
2. DHF found to be easy, so DH is cracked
3. Slightly better but still exp algorithms for DHF are found so Alice and Bob need to up their game, but DH still secure. (JNIP this is the most likely. JNIP is IMHO shifted by 1.

Possible Futures

1. DL found to be easy, so DH is cracked
2. DHF found to be easy, so DH is cracked
3. Slightly better but still exp algorithms for DHF are found so Alice and Bob need to up their game, but DH still secure. (JNIP this is the most likely. JNIP is IMHO shifted by 1. IMHO is In My Humble Opinion.)
4. DHF proven to be hard. KOJQ unlikely in your lifetime.

Diffie-Hellman over Other Domains

Can do Diffie-Hellman with other structures that have these properties, that is, any Cyclic Group. In some cases this may be an advantage in that Eve's task is harder and Alice and Bob's task is not much harder.

Example: Elliptic Curve Diffie-Hellman (actually used).
Example: Braid Diffie-Hellman (not actually used).

Variants of Standard Diffie-Helman

Recall the Diffie-Helman Key Exchange

1. Alice: rand (p, g), p of length L, g gen for $\mathbb{Z}_p$. Arith mod p.
2. Alice sends (p, g) to Bob in the clear (Eve can see it).
3. Alice: rand $a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$, sends g^a.
4. Bob: rand $b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$, sends g^b.
5. Alice:$(g^b)^a = g^{ab}$. Bob:$(g^a)^b = g^{ab}$. g^{ab} is shared secret.

Why does Alice: rand $a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$.
Why not $a \in \{1, \ldots, p-1\}$? Discuss

Recall the Diffie-Helman Key Exchange

1. Alice: rand (p, g), p of length L, g gen for $\mathbb{Z}_p$. Arith mod p.
2. Alice sends (p, g) to Bob in the clear (Eve can see it).
3. Alice: rand $a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$, sends g^a.
4. Bob: rand $b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$, sends g^b.
5. Alice:$(g^b)^a = g^{ab}$. Bob:$(g^a)^b = g^{ab}$. g^{ab} is shared secret.

Why does Alice: rand $a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$.
Why not $a \in \{1, \ldots, p-1\}$? Discuss
If g is small and a is small then Eve can determine a from g^a.

Recall the Diffie-Helman Key Exchange

1. Alice: rand (p, g), p of length L, g gen for $\mathbb{Z}_p$. Arith mod p.
2. Alice sends (p, g) to Bob in the clear (Eve can see it).
3. Alice: rand $a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$, sends g^a.
4. Bob: rand $b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$, sends g^b.
5. Alice:$(g^b)^a = g^{ab}$. Bob:$(g^a)^b = g^{ab}$. g^{ab} is shared secret.

Why does Alice: rand $a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$.
Why not $a \in \{1, \ldots, p-1\}$? Discuss
If g is small and a is small then Eve can determine a from g^a.
But: Eve can compute $g^1, \ldots, g^L$ and if she sees any of those she knows.

Example

$p = 1013$
$g = 5$
$a = 6$
Eve computes ahead of time:
$5^0 = 1$
$5^1 = 5$
$5^2 = 25$
$5^3 = 125$
$5^4 = 625$
$5^5 = 86$
$5^6 = 430$
If Eve sees Alice 430 then she knows $a = 6$
Nothing special about a being small.

Example

$p = 1013$
$g = 40$
$a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\} = \{337, \ldots, 674\}$
Note: We assume that Eve KNOWS these endpoints.
Eve computes
$40^{337} \equiv 919$
$40^{338} \equiv 292$
$40^{339} \equiv 537$
$40^{340} \equiv 207$
$40^{341} \equiv 176$
$40^{342} \equiv 962$
$40^{343} \equiv 999$
If Eve sees Alice send any of $919, 292, 537, 207, 176, 962, 999$ then she knows a
g was big, a was big. Didn't help!

Example

$p = 1013$
$g = 40$
$a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\} = \{337, \ldots, 674\}$
Note: We assume that Eve KNOWS these endpoints.
Eve computes
$40^{337} \equiv 919$
$40^{338} \equiv 292$
$40^{339} \equiv 537$
$40^{340} \equiv 207$
$40^{341} \equiv 176$
$40^{342} \equiv 962$
$40^{343} \equiv 999$
If Eve sees Alice send any of $919, 292, 537, 207, 176, 962, 999$ then she knows a
g was big, a was big. Didn't help!
Of course, Eve has to get VERY LUCKY.

The Real Diffie-Helman

1. Alice finds a (p, g), p of length L, g gen for $\mathbb{Z}_p$. Arith mod p.
2. Alice sends (p, g) to Bob in the clear (Eve can see it).
3. Alice: rand $a \in \{1, \ldots, p-1\}$, sends g^a.
4. Bob: rand $b \in \{1, \ldots, p-1\}$, sends g^b.
5. Alice:$(g^b)^a = g^{ab}$. Bob:$(g^a)^b = g^{ab}$. g^{ab} is shared secret.

Eve comp $g^1, \ldots, g^L$. If $a \in \{1, \ldots, L\}$ Eve knows a.
Debatable Not really a problem:
Either

1. If L is small then Eve would have to get LUCKY to find a.
2. If L is large then Eve is doing LOTS OF computation.

Upshot: a, g small did not make attack much easier for Eve.

Is There Harm In Restricting a, b?

Does requiring $a, b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$ help?

Is There Harm In Restricting a, b?

Does requiring $a, b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$ help?

- Yes: Some obvious easy cases of DL are avoided.
- No: Eve can pre-compute any small number of cases anyway.

Is There Harm In Restricting a, b?

Does requiring $a, b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$ help?

- Yes: Some obvious easy cases of DL are avoided.
- No: Eve can pre-compute any small number of cases anyway.

Does requiring $a, b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$ hurt?

Is There Harm In Restricting a, b?

Does requiring $a, b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$ help?

- Yes: Some obvious easy cases of DL are avoided.
- No: Eve can pre-compute any small number of cases anyway.

Does requiring $a, b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$ hurt?
Key space is smaller, making it easier for Eve.

Is There Harm In Restricting a, b?

Does requiring $a, b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$ help?

- Yes: Some obvious easy cases of DL are avoided.
- No: Eve can pre-compute any small number of cases anyway.

Does requiring $a, b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$ hurt?
Key space is smaller, making it easier for Eve.
A matter of opinion. I think it helps. Others disagree.

How Important Is Public Key?

Used Everywhere

Public key is mostly used for giving out keys to be used for classical systems.
This makes the following work:

Used Everywhere

Public key is mostly used for giving out keys to be used for classical systems.
This makes the following work:

1. Amazon – Credit Cards

Used Everywhere

Public key is mostly used for giving out keys to be used for classical systems.
This makes the following work:

1. Amazon – Credit Cards
2. Ebay – Paypal

Used Everywhere

Public key is mostly used for giving out keys to be used for classical systems.
This makes the following work:

1. Amazon – Credit Cards
2. Ebay – Paypal
3. Facebook privacy –

Used Everywhere

Public key is mostly used for giving out keys to be used for classical systems.
This makes the following work:

1. Amazon – Credit Cards
2. Ebay – Paypal
3. Facebook privacy – just kidding, Facebook has no privacy.

Used Everywhere

Public key is mostly used for giving out keys to be used for classical systems.
This makes the following work:

1. Amazon – Credit Cards
2. Ebay – Paypal
3. Facebook privacy – just kidding, Facebook has no privacy.
4. Every financial institution in the world.

Used Everywhere

Public key is mostly used for giving out keys to be used for classical systems.
This makes the following work:

1. Amazon – Credit Cards
2. Ebay – Paypal
3. Facebook privacy – just kidding, Facebook has no privacy.
4. Every financial institution in the world.
5. Military – though less is known about this.

Turing Awards

The Turing Award is The Nobel Prize of Computer Science.

Given out every year.

We note when someone mentioned in Public Key Crypto won.

1. 1976- Michael Rabin
2. 1995- Manuel Blum
3. 2002- Ron Rivest, Shamir, Len Adelman
4. 2012- Silvio Micali, Shaffi Goldwasser
5. 2015- Whitfield Diffie, Martin Helman

Future: Oded Regev? Jon Katz?