Public Key Crypto: Math Needed and DH

Private-Key Ciphers

What do the following Private Key Encryption Schemes all have in common:

1. Shift Cipher
2. Affine Cipher
3. Vig Cipher
4. General Sub
5. Matrix Cipher
6. One-Time Pad

Private-Key Ciphers

What do the following Private Key Encryption Schemes all have in common:

1. Shift Cipher
2. Affine Cipher
3. Vig Cipher
4. General Sub
5. Matrix Cipher
6. One-Time Pad

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

Private-Key Ciphers

What do the following Private Key Encryption Schemes all have in common:

1. Shift Cipher
2. Affine Cipher
3. Vig Cipher
4. General Sub
5. Matrix Cipher
6. One-Time Pad

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

Private-Key Ciphers

What do the following Private Key Encryption Schemes all have in common:

1. Shift Cipher
2. Affine Cipher
3. Vig Cipher
4. General Sub
5. Matrix Cipher
6. One-Time Pad

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

Private-Key Ciphers

What do the following Private Key Encryption Schemes all have in common:

1. Shift Cipher
2. Affine Cipher
3. Vig Cipher
4. General Sub
5. Matrix Cipher
6. One-Time Pad

Alice and Bob need to meet! (Hence Private Key.)
Can Alice and Bob to establish a key without meeting?
Yes! And that is the key to public-key cryptography.
Aim: We present three such schemes: Diffie-Helman, ElGamal, and RSA. (Diffie-Helman is not quite an encryption scheme.)

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

What is Easy? What is Hard?

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

Examples

1. SAT on a formula with n variables seems to require $2^{O(n)}$ steps. We do not know this.
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!

What is Easy? What is Hard?

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

Examples

1. SAT on a formula with n variables seems to require $2^{O(n)}$ steps. We do not know this.
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 $\lg n$. We want (or don't want) algorithms polynomial in $\lg n$.

What We Count: We will count arithmetic operations as taking 1 time step. This could be an issue with enormous numbers.

Math Needed for Both Diffie-Helman and RSA

Notation

Let p be a prime.

1. $\mathbb{Z}_p$ is the numbers $\{0, \ldots, p-1\}$ with modular addition and multiplication.
2. $\mathbb{Z}_p^*$ is the numbers $\{1, \ldots, p-1\}$ with modular multiplication.

Exponentiation mod p

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

1. $x_0 = a$
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. $x_0 = a$
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? Its called First Attempt, so no.

Exponentiation mod p

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

First Attempt

1. $x_0 = a$
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? Its called First Attempt, so no.
Takes n steps and also x gets really large.

Exponentiation mod p

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

1. $x_0 = a$
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? Its called First Attempt, so no.
Takes n steps and also 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 this take for $a^n \ (p)$.

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 this take for $a^n \ (p)$.Answer: $\lg n$.

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 this take for $a^n \ (p)$.Answer: $\lg n$.

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

Repeated Squaring Algorithm

All arithmetic is mod p.

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

Number of operations: $O(\log n)$.

Diffie-Helman Key Exchange

Generators mod p

Lets take powers of 3 mod 7. All arithmetic is mod 7.
$3^0 \equiv 1$
$3^1 \equiv 3$
$3^2 \equiv 3 \times 3^1 \equiv 9 \equiv 2$
$3^3 \equiv 3 \times 3^2 \equiv 3 \times 2 \equiv 6$
$3^4 \equiv 3 \times 3^3 \equiv 3 \times 6 \equiv 18 \equiv 4$
$3^5 \equiv 3 \times 3^4 \equiv 3 \times 4 \equiv 12 \equiv 5$
$3^6 \equiv 3 \times 3^5 \equiv 3 \times 5 \equiv 15 \equiv 1$

$$\{3^0, 3^1, 3^2, 3^3, 3^4, 3^5, 3^6\} = \{1, 2, 3, 4, 5, 6\} \text{ Not in order}$$

Generators mod p

Lets take powers of 3 mod 7. All arithmetic is mod 7.

$3^0 \equiv 1$

$3^1 \equiv 3$

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

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

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

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

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

$$\{3^0, 3^1, 3^2, 3^3, 3^4, 3^5, 3^6\} = \{1, 2, 3, 4, 5, 6\} \text{ Not in order}$$

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

Generators mod p

Lets take powers of 3 mod 7. All arithmetic is mod 7.

$3^0 \equiv 1$
$3^1 \equiv 3$
$3^2 \equiv 3 \times 3^1 \equiv 9 \equiv 2$
$3^3 \equiv 3 \times 3^2 \equiv 3 \times 2 \equiv 6$
$3^4 \equiv 3 \times 3^3 \equiv 3 \times 6 \equiv 18 \equiv 4$
$3^5 \equiv 3 \times 3^4 \equiv 3 \times 4 \equiv 12 \equiv 5$
$3^6 \equiv 3 \times 3^5 \equiv 3 \times 5 \equiv 15 \equiv 1$

$$\{3^0, 3^1, 3^2, 3^3, 3^4, 3^5, 3^6\} = \{1, 2, 3, 4, 5, 6\} \text{ Not in order}$$

3 is a generator for $\mathbb{Z}_7$.
Definition: If p is a prime and $\{g^0, g^1, \ldots, g^{p-1}\} = \{1, \ldots, p-1\}$ then g is a generator for $\mathbb{Z}_p$.

Discrete Log-Example

Fact: 5 is a generator mod 73. All arithmetic is mod 73.
Discuss the following with your neighbor:

1. Find x such that $5^x \equiv 25$
2. Find x such that $5^x \equiv 26$

Discrete Log-Example

Fact: 5 is a generator mod 73. All arithmetic is mod 73.
Discuss the following with your neighbor:

1. Find x such that $5^x \equiv 25$
2. Find x such that $5^x \equiv 26$

1. Find x such that $5^x \equiv 25$. $x = 2$ obv works.
2. Find x such that $5^x \equiv 26$. Do not know. Could try computing $5^3, 5^4, \ldots,$ until you get 26. Might take ~ 70 steps.

The second problem seems hard.

Discrete Log-General

Definition Let p be a prime and g be a generator mod p.
The Discrete Log Problem is:
given y, find x such that $g^x = y$.

Discuss: Is this problem computationally hard?

Discrete Log-General

Definition Let p be a prime and g be a generator mod p.
The Discrete Log Problem is:
given y, find x such that $g^x = y$.

Discuss: Is this problem computationally hard?

1. If g, y are small so that then could be easy.
   Example: $7^x \equiv 49 \pmod{1009}$ is easy.
2. If g small, y large, then the problem is sometimes easy (HW).
3. If $g, y \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$ then problem suspected hard.
4. Obv alg: $O(p)$ steps. There is an $O(\sqrt{p})$ alg. Still too slow.

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.

Finding Generators

First Attempt at, given p, find a gen for $\mathbb{Z}_p$

1. Input p
2. For $g = 2$ to $p - 1$

   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: $\sim p/2$ g's are gens so $O(1)$ iterations.
CON: Computing $g^1, \ldots, g^{p-1}$ is $O(p \log p)$ operations.

Finding Generators

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

Second Attempt at, given 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 $g = 2$ to $p-1$

   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 Generators

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

Second Attempt at, given 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 $g = 2$ to $p-1$

   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. $|F| \leq \log p$ so okay.

Finding Generators

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

Second Attempt at, given 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 $g = 2$ to $p-1$

   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. $|F| \leq \log p$ so okay.
BIG CON: Factoring $p-1$? Really? Darn!

Finding Generators

Idea:Pick p such that $p-1=2q$ where q is prime.

Third Attempt at, given p, find a gen for $\mathbb{Z}_p$

1. Input p a prime such that $p-1=2q$ where q is prime.
2. Factor $p-1$. Let F be the set of its factors except $p-1$. Thats EASY: $F=\{2,q\}$.
3. For $g=2$ to $p-1$

   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 Generators

Idea: Pick p such that $p-1=2q$ where q is prime.
Third Attempt at, given p, find a gen for $\mathbb{Z}_p$

1. Input p a prime such that $p-1=2q$ where q is prime.
2. Factor $p-1$. Let F be the set of its factors except $p-1$.
   Thats EASY: $F=\{2,q\}$.
3. For $g=2$ to $p-1$

   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 Generators

Idea:Pick p such that $p-1=2q$ where q is prime.
Third Attempt at, given p, find a gen for $\mathbb{Z}_p$

1. Input p a prime such that $p-1=2q$ where q is prime.
2. Factor $p-1$. Let F be the set of its factors except $p-1$. Thats EASY: $F=\{2,q\}$.
3. For $g=2$ to $p-1$

   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: None. But need both p and $\frac{p-1}{2}$ are primes.

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, $2 \le i \le p-1$, 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, $2 \le i \le p-1$, then $\frac{p!}{i!(p-i)!} \in \mathbb{N}$ and is divisible by p.

Proof.

The expression is the answer to a question that has a $\mathbb{N}$ solution:

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

Since $\frac{p!}{i!(p-i)!} \in \mathbb{N}$, p divides the numerator, p does not divide the denominator, p divides the number.

$\square$

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

Primality Testing

Lemma

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

Lemma

p *prime,* $a \in \mathbb{N}$, $a^p \equiv a \pmod{p}$.

Proof.

Fix prime p. By induction on a. Base Case: $1^p \equiv 1$.
Ind Hyp: $a^p \equiv a \pmod{p}$
Ind Step:

$$(a+1)^p = \sum_{i=0}^{n} \binom{p}{i} a^i 1^{p-i} = \sum_{i=0}^{p} \binom{p}{i} a^i \equiv a^p + a^0 \equiv a + 1$$

□

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 random set R of $a \in \{2, \ldots, p-1\}$ of size $2\lceil \lg p \rceil$ (Could take $c\lceil \lg p \rceil$ for any c. Use $O(\lg p)$ so that this step is efficient.)
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 unluckily 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 $\leq \frac{1}{2^p}$. Good enough!
3. Exists deterministic poly time algorithm but is much slower.
4. n is a Carmichael Numbers 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 n, generate a prime of length n.

1. Input(n)
2. Pick $y \in \{0, 1\}^{n-1}$ at random.
3. $x = 1y$ (so x is a true n-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 n, generate a prime of length n.

1. Input(n)
2. Pick $y \in \{0,1\}^{n-1}$ at random.
3. $x = 1y$ (so x is a true n-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: NT tells us returns a prime within $3n^2$ tries with high prob.

Generating Primes (also needed for RSA)

Take as given: Primality Testing is FAST.

First Attempt at, given n, generate a prime of length n.

1. Input(n)
2. Pick $y \in \{0,1\}^{n-1}$ at random.
3. $x = 1y$ (so x is a true n-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: NT tells us returns a prime within $3n^2$ tries with high prob.
CON: None! Algorithm is fine! Can speed it up a bit (HW).

Generating Safe Primes

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

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

1. Input(n)
2. Pick $y \in \{0,1\}^{n-2}1$ at random.
3. $x = 1y$ (note that x is odd).
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 n, generate a safe prime of length n

1. Input(n)
2. Pick $y \in \{0,1\}^{n-2}1$ at random.
3. $x = 1y$ (note that x is odd).
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: NT 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 n, generate a safe prime of length n

1. Input(n)
2. Pick $y \in \{0,1\}^{n-2}1$ at random.
3. $x = 1y$ (note that x is odd).
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: NT tells us returns prime quickly with high prob.
CON: None. Algorithm is fine! Can speed it up a bit (HW).

The Diffie-Helman Key Exchange

Alice and Bob will share a secret s.

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

The Diffie-Helman Key Exchange

Alice and Bob will share a secret s.

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

PRO: Alice and Bob can execute the protocol easily.

The Diffie-Helman Key Exchange

Alice and Bob will share a secret s.

1. Alice finds a (p, g), p of length n, 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 picks random $a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Alice computes g^a and sends it to Bob in the clear (Eve can see it).
4. Bob picks random $b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Bob computes g^b and sends it to Alice in the clear (Eve can see it).
5. Alice computes $(g^b)^a = g^{ab}$.
6. Bob computes $(g^a)^b = g^{ab}$.
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-Helman Key Exchange

Alice and Bob will share a secret s.

1. Alice finds a (p, g), p of length n, 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 picks random $a \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Alice computes g^a and sends it to Bob in the clear (Eve can see it).
4. Bob picks random $b \in \{\frac{p}{3}, \ldots, \frac{2p}{3}\}$. Bob computes g^b and sends it to Alice in the clear (Eve can see it).
5. Alice computes $(g^b)^a = g^{ab}$.
6. Bob computes $(g^a)^b = g^{ab}$.
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?