Practical Aspects of Modern Cryptography

Winter 2011

Josh Benaloh
Brian LaMacchia

Cryptography is …

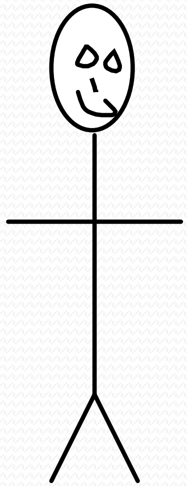
- Protecting Privacy of Data
- Authentication of Identities
- Preservation of Integrity

… basically any protocols designed to operate in an environment *absent* of universal trust.

Characters

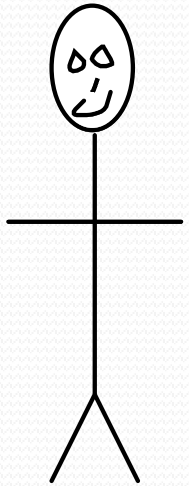
Characters

Alice



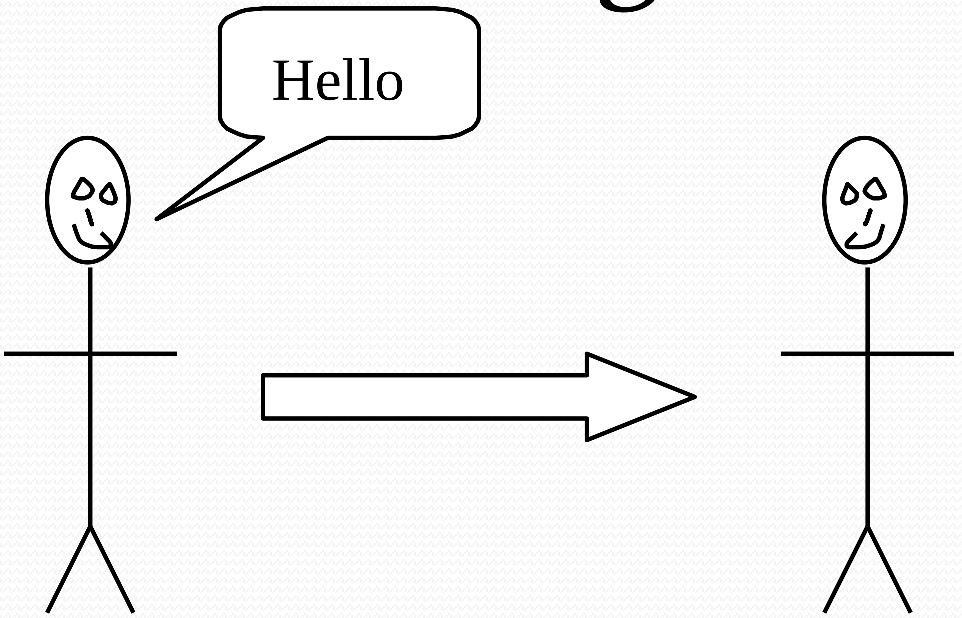
Characters

Bob



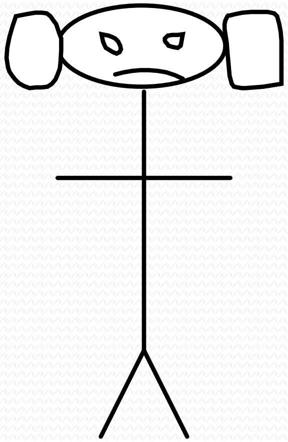
Basic Communication

Alice talking to Bob



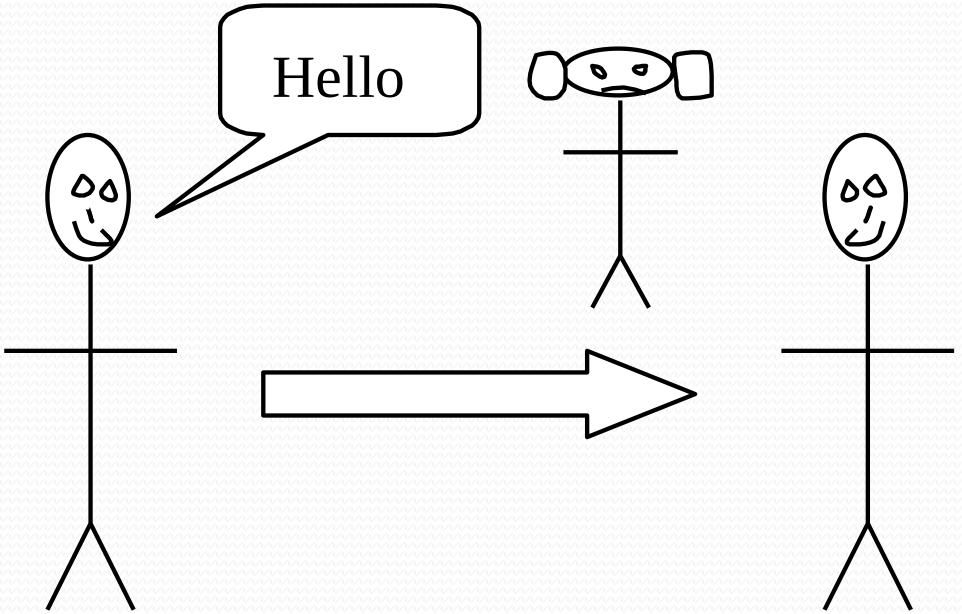
Another Character

Eve

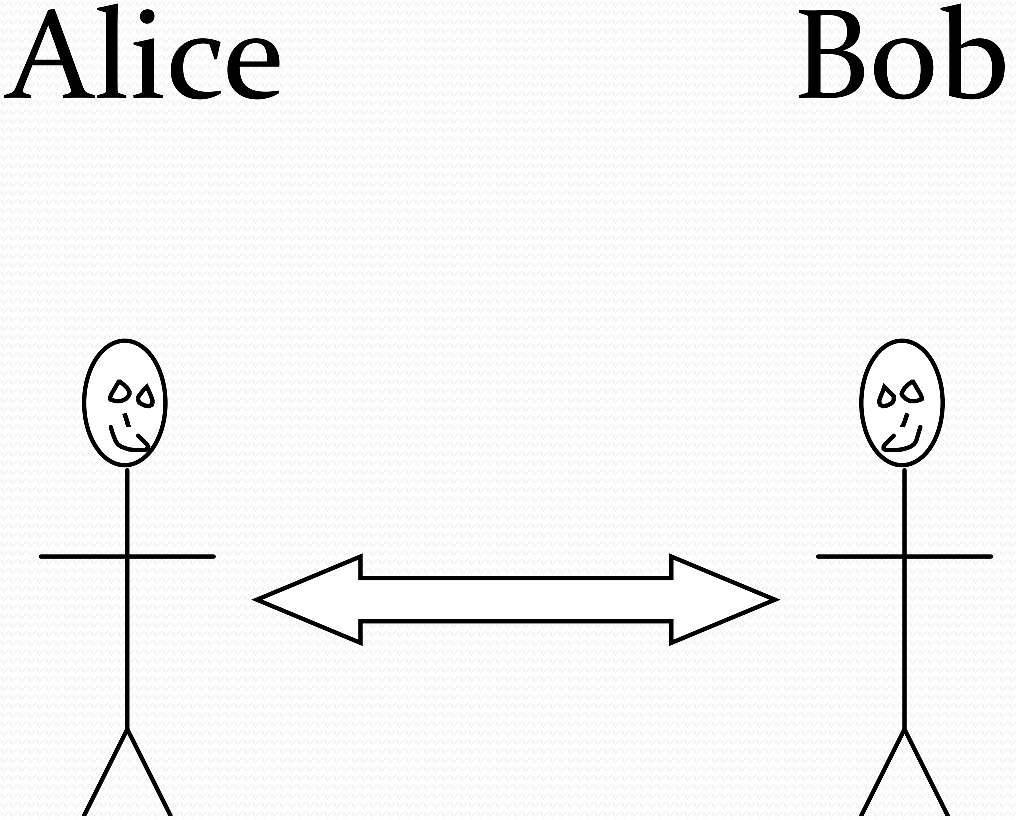


Basic Communication Problem

Eve listening to
Alice talking to Bob



Two-Party Environments



Remote Coin Flipping

- Alice and Bob decide to make a decision by flipping a coin.
- Alice and Bob are not in the same place.

Ground Rule

Protocol must be asynchronous.

- We cannot assume simultaneous actions.
- Players must take turns.

Is Remote Coin Flipping Possible?

Is Remote Coin Flipping Possible?

Two-part answer:

Is Remote Coin Flipping Possible?

Two-part answer:

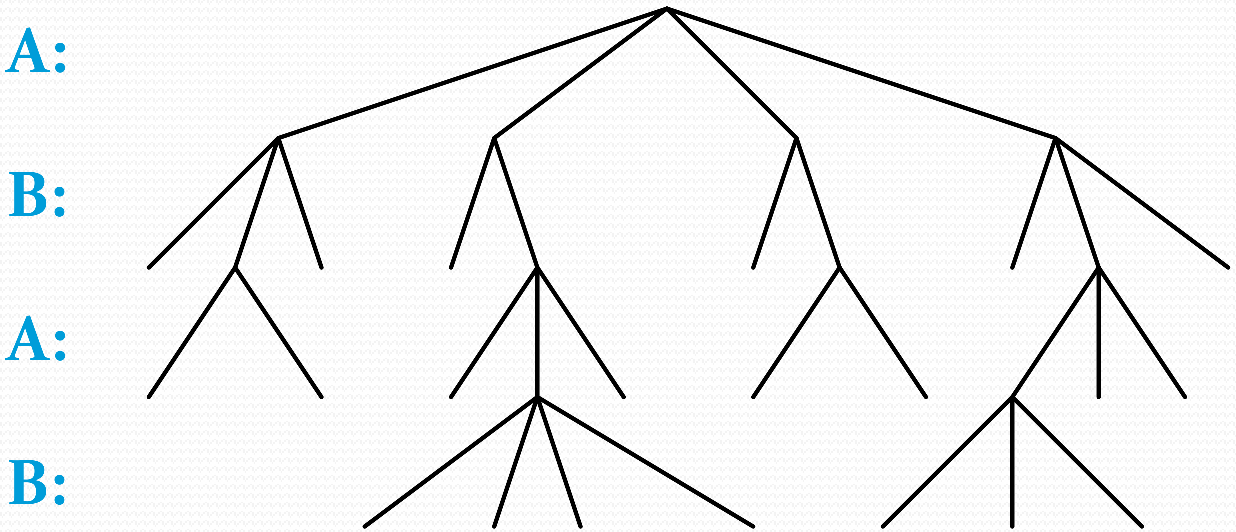
- NO – I will sketch a formal proof.

Is Remote Coin Flipping Possible?

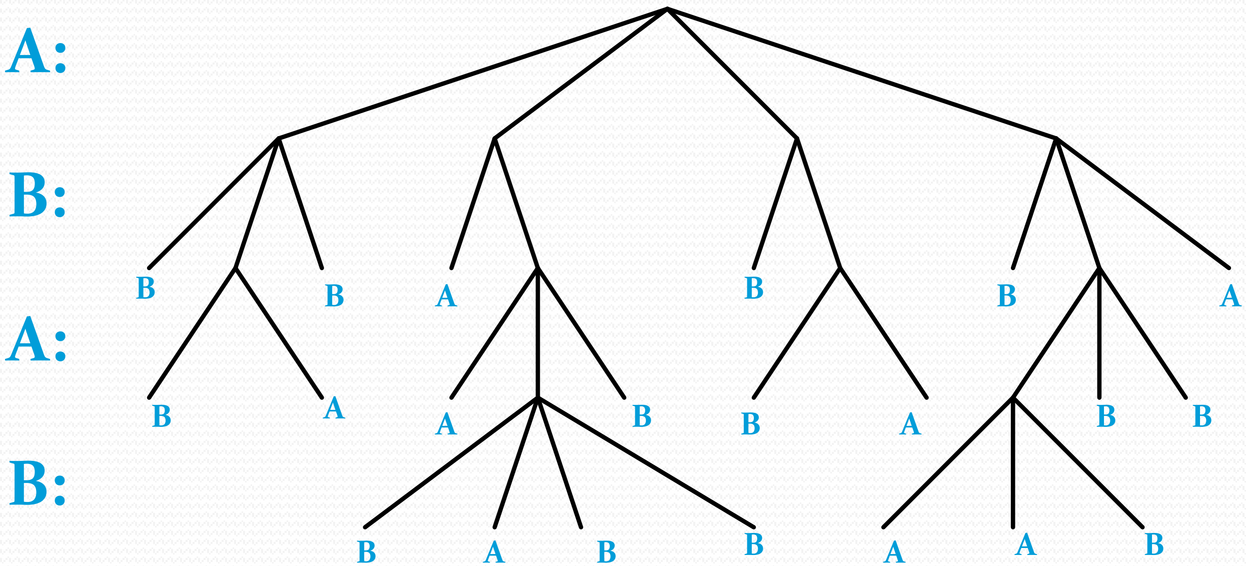
Two-part answer:

- NO – I will sketch a formal proof.
- YES – I will provide an effective protocol.

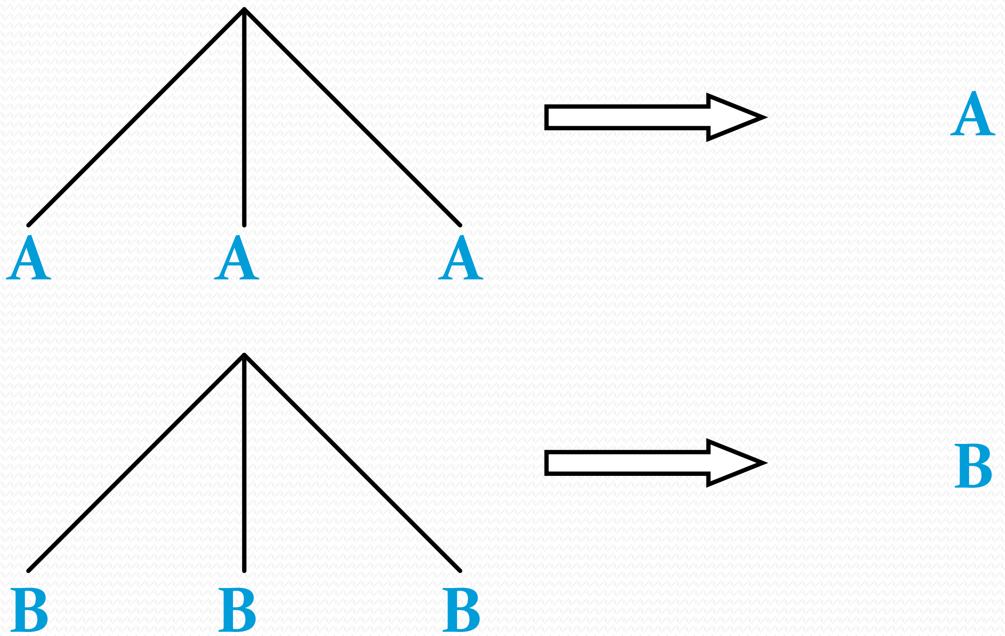
A Protocol Flow Tree



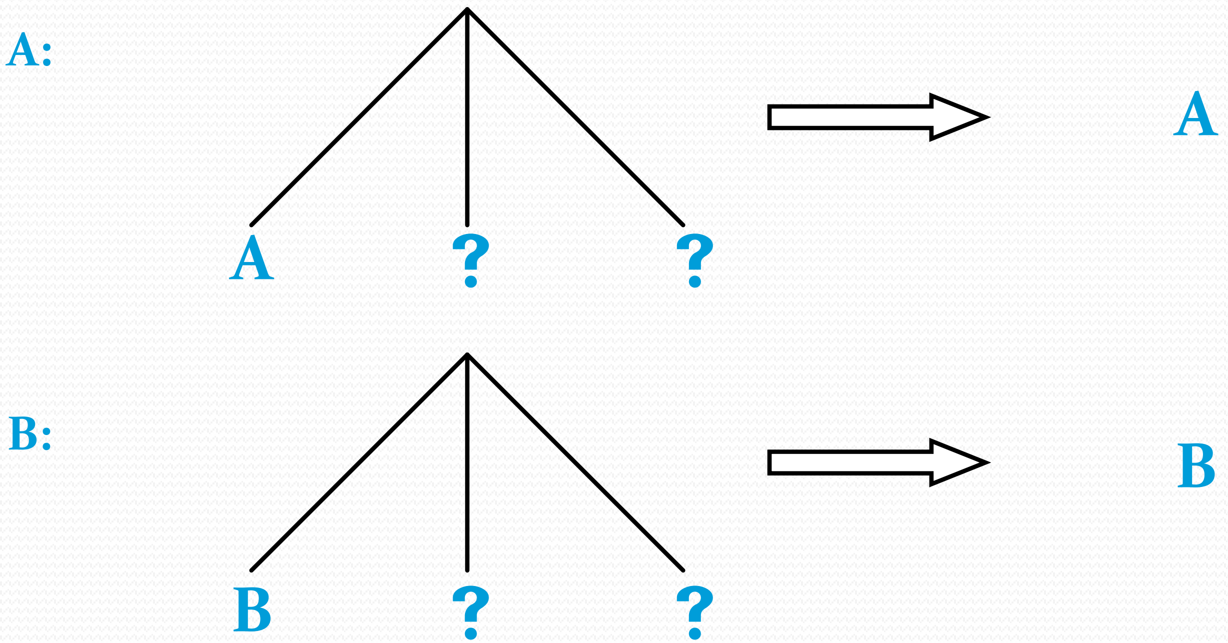
A Protocol Flow Tree



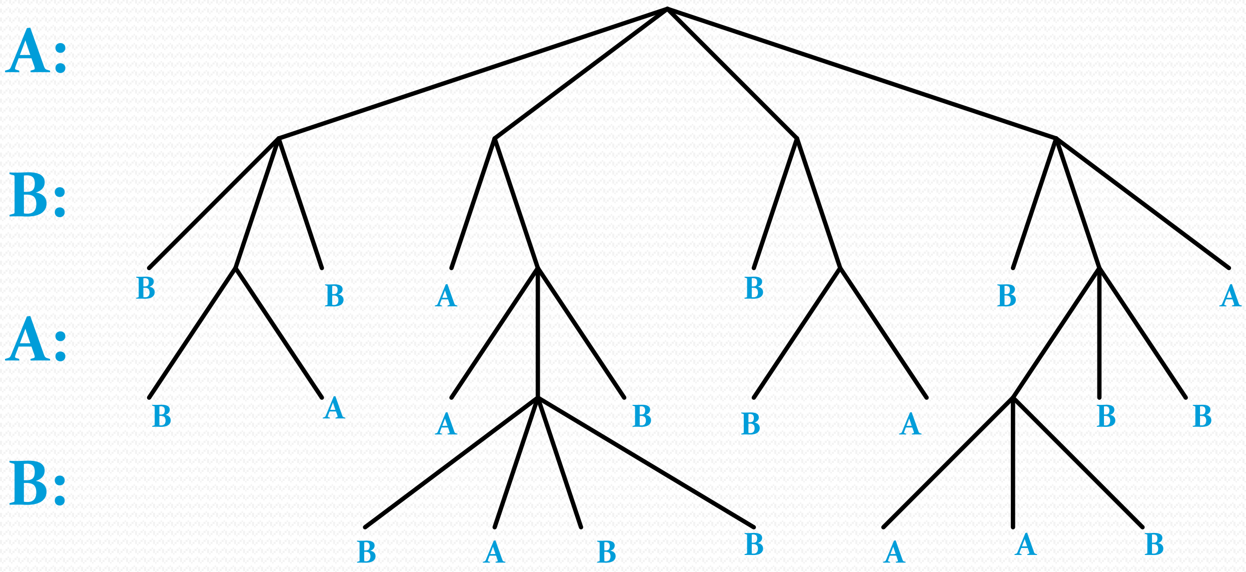
Pruning the Tree



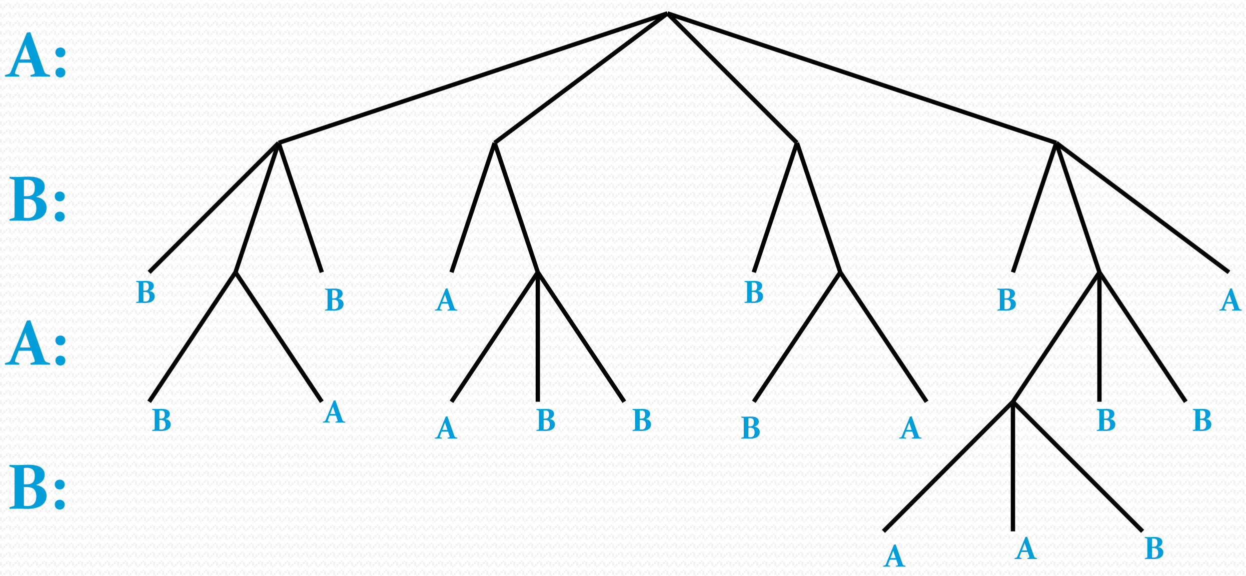
Pruning the Tree



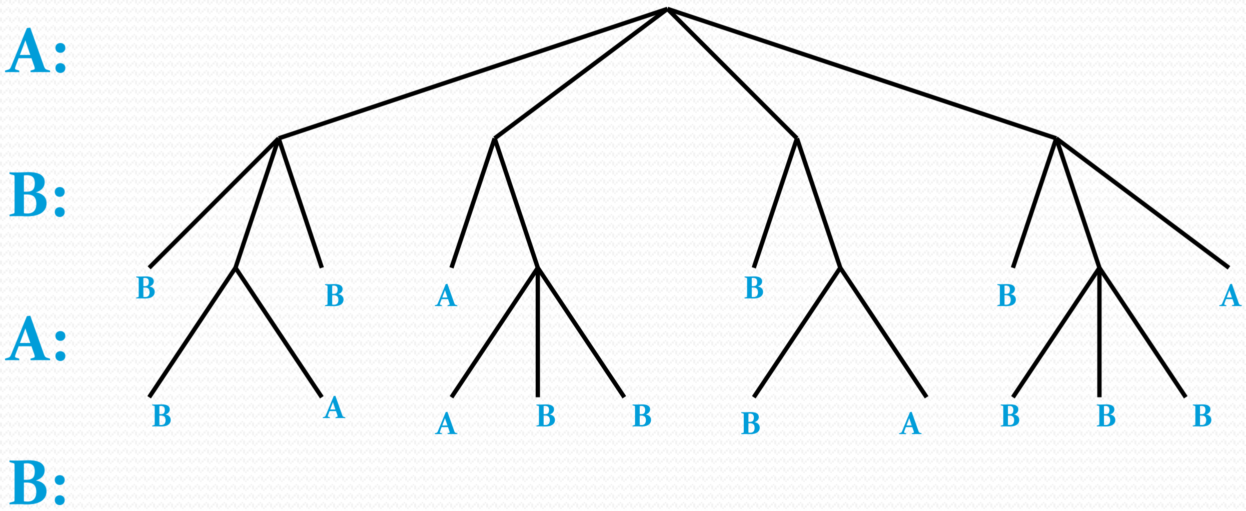
A Protocol Flow Tree



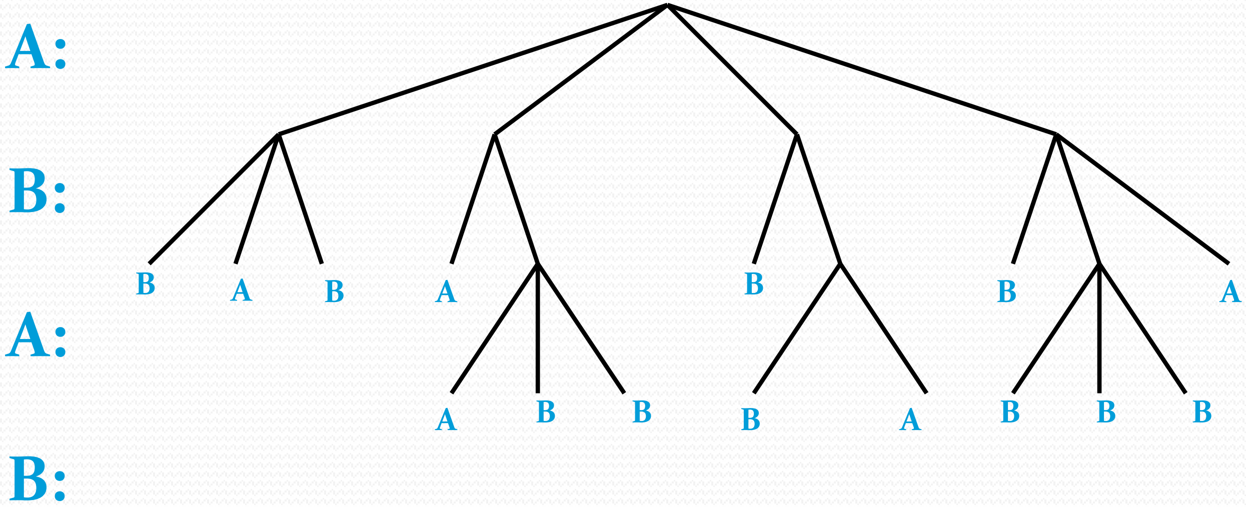
A Protocol Flow Tree



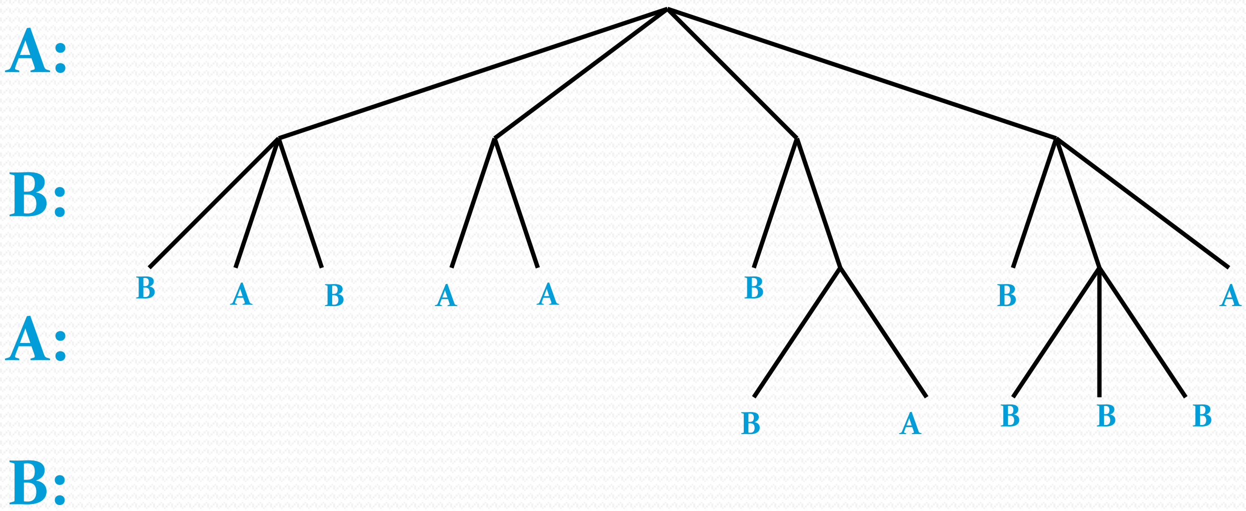
A Protocol Flow Tree



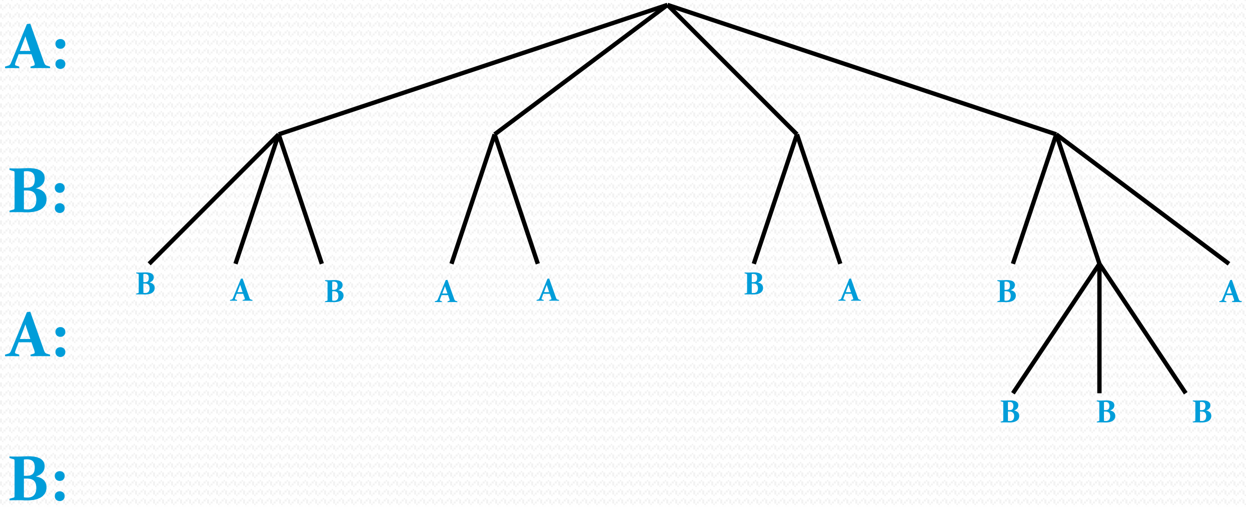
A Protocol Flow Tree



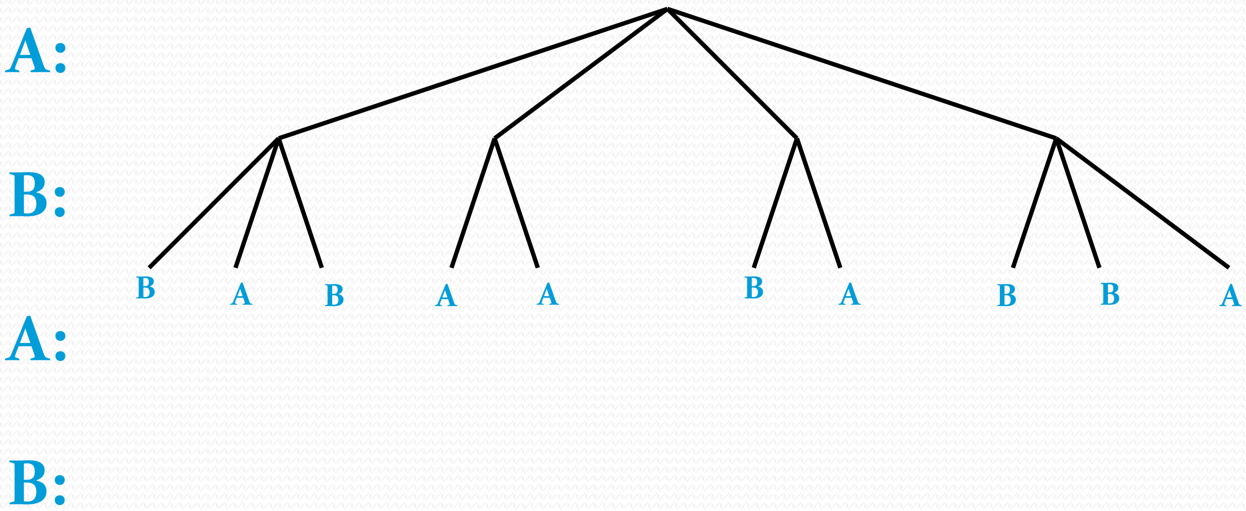
A Protocol Flow Tree



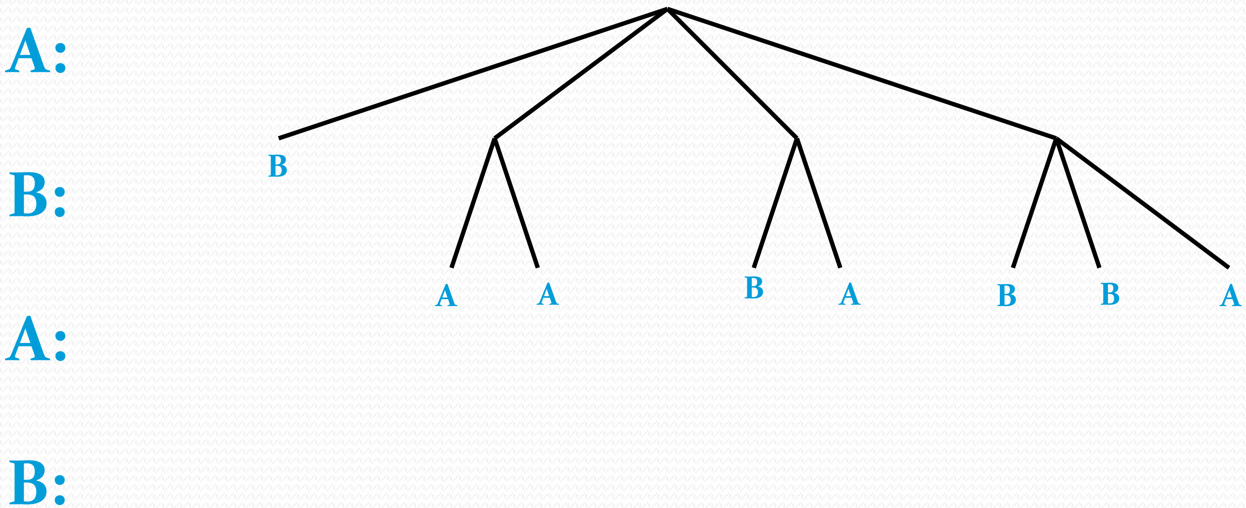
A Protocol Flow Tree



A Protocol Flow Tree



A Protocol Flow Tree



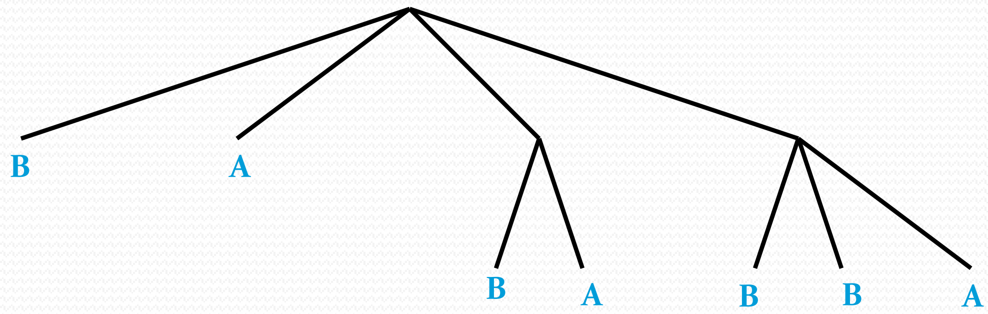
A Protocol Flow Tree

A:

B:

A:

B:



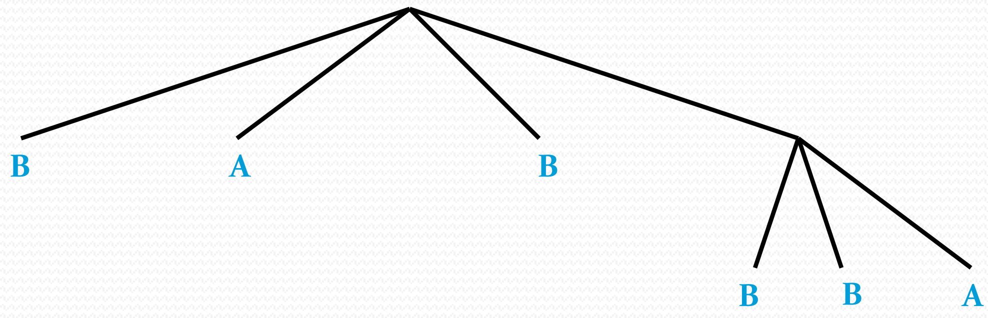
A Protocol Flow Tree

A:

B:

A:

B:



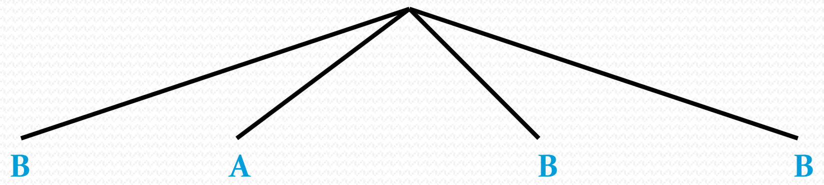
A Protocol Flow Tree

A:

B:

A:

B:



A Protocol Flow Tree

A

A:

B:

A:

B:

A Protocol Flow Tree

A

Completing the Pruning

When the pruning is complete one will end up with either

Completing the Pruning

When the pruning is complete one will end up with either

- a winner before the protocol has begun, or

Completing the Pruning

When the pruning is complete one will end up with either

- a winner before the protocol has begun, or
- a useless infinite game.

Conclusion of Part I

Remote coin flipping is utterly impossible!!!

How to Remotely Flip a Coin

How to Remotely Flip a Coin

The INTEGERS

How to Remotely Flip a Coin

<u>The INTEGERS</u>

0 4 8 12 16 …

How to Remotely Flip a Coin

The INTEGERS

0	4	8	12	16	…
1	5	9	13	17	…

How to Remotely Flip a Coin

The INTEGERS

0	4	8	12	16	…
1	5	9	13	17	…
2	6	10	14	18	…

How to Remotely Flip a Coin

The INTEGERS

0	4	8	12	16	…
1	5	9	13	17	…
2	6	10	14	18	…
3	7	11	15	19	…

How to Remotely Flip a Coin

The INTEGERS

Even

0	4	8	12	16	…
1	5	9	13	17	…
2	6	10	14	18	…
3	7	11	15	19	…

How to Remotely Flip a Coin

<u>The INTEGERS</u>

	0	**4**	**8**	**12**	**16 …**
$4n + 1$:	**1**	**5**	**9**	**13**	**17 …**
	2	**6**	**10**	**14**	**18 …**
$4n - 1$:	**3**	**7**	**11**	**15**	**19 …**

How to Remotely Flip a Coin

The INTEGERS

	0	4	8	12	16	…
Type +1:	1	5	9	13	17	…
	2	6	10	14	18	…
Type −1:	3	7	11	15	19	…

How to Remotely Flip a Coin

Fact 1

Multiplying two (odd) integers of the same type always yields a product of Type +1.

$$(4p+1)(4q+1) = 16pq+4p+4q+1 = 4(4pq+p+q)+1$$

$$(4p-1)(4q-1) = 16pq-4p-4q+1 = 4(4pq-p-q)+1$$

How to Remotely Flip a Coin

<u>Fact 2</u>

There is no known method (other than factoring) to distinguish a product of two "Type +1" integers from a product of two "Type –1" integers.

How to Remotely Flip a Coin

Fact 3

Factoring large integers is believed to be ***much*** harder than multiplying large integers.

How to Remotely Flip a Coin

How to Remotely Flip a Coin

Alice Bob

How to Remotely Flip a Coin

Alice

Bob

- Randomly select a bit $b \in \{\pm 1\}$ and two *large* integers P and Q – both of type b.

How to Remotely Flip a Coin

Alice	Bob

- Randomly select a bit $b \in \{\pm 1\}$ and two *large* integers P and Q – both of type b.
- Compute $N = PQ$.

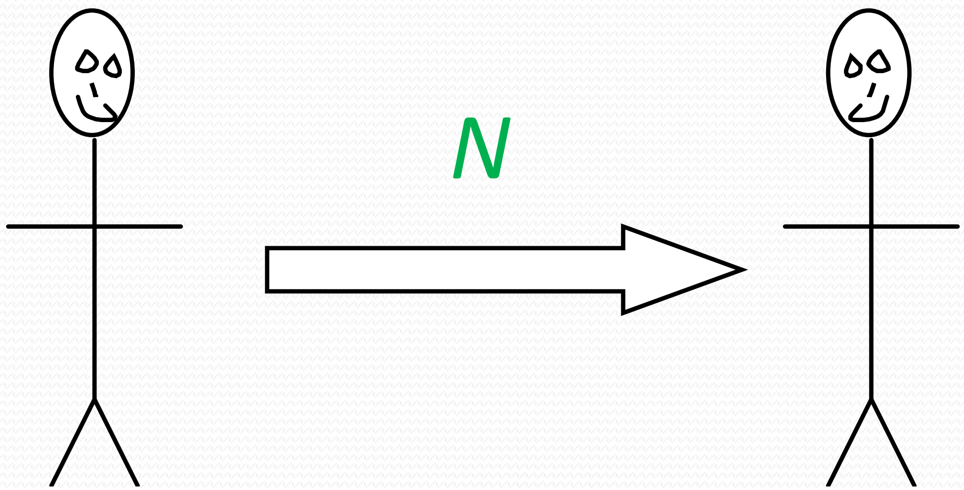
How to Remotely Flip a Coin

Alice | Bob

- Randomly select a bit $b \in \{\pm 1\}$ and two *large* integers P and Q – both of type b.
- Compute $N = PQ$.
- Send N to Bob.

How to Remotely Flip a Coin

Alice Bob



How to Remotely Flip a Coin

Alice | Bob

- Randomly select a bit $b \in \{\pm 1\}$ and two *large* integers P and Q – both of type b.
- Compute $N = PQ$.
- Send N to Bob.

How to Remotely Flip a Coin

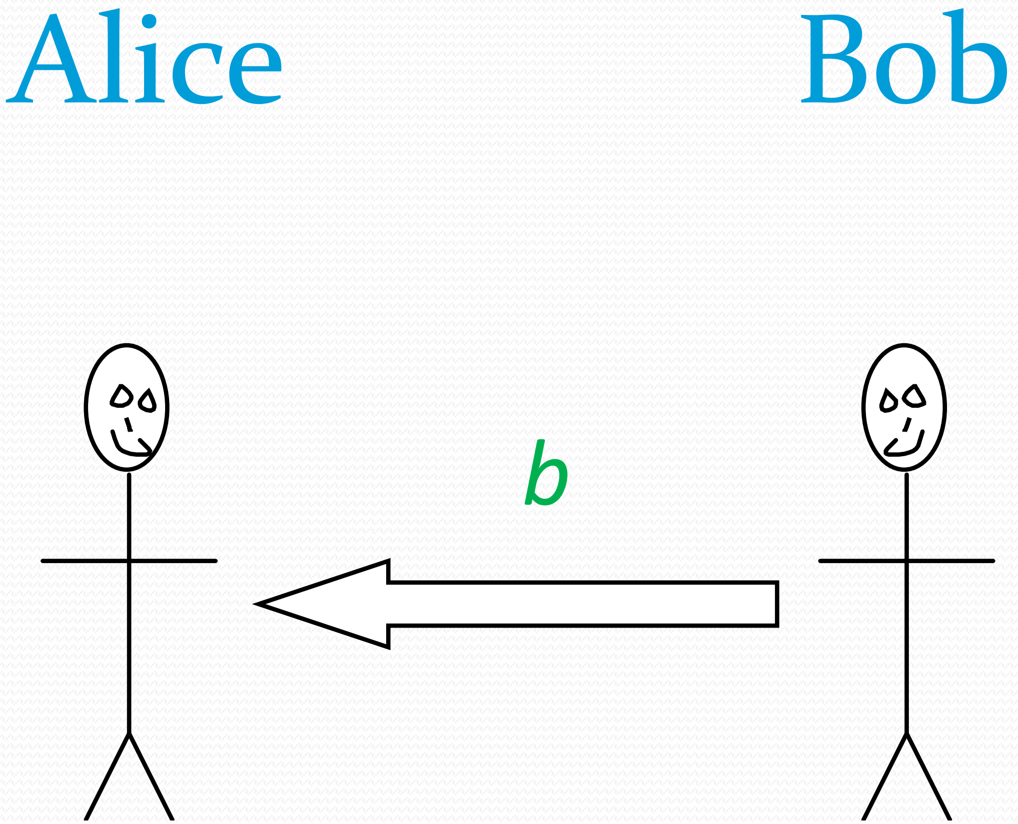
Alice

- Randomly select a bit $b \in \{\pm 1\}$ and two *large* integers P and Q – both of type b.
- Compute $N = PQ$.
- Send N to Bob.

Bob

- After receiving N from Alice, guess the value of b and send this guess to Alice.

How to Remotely Flip a Coin



How to Remotely Flip a Coin

Alice

- Randomly select a bit $b \in \{\pm 1\}$ and two *large* integers P and Q – both of type b.
- Compute $N = PQ$.
- Send N to Bob.

Bob

- After receiving N from Alice, guess the value of b and send this guess to Alice.

How to Remotely Flip a Coin

Alice

- Randomly select a bit $b \in \{\pm 1\}$ and two *large* integers P and Q – both of type b.
- Compute $N = PQ$.
- Send N to Bob.

Bob

- After receiving N from Alice, guess the value of b and send this guess to Alice.

Bob wins if and only if he correctly guesses the value of b.

How to Remotely Flip a Coin

Alice

- Randomly select a bit $b \in \{\pm 1\}$ and two *large* integers P and Q – both of type b.
- Compute $N = PQ$.
- Send N to Bob.

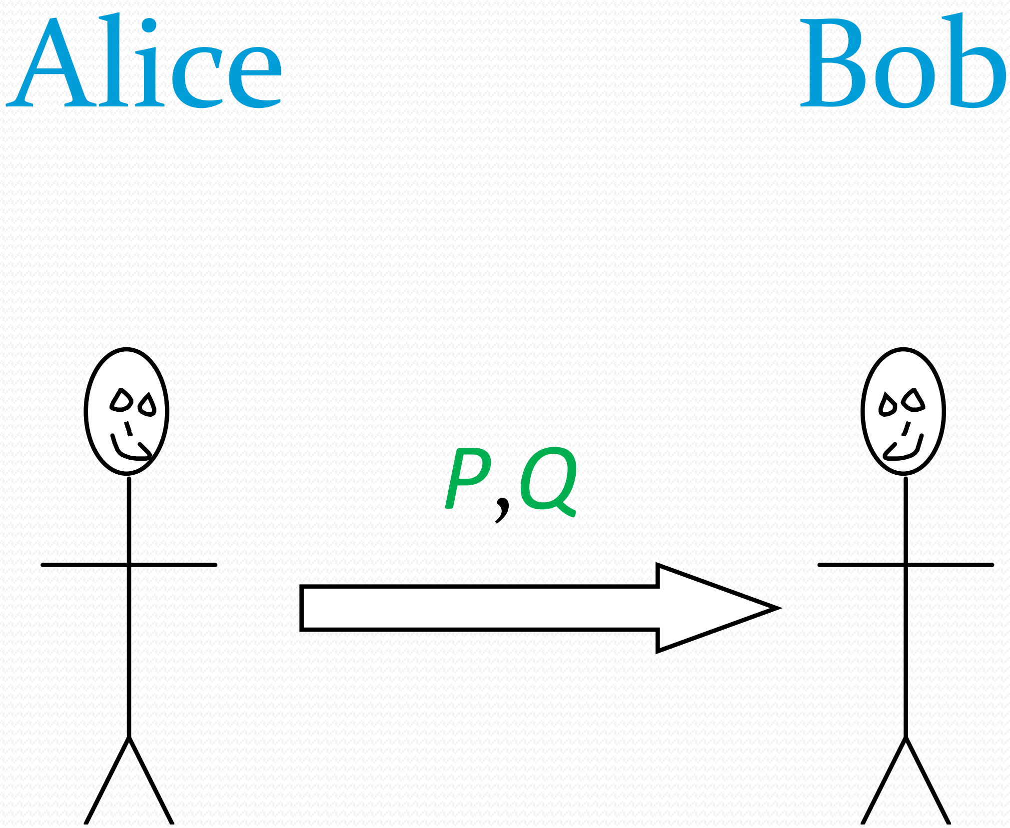
After receiving b from Bob, reveal P and Q.

Bob

- After receiving N from Alice, guess the value of b and send this guess to Alice.

Bob wins if and only if he correctly guesses the value of b.

How to Remotely Flip a Coin



How to Remotely Flip a Coin

Alice

- Randomly select a bit $b \in \{\pm 1\}$ and two *large* integers P and Q – both of type b.
- Compute $N = PQ$.
- Send N to Bob.

After receiving b from Bob, reveal P and Q.

Bob

- After receiving N from Alice, guess the value of b and send this guess to Alice.

Bob wins if and only if he correctly guesses the value of b.

Let's Play

<u>The INTEGERS</u>

	0	4	8	12	16	…
Type +1:	1	5	9	13	17	…
	2	6	10	14	18	…
Type −1:	3	7	11	15	19	…

How to Remotely Flip a Coin

Alice

- Randomly select a bit $b \in \{\pm 1\}$ and two *large* integers P and Q – both of type b.
- Compute $N = PQ$.
- Send N to Bob.

After receiving b from Bob, reveal P and Q.

Bob

- After receiving N from Alice, guess the value of b and send this guess to Alice.

Bob wins if and only if he correctly guesses the value of b.

How to Remotely Flip a Coin

Alice

- Randomly select a bit $b \in \{\pm 1\}$ and two *large primes* P and Q – both of type b.
- Compute $N = PQ$.
- Send N to Bob.

After receiving b from Bob, reveal P and Q.

Bob

- After receiving N from Alice, guess the value of b and send this guess to Alice.

Bob wins if and only if he correctly guesses the value of b.

Checking Primality

Basic result from group theory –

If p is a prime, then for integers a such that $0 < a < p$, then $a^{p-1} \bmod p = 1$.

This is almost never true when p is composite.

How are the Answers Reconciled?

How are the Answers Reconciled?

- The impossibility proof assumed unlimited computational ability.

How are the Answers Reconciled?

- The impossibility proof assumed unlimited computational ability.
- The protocol is not 50/50 – Bob has a small advantage.

Applications of Remote Flipping

- Remote Card Playing
- Internet Gambling
- Various “Fair” Agreement Protocols

Bit Commitment

We have implemented remote coin flipping via *bit commitment.*

Commitment protocols can also be used for

- Sealed bidding
- Undisclosed contracts
- Authenticated predictions

One-Way Functions

We have implemented bit commitment via *one-way functions.*

One-way functions can be used for

- Authentication
- Data integrity
- Strong “randomness”

One-Way Functions

One-Way Functions

Two basic classes of one-way functions

One-Way Functions

Two basic classes of one-way functions

- Mathematical

One-Way Functions

Two basic classes of one-way functions

- Mathematical
  - Multiplication: $Z=X\times Y$

One-Way Functions

Two basic classes of one-way functions

- Mathematical
  - Multiplication: $Z=X\times Y$
  - Modular Exponentiation: $Z = Y^X \bmod N$

One-Way Functions

Two basic classes of one-way functions

- Mathematical
  - Multiplication: $Z = X \times Y$
  - Modular Exponentiation: $Z = Y^X \bmod N$
- Ugly

The Fundamental Equation

$$Z=Y^X \bmod N$$

Modular Arithmetic

Modular Arithmetic

Z mod N is the integer remainder when Z is divided by N.

Modular Arithmetic

Z mod N is the integer remainder when Z is divided by N.

<u>The Division Theorem</u>

For all integers Z and N>0, there exist unique integers Q and R such that $Z = Q \times N + R$ and $0 \leq R < N$.

Modular Arithmetic

$Z \bmod N$ is the integer remainder when Z is divided by N.

<u>The Division Theorem</u>

For all integers Z and $N>0$, there exist unique integers Q and R such that $Z = Q \times N + R$ and $0 \leq R < N$.

By definition, this unique $R = Z \bmod N$.

Modular Arithmetic

- To compute (A+B) mod N,
  compute (A+B) and take the result mod N.

Modular Arithmetic

- To compute (A+B) mod N,
  compute (A+B) and take the result mod N.
- To compute (A-B) mod N,
  compute (A-B) and take the result mod N.

Modular Arithmetic

- To compute $(A+B) \bmod N$,
  compute $(A+B)$ and take the result $\bmod N$.
- To compute $(A-B) \bmod N$,
  compute $(A-B)$ and take the result $\bmod N$.
- To compute $(A\times B) \bmod N$,
  compute $(A\times B)$ and take the result $\bmod N$.

Modular Arithmetic

- To compute $(A+B) \bmod N$,
  compute $(A+B)$ and take the result $\bmod N$.
- To compute $(A-B) \bmod N$,
  compute $(A-B)$ and take the result $\bmod N$.
- To compute $(A\times B) \bmod N$,
  compute $(A\times B)$ and take the result $\bmod N$.
- To compute $(A\div B) \bmod N$, …

Modular Division

Modular Division

What is the value of $(1 \div 2) \bmod 7$?

We need a solution to $2x \bmod 7 = 1$.

Modular Division

What is the value of $(1 \div 2) \bmod 7$?

We need a solution to $2x \bmod 7 = 1$.

Try $x = 4$.

Modular Division

What is the value of $(1 \div 2) \bmod 7$?

We need a solution to $2x \bmod 7 = 1$.

Try $x = 4$.

What is the value of $(7 \div 5) \bmod 11$?

We need a solution to $5x \bmod 11 = 7$.

Modular Division

What is the value of $(1 \div 2) \bmod 7$?

We need a solution to $2x \bmod 7 = 1$.

Try $x = 4$.

What is the value of $(7 \div 5) \bmod 11$?

We need a solution to $5x \bmod 11 = 7$.

Try $x = 8$.

Modular Division

Modular Division

Is modular division always well-defined?

Modular Division

Is modular division always well-defined?

$(1 \div 3) \bmod 6 =$?

Modular Division

Is modular division always well-defined?

$(1 \div 3) \bmod 6 =$?

$3x \bmod 6 = 1$ has no solution!

Modular Division

Is modular division always well-defined?

$(1 \div 3) \bmod 6 =$?

$3x \bmod 6 = 1$ has no solution!

<u>Fact</u>

$(A \div B) \bmod N$ always has a solution when $\gcd(B,N) = 1$.

Modular Division

<u>Fact 1</u>

$(A \div B) \bmod N$ always has a solution when $\gcd(B,N) = 1$.

Modular Division

Fact 1

$(A \div B) \bmod N$ always has a solution when $\gcd(B,N) = 1$.

Fact 2

$(A \div B) \bmod N$ never has a solution when $\gcd(A,B) = 1$ and $\gcd(B,N) \neq 1$.

Greatest Common Divisors

Greatest Common Divisors

$\gcd(A, B) = \gcd(B, A - B)$

Greatest Common Divisors

gcd(A , B) = gcd(B , A – B)

since any common factor of A and B

is also a factor of A – B

and

since any common factor of B and A – B

is also a factor of A.

Greatest Common Divisors

gcd(A , B) = gcd(B , A – B)

gcd(21,12) = gcd(12,9) = gcd(9,3)

= gcd(3,6) = gcd(6,3) = gcd(3,3)

= gcd(3,0) = 3

Greatest Common Divisors

$\gcd(A, B) = \gcd(B, A - B)$

Greatest Common Divisors

$\gcd(A, B) = \gcd(B, A - B)$

$\gcd(A, B) = \gcd(B, A - kB)$ for any integer k.

Greatest Common Divisors

gcd(A , B) = gcd(B , A – B)

gcd(A , B) = gcd(B , A – kB) for any integer k.

gcd(A , B) = gcd(B , A mod B)

Greatest Common Divisors

$\gcd(A, B) = \gcd(B, A - B)$

$\gcd(A, B) = \gcd(B, A - kB)$ for any integer k.

$\gcd(A, B) = \gcd(B, A \bmod B)$

$\gcd(21,12) = \gcd(12,9) = \gcd(9,3)$

$= \gcd(3,0) = 3$

Extended Euclidean Algorithm

Given integers A and B, find integers X and Y such that $AX + BY = \gcd(A,B)$.

Extended Euclidean Algorithm

Given integers A and B, find integers X and Y such that $AX + BY = \gcd(A,B)$.

When $\gcd(A,B) = 1$, solve $AX \bmod B = 1$, by finding X and Y such that

$$AX + BY = \gcd(A,B) = 1.$$

Extended Euclidean Algorithm

Given integers A and B, find integers X and Y such that $AX + BY = \gcd(A,B)$.

When $\gcd(A,B) = 1$, solve $AX \bmod B = 1$, by finding X and Y such that

$$AX + BY = \gcd(A,B) = 1.$$

Compute $(C \div A) \bmod B$ as $C \times (1 \div A) \bmod B$.

Extended Euclidean Algorithm

gcd(35, 8) =

gcd(8, 35 mod 8) = gcd(8, 3) =

gcd(3, 8 mod 3) = gcd(3, 2) =

gcd(2, 3 mod 2) = gcd(2, 1) =

gcd(1, 2 mod 1) = gcd(1, 0) = 1

Extended Euclidean Algorithm

$35 = 8 \times 4 + 3$

Extended Euclidean Algorithm

$$35 = 8 \times 4 + 3$$

$$8 = 3 \times 2 + 2$$

Extended Euclidean Algorithm

$$35 = 8 \times 4 + 3$$

$$8 = 3 \times 2 + 2$$

$$3 = 2 \times 1 + 1$$

Extended Euclidean Algorithm

$$35 = 8 \times 4 + 3$$

$$8 = 3 \times 2 + 2$$

$$3 = 2 \times 1 + 1$$

$$2 = 1 \times 2 + 0$$

Extended Euclidean Algorithm

$$35 = 8 \times 4 + 3 \qquad\qquad 3 = 35 - 8 \times 4$$

$$8 = 3 \times 2 + 2 \qquad\qquad 2 = 8 - 3 \times 2$$

$$3 = 2 \times 1 + 1 \qquad\qquad 1 = 3 - 2 \times 1$$

$$2 = 1 \times 2 + 0$$

Extended Euclidean Algorithm

$$3 = 35 - 8 \times 4$$

$$2 = 8 - 3 \times 2$$

$$1 = 3 - 2 \times 1$$

Extended Euclidean Algorithm

$$3 = 35 - 8 \times 4$$

$$2 = 8 - 3 \times 2$$

$$1 = 3 - 2 \times 1 = (35 - 8 \times 4) - (8 - 3 \times 2) \times 1$$

Extended Euclidean Algorithm

$\mathbf{3 = 35 - 8 \times 4}$

$\mathbf{2 = 8 - 3 \times 2}$

$\mathbf{1 = 3 - 2 \times 1 = (35 - 8 \times 4) - (8 - 3 \times 2) \times 1 = (35 - 8 \times 4) - (8 - (35 - 8 \times 4) \times 2) \times 1}$

Extended Euclidean Algorithm

$3 = 35 - 8 \times 4$

$2 = 8 - 3 \times 2$

$1 = 3 - 2 \times 1 = (35 - 8 \times 4) - (8 - 3 \times 2) \times 1 = (35 - 8 \times 4) - (8 - (35 - 8 \times 4) \times 2) \times 1 = 35 \times 3 - 8 \times 13$

Extended Euclidean Algorithm

Given $A, B > 0$, set $x_1 = 1$, $x_2 = 0$, $y_1 = 0$, $y_2 = 1$, $a_1 = A$, $b_1 = B$, $i = 1$.

Repeat while $b_i > 0$: $\{i = i + 1$;

$q_i = a_{i-1} \text{div } b_{i-1}$; $b_i = a_{i-1} - q_i b_{i-1}$; $a_i = b_{i-1}$;

$x_{i+1} = x_{i-1} - q_1 x_i$; $y_{i+1} = y_{i-1} - q_1 y_i\}$.

For all i: $Ax_i + By_i = a_i$. Final $a_i = \text{gcd}(A,B)$.

If $a_i = 1$, then $x_i = A^{-1} \bmod B$ and $y_i = B^{-1} \bmod A$.

The Fundamental Equation

$$Z=Y^X \bmod N$$

The Fundamental Equation

$$Z = Y^X \bmod N$$

When Z is unknown, it can be efficiently computed.

The Fundamental Equation

$$Z = Y^X \bmod N$$

When X is unknown, the problem is known as the *discrete logarithm* and is generally believed to be hard to solve.

The Fundamental Equation

$$Z=Y^X \bmod N$$

When Y is unknown, the problem is known as *discrete root finding* and is generally believed to be hard to solve...

The Fundamental Equation

$$Z = Y^X \bmod N$$

… *unless* the factorization of N is known.

The Fundamental Equation

$$Z = Y^X \bmod N$$

The problem is not well-studied for the case when N is unknown.

Implementation

$$Z = Y^X \bmod N$$

How to compute $Y^X \bmod N$

How to compute Y^X mod N

Compute Y^X and then reduce mod N.

How to compute Y^X mod N

<u>Compute Y^X and then reduce mod N.</u>

- If X, Y, and N each are 2,048-bit integers, Y^X consists of ~2^{2059} bits.

How to compute Y^X mod N

<u>Compute Y^X and then reduce mod N.</u>

- If X, Y, and N each are 2,048-bit integers, Y^X consists of ~2^{2059} bits.
- Since there are roughly 2^{250} particles in the universe, storage is a problem.

How to compute $Y^X \bmod N$

How to compute Y^X mod N

- Repeatedly multiplying by Y (followed each time by a reduction modulo N) X times solves the storage problem.

How to compute $Y^X \bmod N$

- Repeatedly multiplying by Y (followed each time by a reduction modulo N) X times solves the storage problem.
- However, we would need to perform ~2^{900} 64-bit multiplications per second to complete the computation before the sun burns out.

How to compute Y^X mod N

How to compute Y^X mod N

Multiplication by Repeated Doubling

How to compute Y^X mod N

<u>Multiplication by Repeated Doubling</u>

To compute $X \times Y$,

How to compute Y^X mod N

<u>Multiplication by Repeated Doubling</u>

To compute $X \times Y$,

compute $Y, 2Y, 4Y, 8Y, 16Y, \ldots$

How to compute Y^X mod N

<u>Multiplication by Repeated Doubling</u>

To compute $X \times Y$,

compute $Y, 2Y, 4Y, 8Y, 16Y, \ldots$

and sum up those values dictated by the binary representation of X.

How to compute Y^X mod N

<u>Multiplication by Repeated Doubling</u>

To compute $X \times Y$,

compute $Y, 2Y, 4Y, 8Y, 16Y, \ldots$

and sum up those values dictated by the binary representation of X.

<u>Example</u>: $26Y = 2Y + 8Y + 16Y$.

How to compute $Y^X \bmod N$

How to compute Y^X mod N

Exponentiation by Repeated Squaring

How to compute Y^X mod N

<u>Exponentiation by Repeated Squaring</u>

To compute Y^X,

How to compute Y^X mod N

<u>Exponentiation by Repeated Squaring</u>

To compute Y^X,

compute $Y, Y^2, Y^4, Y^8, Y^{16}, \ldots$

How to compute Y^X mod N

<u>Exponentiation by Repeated Squaring</u>

To compute Y^X,

compute $Y, Y^2, Y^4, Y^8, Y^{16}, \ldots$

and multiply those values dictated by the binary representation of X.

How to compute Y^X mod N

<u>Exponentiation by Repeated Squaring</u>

To compute Y^X,

compute $Y, Y^2, Y^4, Y^8, Y^{16}, \ldots$

and multiply those values dictated by the binary representation of X.

<u>Example</u>: $Y^{26} = Y^2 \times Y^8 \times Y^{16}$.

How to compute Y^X mod N

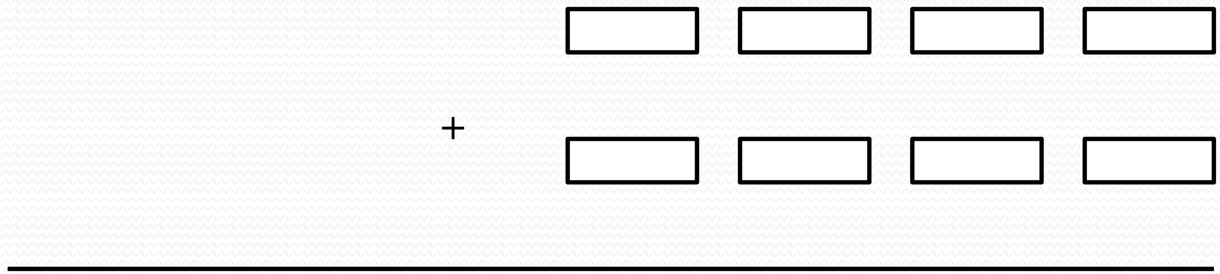
We can now perform a 2,048-bit modular exponentiation using ~3,072 2,048-bit modular multiplications.

- 2,048 squarings: $y, y^2, y^4, \ldots, y^{2^{2048}}$
- ~1024 "ordinary" multiplications

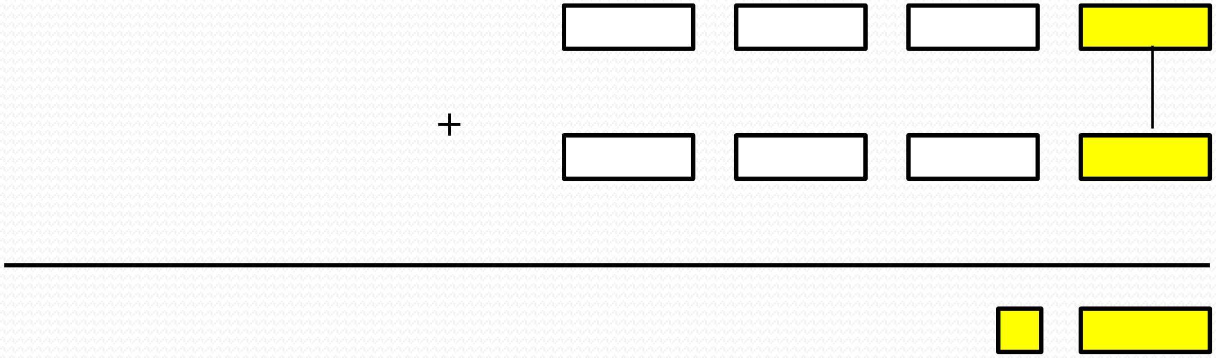
Large-Integer Operations

- Addition and Subtraction
- Multiplication
- Division and Remainder (Mod N)
- Exponentiation

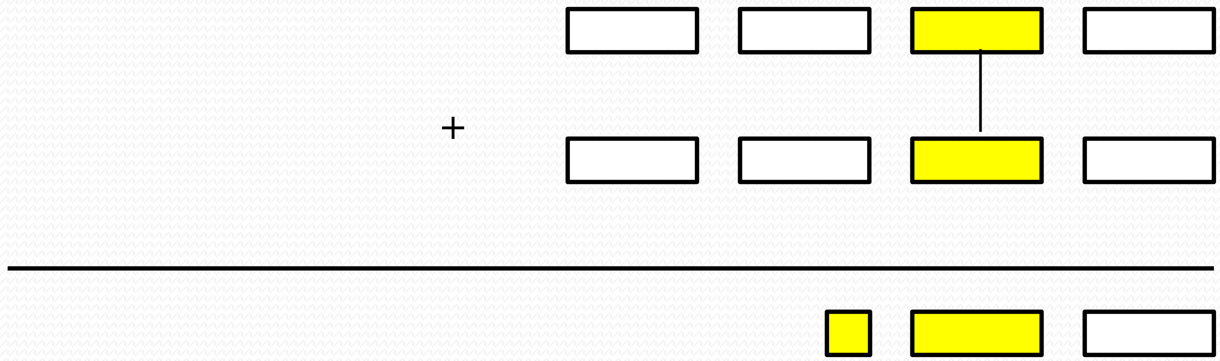
Large-Integer Addition



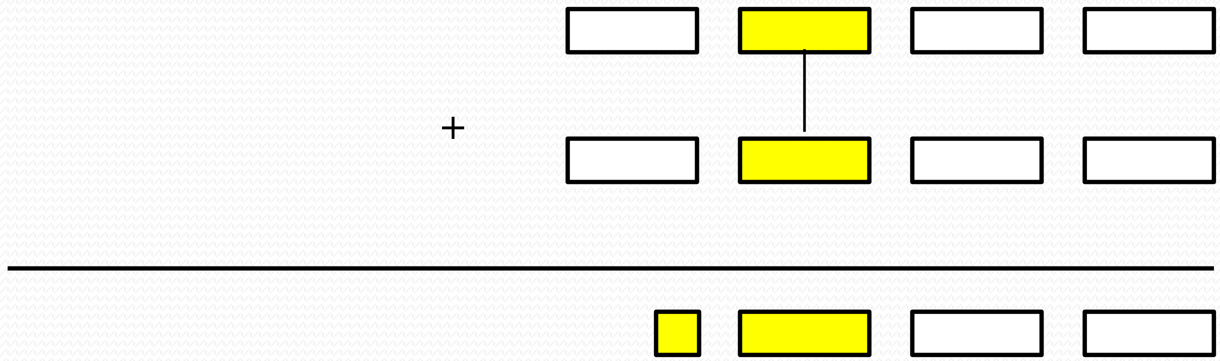
Large-Integer Addition



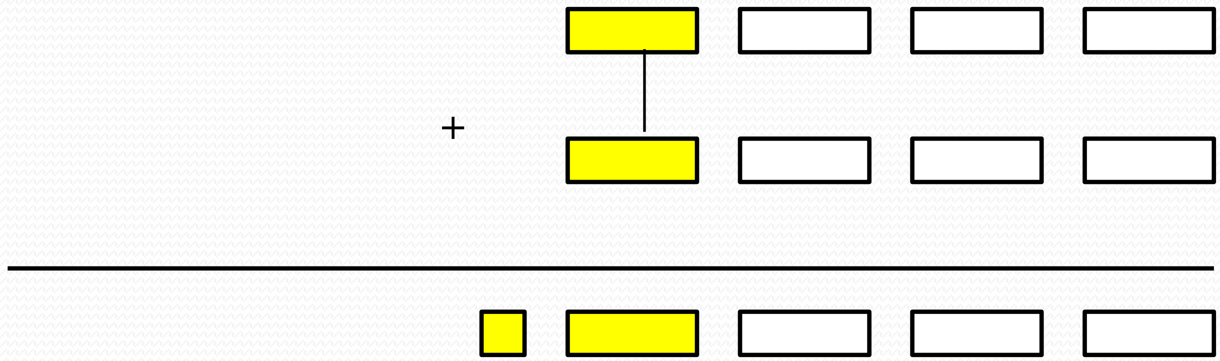
Large-Integer Addition



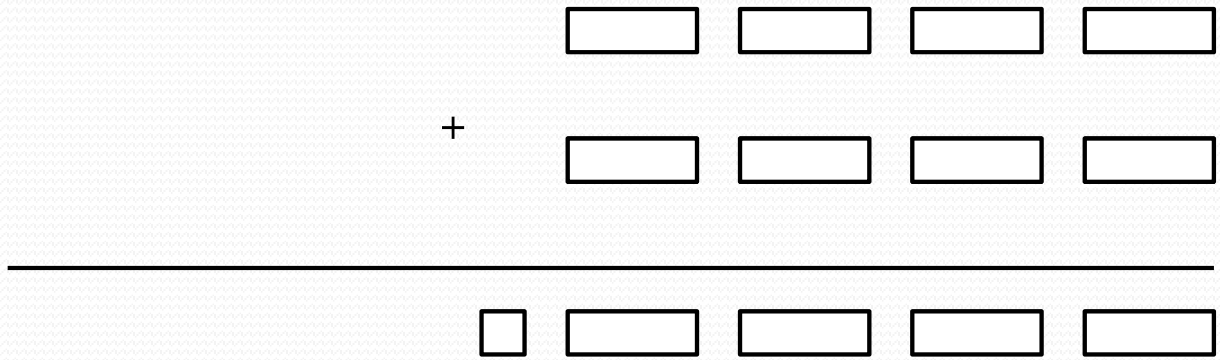
Large-Integer Addition



Large-Integer Addition



Large-Integer Addition

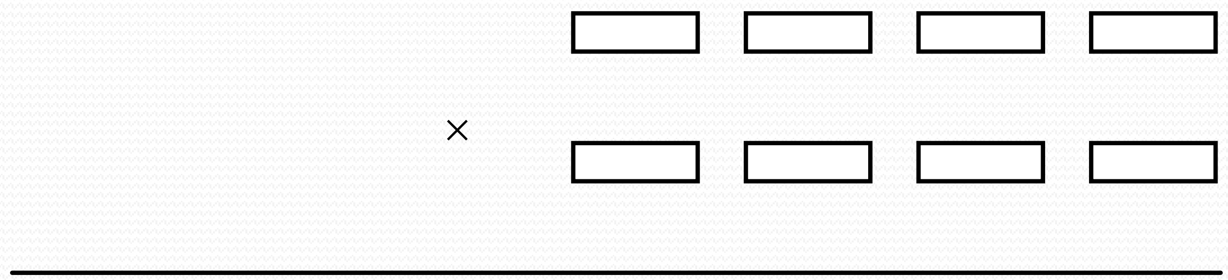


Large-Integer Addition

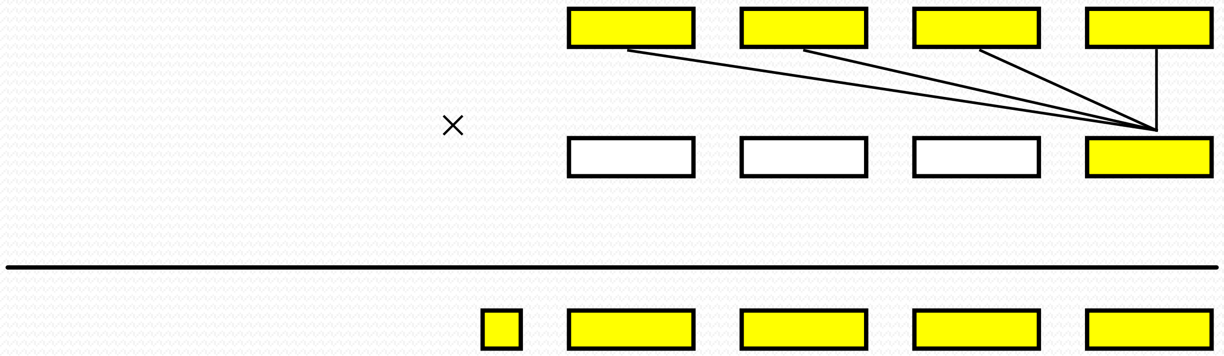
In general, adding two large integers – each consisting of n small blocks – requires $O(n)$ small-integer additions.

Large-integer subtraction is similar.

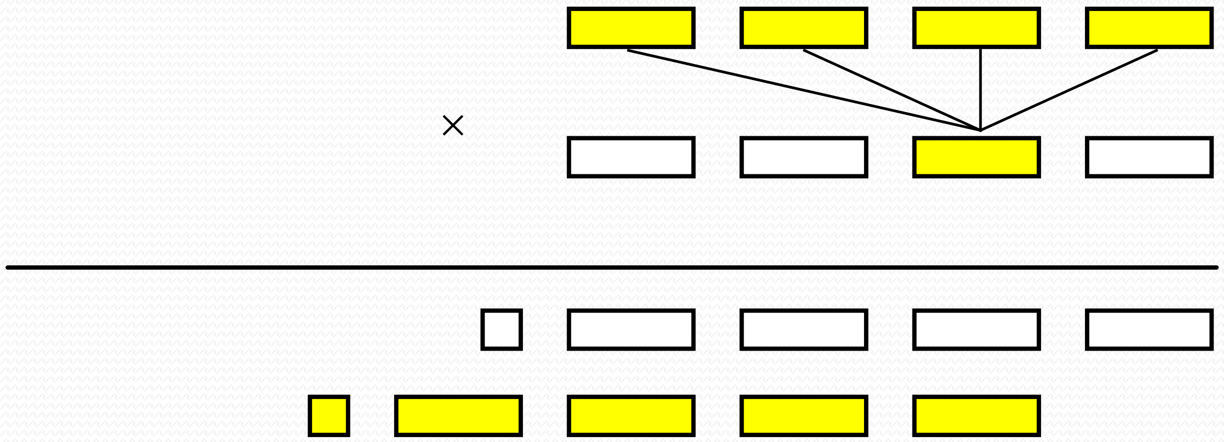
Large-Integer Multiplication



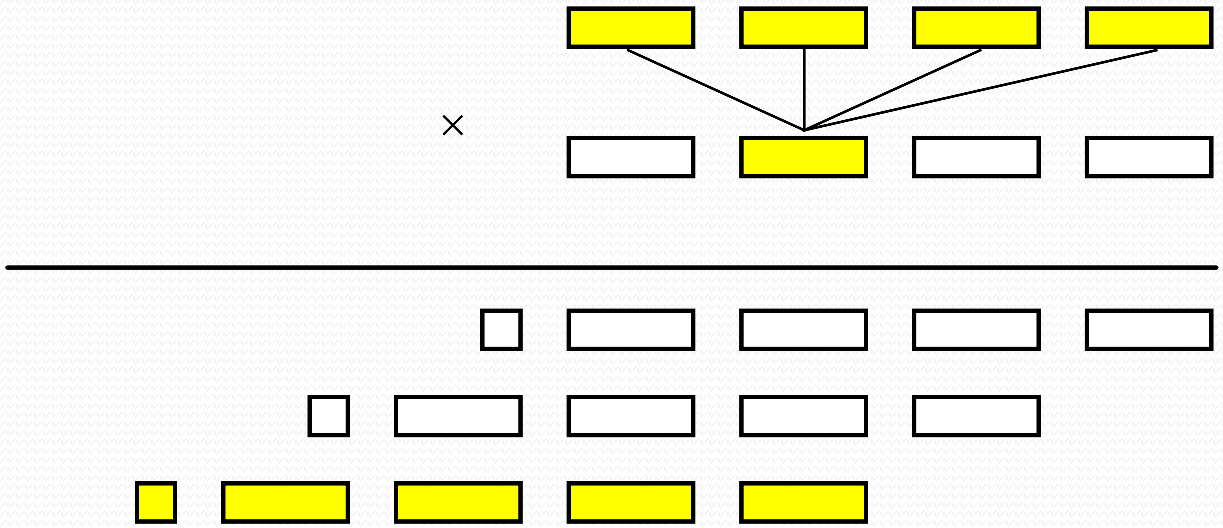
Large-Integer Multiplication



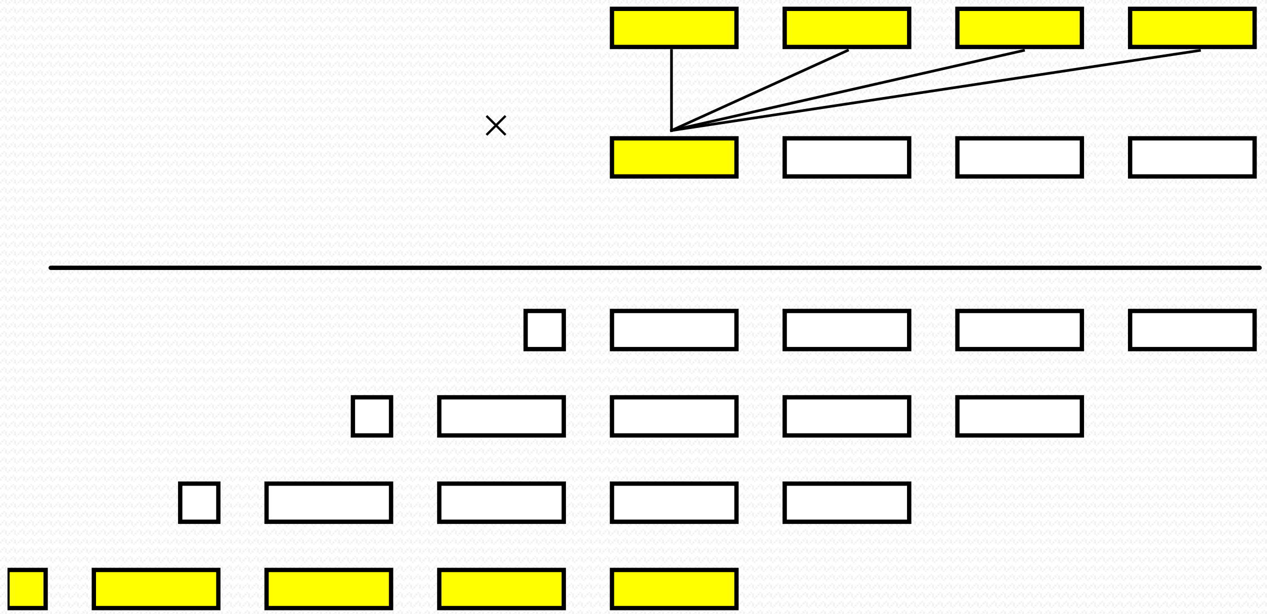
Large-Integer Multiplication



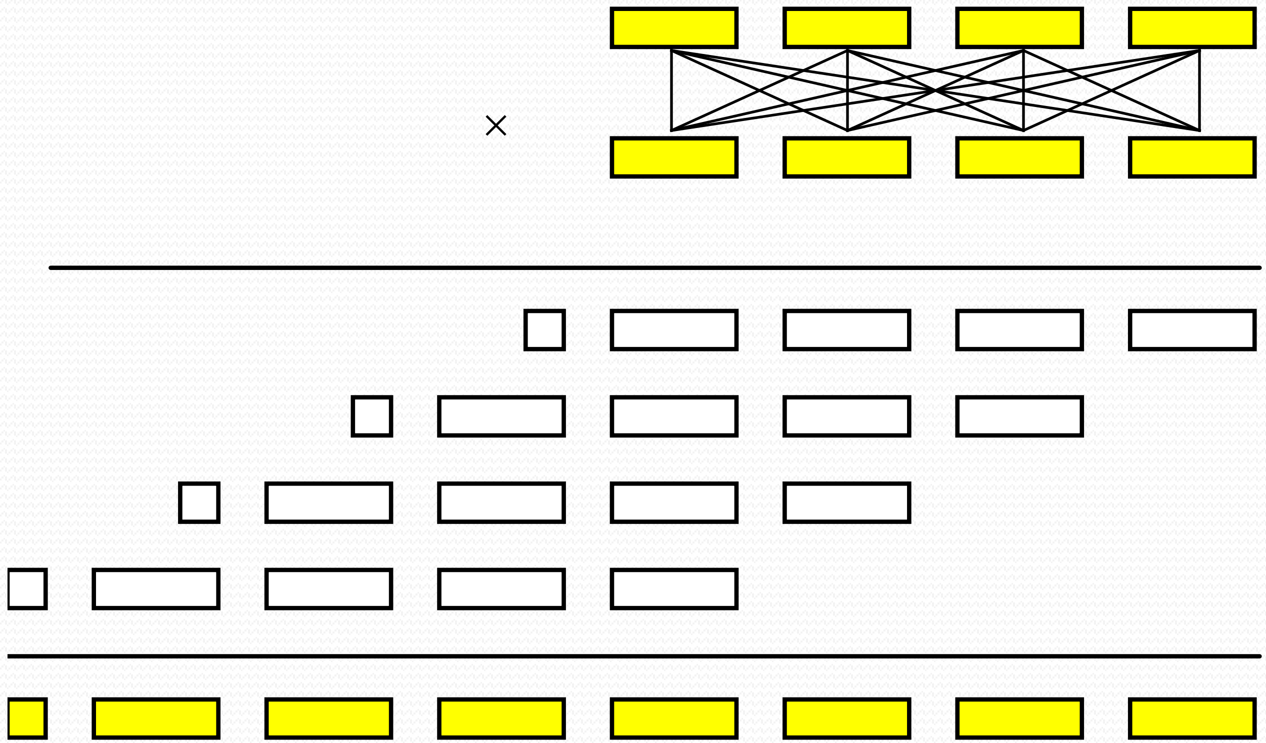
Large-Integer Multiplication



Large-Integer Multiplication



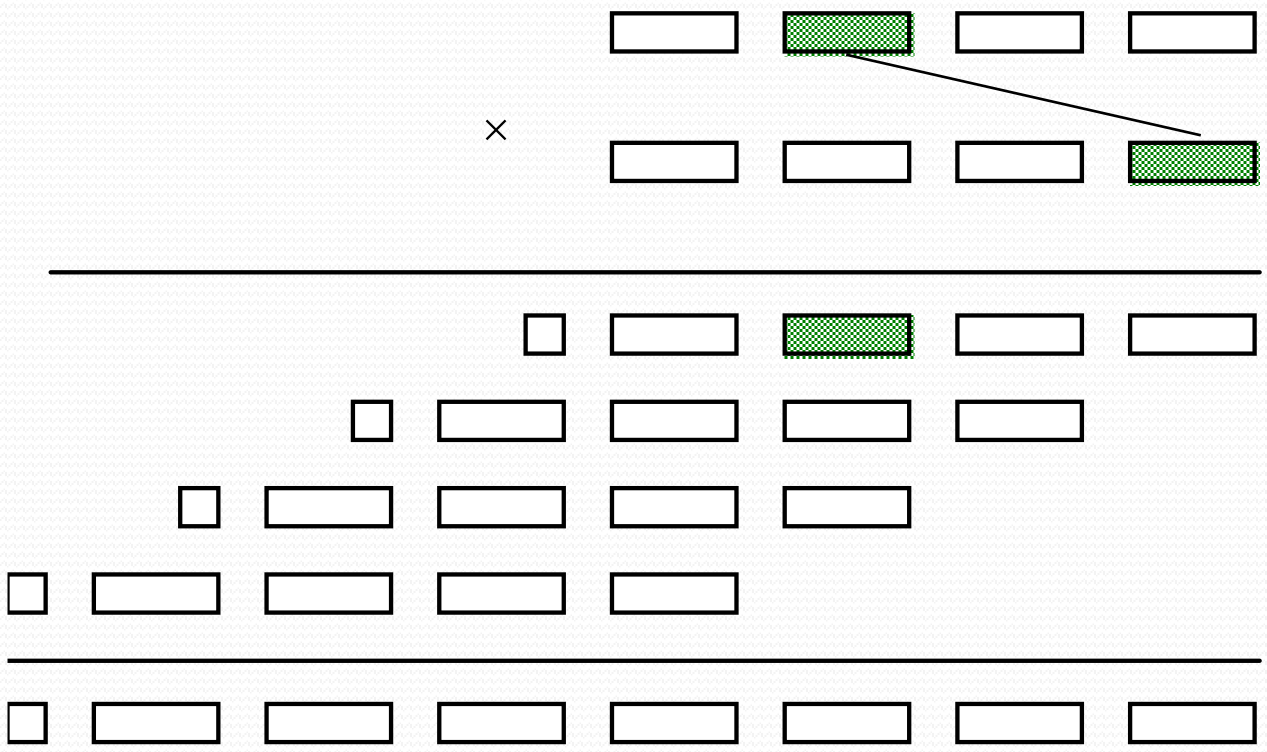
Large-Integer Multiplication



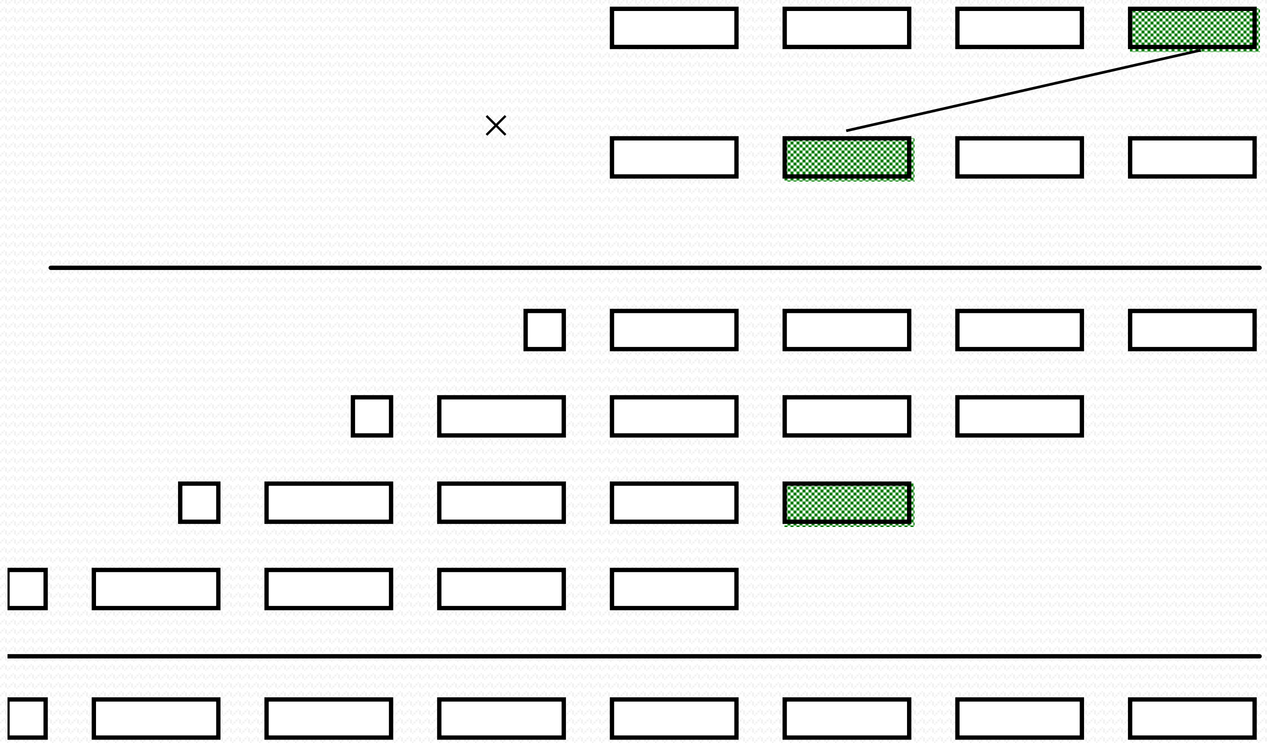
Large-Integer Multiplication

In general, multiplying two large integers – each consisting of n small blocks – requires $O(n^2)$ small-integer multiplications and $O(n)$ *large-integer* additions.

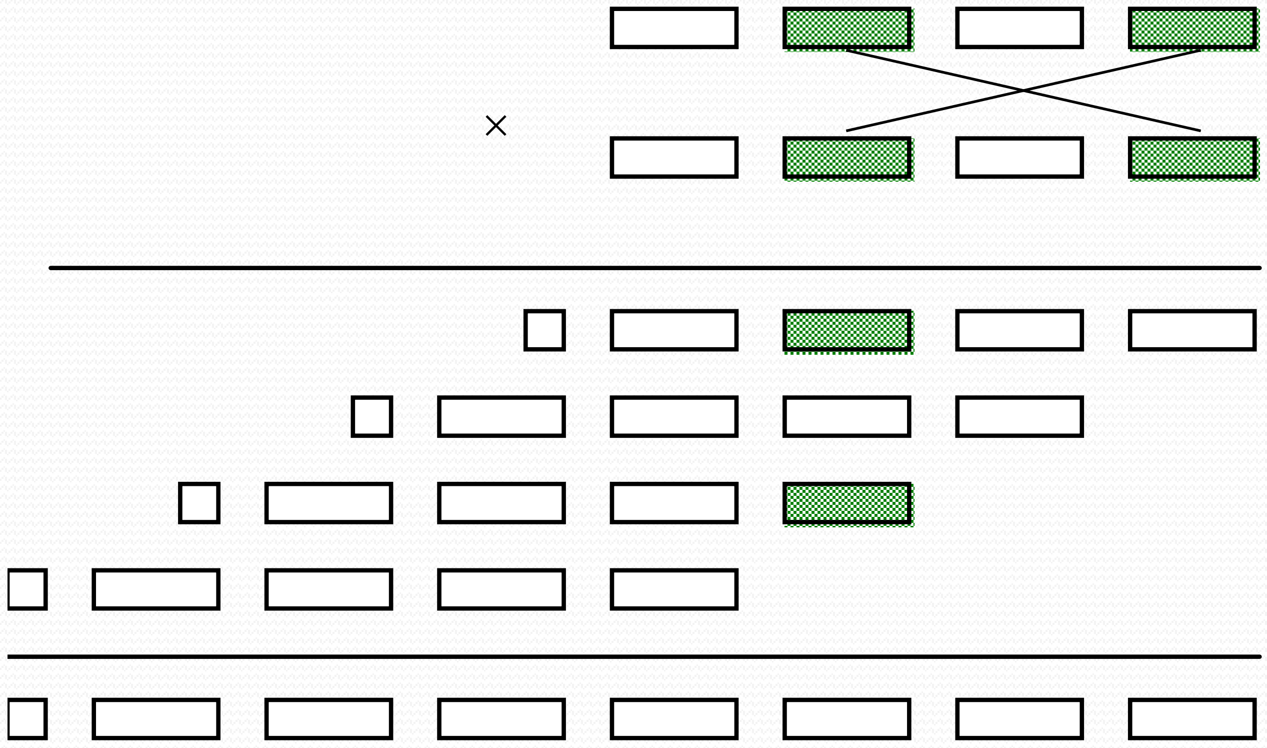
Large-Integer Squaring



Large-Integer Squaring



Large-Integer Squaring



Large-Integer Squaring

Careful bookkeeping can save nearly half of the small-integer multiplications (and nearly half of the time).

Recall computing Y^X mod N

- About 2/3 of the multiplications required to compute Y^X are actually squarings.

- Overall, efficient squaring can save about 1/3 of the small multiplications required for modular exponentiation.

Karatsuba Multiplication

$$(A x + B)(C x + D) = AC x^2 + (AD+BC) x + BD$$

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

Given 4 coefficients A, B, C, and D,

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

Given 4 coefficients A, B, C, and D, we need to compute 3 values:

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

Given 4 coefficients A, B, C, and D,

we need to compute 3 values:

AC, AD+BC, and BD.

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

Given 4 coefficients A, B, C, and D,

we need to compute 3 values:

AC, AD+BC, and BD.

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

Given 4 coefficients A, B, C, and D,
we need to compute 3 values:
AC, AD+BC, and BD.

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

Given 4 coefficients A, B, C, and D,

we need to compute 3 values:

AC, AD+BC, and BD.

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

$$(A+B)(C+D) = AC + AD + BC + BD$$

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

$$(A+B)(C+D) = AC + AD + BC + BD$$

$$(A+B)(C+D) - AC - BD = AD + BC$$

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

$$(A+B)(C+D) = AC + AD + BC + BD$$

$$(A+B)(C+D) - AC - BD = AD + BC$$

3 multiplications, 2 additions, 2 subtractions

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

$$(A+B)(C+D) = AC + AD + BC + BD$$

$$(A+B)(C+D) - AC - BD = AD + BC$$

3 multiplications, 2 additions, 2 subtractions

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

$$(A+B)(C+D) = AC + AD + BC + BD$$

$$(A+B)(C+D) - AC - BD = AD + BC$$

3 multiplications, 2 additions, 2 subtractions

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

$$(A+B)(C+D) = AC + AD + BC + BD$$

$$(A+B)(C+D) - AC - BD = AD + BC$$

3 multiplications, 2 additions, 2 subtractions

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

$$(A+B)(C+D) = AC + AD + BC + BD$$

$$(A+B)(C+D) - AC - BD = AD + BC$$

3 multiplications, 2 additions, 2 subtractions

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

$$(A+B)(C+D) = AC + AD + BC + BD$$

$$(A+B)(C+D) - AC - BD = AD + BC$$

3 multiplications, 2 additions, 2 subtractions

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

$$(A+B)(C+D) = AC + AD + BC + BD$$

$$(A+B)(C+D) - AC - BD = AD + BC$$

3 multiplications, 2 additions, 2 subtractions

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

$$(A+B)(C+D) = AC + AD + BC + BD$$

$$(A+B)(C+D) - AC - BD = AD + BC$$

3 multiplications, 2 additions, 2 subtractions

Karatsuba Multiplication

$$(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$$

4 multiplications, 1 addition

$$(A+B)(C+D) = AC + AD + BC + BD$$

$$(A+B)(C+D) - AC - BD = AD + BC$$

3 multiplications, 2 additions, 2 subtractions

Karatsuba Multiplication

- This can be done on integers as well as on polynomials, but it's not as nice on integers because of carries.
- The larger the integers, the larger the benefit.

Karatsuba Multiplication

$(A\times 2^k+B)(C\times 2^k+D) =$

$$AC\times 2^{2k} + (AD+BC)\times 2^k + BD$$

4 multiplications, 1 addition

$$(A+B)(C+D) = AC + AD + BC + BD$$

$$(A+B)(C+D) - AC - BD = AD + BC$$

3 multiplications, 2 additions, 2 subtractions

Chinese Remaindering

If $X = A \bmod P$, $X = B \bmod Q$, and $\gcd(P,Q) = 1$, then $X \bmod P \cdot Q$ can be computed as

$$X = A \cdot Q \cdot (Q^{-1} \bmod P) + B \cdot P \cdot (P^{-1} \bmod Q).$$

Chinese Remaindering

If $N = PQ$, then a computation mod N can be accomplished by performing the same computation mod P and again mod Q and then using Chinese Remaindering to derive the answer to the mod N computation.

Chinese Remaindering

Since modular exponentiation of n-bit integers requires $O(n^3)$ time, performing two modular exponentiations on half size values requires only about one quarter of the time of a single n-bit modular exponentiation.

Modular Reduction

Generally, computing $(A \times B) \bmod N$ requires much more than twice the time to compute $A \times B$.

Modular Reduction

Generally, computing $(A \times B) \bmod N$ requires much more than twice the time to compute $A \times B$.

Large-integer division is …

Modular Reduction

Generally, computing $(A \times B) \bmod N$ requires much more than twice the time to compute $A \times B$.

Large-integer division is …

slow …

Modular Reduction

Generally, computing $(A \times B) \bmod N$ requires much more than twice the time to compute $A \times B$.

Large-integer division is …

slow … cumbersome

Modular Reduction

Generally, computing $(A \times B) \bmod N$ requires much more than twice the time to compute $A \times B$.

Large-integer division is …

slow … cumbersome … disgusting

Modular Reduction

Generally, computing $(A \times B) \bmod N$ requires much more than twice the time to compute $A \times B$.

Large-integer division is …

slow … cumbersome … disgusting … wretched

The Montgomery Method

The Montgomery Method performs a domain transform to a domain in which the modular reduction operation can be achieved by multiplication and simple truncation.

Since a single modular exponentiation requires many modular multiplications and reductions, transforming the arguments is well justified.

Montgomery Multiplication

Let A, B, and M be n-block integers represented in base x with $0 \leq M < x^n$.

Let $R = x^n$. $\mathrm{GCD}(R,M) = 1$.

The *Montgomery Product* of A and B modulo M is the integer $ABR^{-1} \bmod M$.

Let $M' = -M^{-1} \bmod R$ and $S = ABM' \bmod R$.

Fact: $(AB+SM)/R \equiv ABR^{-1} \pmod{M}$.

Using the Montgomery Product

The Montgomery Product $ABR^{-1} \bmod M$ can be computed in the time required for two ordinary large-integer multiplications.

Montgomery transform: $A \rightarrow AR \bmod M$.

The Montgomery product of $(AR \bmod M)$ and $(BR \bmod M)$ is $(ABR \bmod M)$.

One-Way Functions

$$Z = Y^X \bmod N$$

One-Way Functions

Informally, $F : X \rightarrow Y$ is a *one-way* if

- Given x, $y = F(x)$ is easily computable.
- Given y, it is difficult to find *any* x for which $y = F(x)$.

One-Way Functions

The family of functions

$$F_{Y,N}(X) = Y^X \bmod N$$

is *believed* to be one-way for *most* N and Y.

One-Way Functions

The family of functions

$$F_{Y,N}(X) = Y^X \bmod N$$

is *believed* to be one-way for *most* N and Y.

No one has ever *proven* a function to be one-way, and doing so would, at a minimum, yield as a consequence that $P \neq NP$.

One-Way Functions

When viewed as a two-argument function, the (candidate) one-way function

$$F_N(Y,X) = Y^X \bmod N$$

also satisfies a useful additional property which has been termed *quasi-commutivity:*

$$F(F(Y,X_1),X_2) = F(F(Y,X_2),X_1)$$

since $Y^{X_1 X_2} = Y^{X_2 X_1}$.

Diffie-Hellman Key Exchange

Alice Bob

Diffie-Hellman Key Exchange

Alice

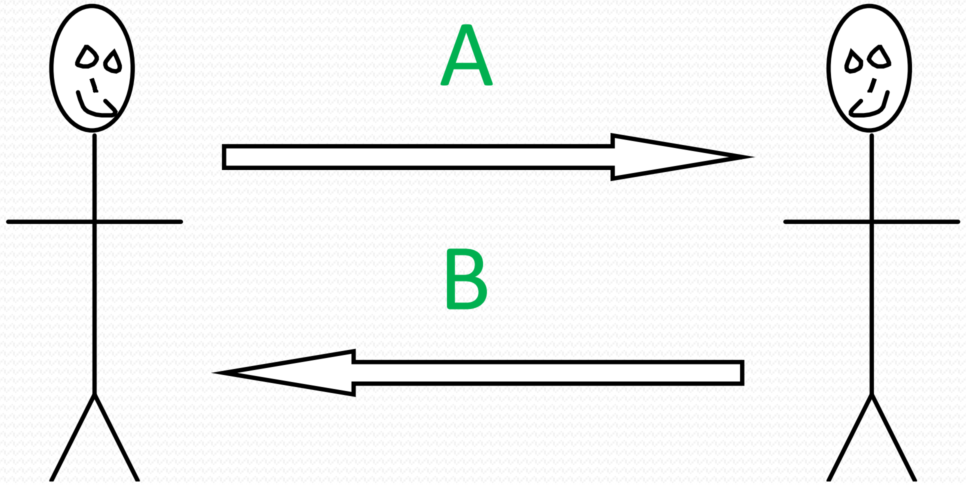
- Randomly select a large integer a and send $A = Y^a \bmod N$.

Bob

- Randomly select a large integer b and send $B = Y^b \bmod N$.

Diffie-Hellman Key Exchange

Alice Bob



Diffie-Hellman Key Exchange

Alice

- Randomly select a large integer a and send $A = Y^a \bmod N$.

Bob

- Randomly select a large integer b and send $B = Y^b \bmod N$.

Diffie-Hellman Key Exchange

Alice

- Randomly select a large integer a and send $A = Y^a \bmod N$.
- Compute the key $K = B^a \bmod N$.

Bob

- Randomly select a large integer b and send $B = Y^b \bmod N$.
- Compute the key $K = A^b \bmod N$.

Diffie-Hellman Key Exchange

Alice

- Randomly select a large integer a and send $A = Y^a \bmod N$.
- Compute the key $K = B^a \bmod N$.

Bob

- Randomly select a large integer b and send $B = Y^b \bmod N$.
- Compute the key $K = A^b \bmod N$.

$$B^a = Y^{ba} = Y^{ab} = A^b$$

Diffie-Hellman Key Exchange

Diffie-Hellman Key Exchange

What does Eve see?

Diffie-Hellman Key Exchange

What does Eve see?

Y, Y^a, Y^b

Diffie-Hellman Key Exchange

What does Eve see?

$$Y, Y^{a}, Y^{b}$$

… but the exchanged key is Y^{ab}.

Diffie-Hellman Key Exchange

What does Eve see?

$$Y, Y^{a}, Y^{b}$$

… but the exchanged key is Y^{ab}.

Belief: Given Y, Y^{a}, Y^{b} it is difficult to compute Y^{ab}.

Diffie-Hellman Key Exchange

What does Eve see?

$$Y,\ Y^{a},\ Y^{b}$$

… but the exchanged key is Y^{ab}.

Belief: Given Y, Y^{a}, Y^{b} it is difficult to compute Y^{ab}.

Contrast with discrete logarithm assumption: Given Y, Y^{a} it is difficult to compute a.

More on *Quasi-Commutivity*

Quasi-commutivity has additional applications.

- decentralized digital signatures
- membership testing
- digital time-stamping

One-Way Trap-Door Functions

$$Z = Y^X \bmod N$$

One-Way Trap-Door Functions

$$Z=Y^X \bmod N$$

Recall that this equation is solvable for Y if the factorization of N is known, but is *believed* to be hard otherwise.

RSA Public-Key Cryptosystem

Alice | Anyone

RSA Public-Key Cryptosystem

Alice | **Anyone**

- Select two large random primes P & Q.

RSA Public-Key Cryptosystem

Alice | Anyone

- Select two large random primes P & Q.
- Publish the product N=PQ.

RSA Public-Key Cryptosystem

Alice

- Select two large random primes P & Q.
- Publish the product $N=PQ$.

Anyone

- To send message Y to Alice, compute

  $Z=Y^X \bmod N$.

RSA Public-Key Cryptosystem

Alice

- Select two large random primes P & Q.
- Publish the product $N=PQ$.

Anyone

- To send message Y to Alice, compute
  $Z=Y^X \bmod N$.
- Send Z and X to Alice.

RSA Public-Key Cryptosystem

Alice

- Select two large random primes P & Q.
- Publish the product N=PQ.

- Use knowledge of P & Q to compute Y.

Anyone

- To send message Y to Alice, compute

  $Z=Y^X \bmod N$.
- Send Z and X to Alice.

RSA Public-Key Cryptosystem

In practice, the exponent X is almost always fixed to be $X = 65537 = 2^{16} + 1$.

Some RSA Details

When $N=PQ$ is the product of distinct primes,

$$Y^X \bmod N = Y$$

whenever

$X \bmod (P-1)(Q-1) = 1$ and $0 \le Y < N$.

Some RSA Details

When $N=PQ$ is the product of distinct primes,

$$Y^X \bmod N = Y$$

whenever

$X \bmod (P-1)(Q-1) = 1$ and $0 \leq Y < N$.

Alice can easily select integers E and D such that $E \times D \bmod (P-1)(Q-1) = 1$.

Some RSA Details

Encryption: $E(Y) = Y^E \bmod N$.

Decryption: $D(Y) = Y^D \bmod N$.

$$
\begin{aligned}
D(E(Y)) & \\
&= (Y^E \bmod N)^D \bmod N \\
&= Y^{ED} \bmod N \\
&= Y
\end{aligned}
$$

RSA Signatures

RSA Signatures

An additional property

RSA Signatures

An additional property

$D(E(Y)) = Y^{ED} \bmod N = Y$

RSA Signatures

An additional property

$D(E(Y)) = Y^{ED} \bmod N = Y$

$E(D(Y)) = Y^{DE} \bmod N = Y$

RSA Signatures

An additional property

$D(E(Y)) = Y^{ED} \bmod N = Y$

$E(D(Y)) = Y^{DE} \bmod N = Y$

Only Alice (knowing the factorization of N) knows D. Hence only Alice can compute $D(Y) = Y^{D} \bmod N$.

RSA Signatures

An additional property

$D(E(Y)) = Y^{ED} \bmod N = Y$

$E(D(Y)) = Y^{DE} \bmod N = Y$

Only Alice (knowing the factorization of N) knows D. Hence only Alice can compute $D(Y) = Y^D \bmod N$.

This $D(Y)$ serves as Alice's signature on Y.

Public Key Directory

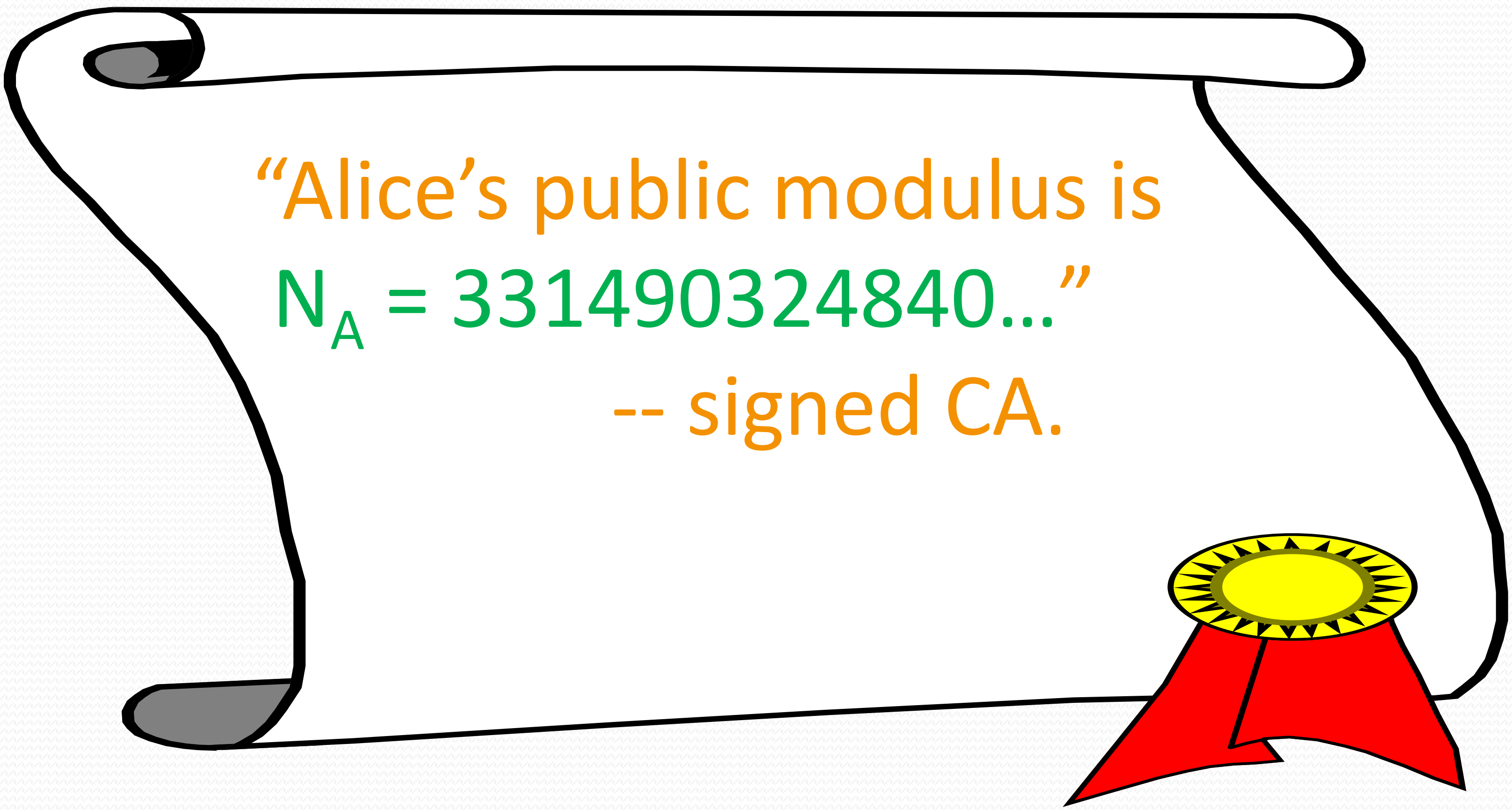
Name	Public Key	Encryption
Alice	N_A	$E_A(Y)=Y^E \bmod N_A$
Bob	N_B	$E_B(Y)=Y^E \bmod N_B$
Carol	N_C	$E_C(Y)=Y^E \bmod N_C$
⋮	⋮	⋮

Public Key Directory

Name	Public Key	Encryption
Alice	N_A	$E_A(Y)=Y^E \bmod N_A$
Bob	N_B	$E_B(Y)=Y^E \bmod N_B$
Carol	N_C	$E_C(Y)=Y^E \bmod N_C$
⋮	⋮	⋮

(Recall that E is commonly fixed to be E=65537.)

Certificate Authority



Trust Chains

Alice certifies Bob’s key.

Bob certifies Carol’s key.

If I trust Alice should I accept Carol’s key?

Authentication

Authentication

How can I use RSA to *authenticate* someone's identity?

Authentication

How can I use RSA to *authenticate* someone's identity?

If Alice's public key E_A, just pick a random message m and send $E_A(m)$.

Authentication

How can I use RSA to *authenticate* someone's identity?

If Alice's public key E_A, just pick a random message m and send $E_A(m)$.

If m comes back, I must be talking to Alice.

Authentication

Should Alice be happy with this method of authentication?

Authentication

Should Alice be happy with this method of authentication?

Bob sends Alice the authentication string
y = "I owe Bob $1,000,000 - signed Alice."

Authentication

Should Alice be happy with this method of authentication?

Bob sends Alice the authentication string
y = "I owe Bob $1,000,000 - signed Alice."

Alice dutifully authenticates herself by decrypting (putting her signature on) *y*.

Authentication

What if Alice only returns authentication queries when the decryption has a certain format?

RSA Cautions

Is it reasonable to sign/decrypt something given to you by someone else?

Note that RSA is multiplicative. Can this property be used/abused?

RSA Cautions

$$D(Y_1) \times D(Y_2) = D(Y_1 \times Y_2)$$

Thus, if I've decrypted (or signed) Y_1 and Y_2, I've also decrypted (or signed) $Y_1 \times Y_2$.

The Hastad Attack

Given

$$E_1(x) = x^3 \bmod n_1$$

$$E_2(x) = x^3 \bmod n_2$$

$$E_3(x) = x^3 \bmod n_3$$

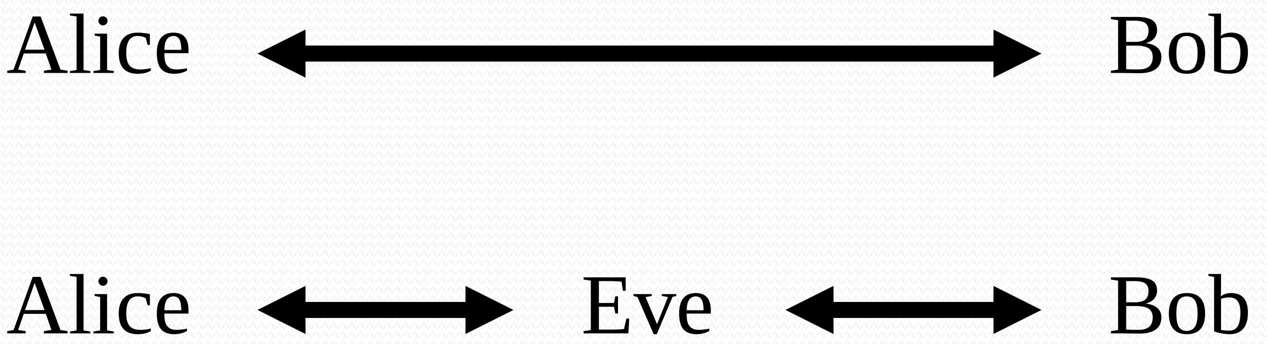
one can easily compute x.

The Bleichenbacher Attack

PKCS#1 Message Format:

$$00\ 01\ \underbrace{XX\ XX\ \ldots\ XX}_{\text{random non-zero bytes}}\ 00\ \underbrace{YY\ YY\ \ldots\ YY}_{\text{message}}$$

“Man-in-the-Middle” Attacks



The Practical Side

The Practical Side

- RSA can be used to encrypt any data.

The Practical Side

- RSA can be used to encrypt any data.
- Public-key (asymmetric) cryptography is very inefficient when compared to traditional private-key (symmetric) cryptography.

The Practical Side

The Practical Side

For efficiency, one generally uses RSA (or another public-key algorithm) to transmit a private (symmetric) key.

The Practical Side

For efficiency, one generally uses RSA (or another public-key algorithm) to transmit a private (symmetric) key.

The private *session* key is used to encrypt any subsequent data.

The Practical Side

For efficiency, one generally uses RSA (or another public-key algorithm) to transmit a private (symmetric) key.

The private *session* key is used to encrypt any subsequent data.

Digital signatures are only used to sign a *digest* of the message.