

Practical Aspects of Modern Cryptography

Josh Benaloh
Brian LaMacchia
John Manferdelli



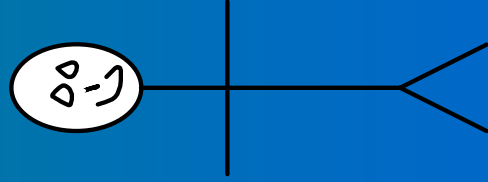
Cryptography is ...

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

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

Characters

Alice



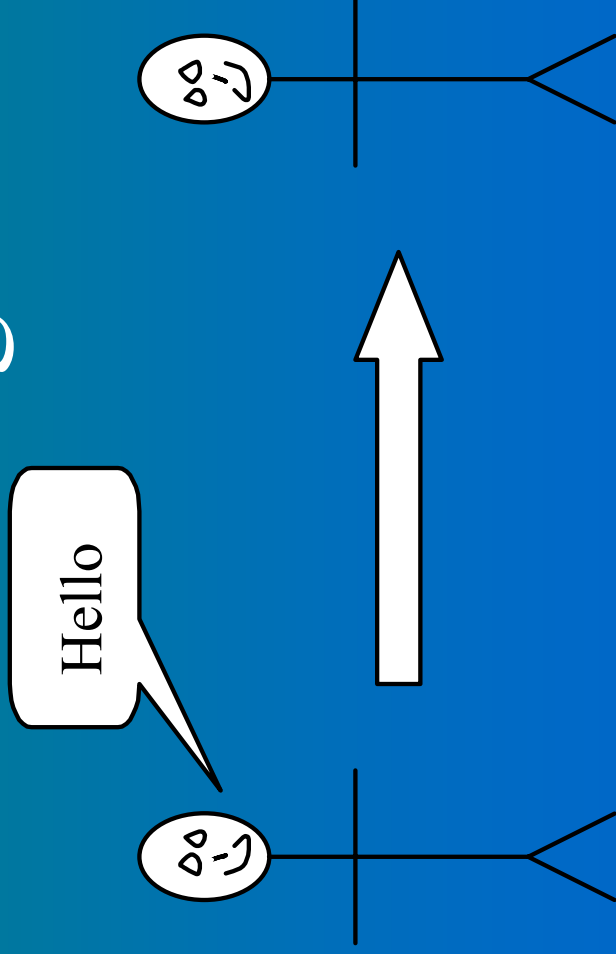
Characters

Bob



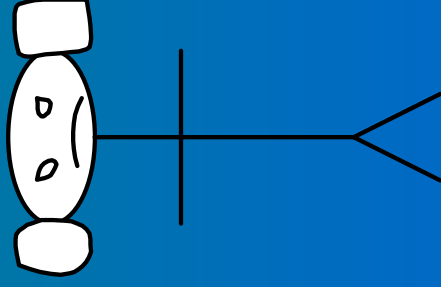
Basic Communication

Alice talking to Bob



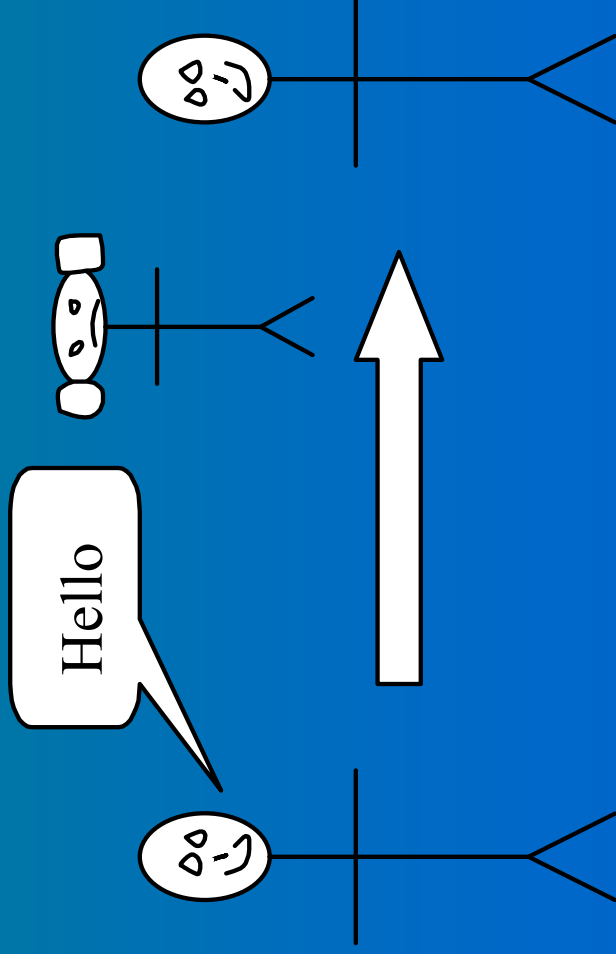
Another Character

Eve



Basic Communication Problem

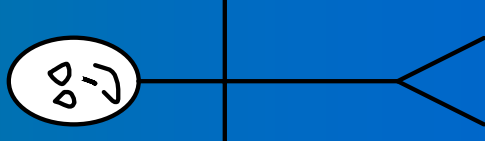
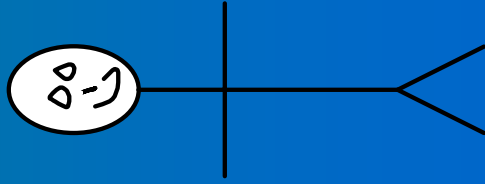
Eve listening to
Alice talking to Bob



Two-Party Environments

Alice

Bob



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.

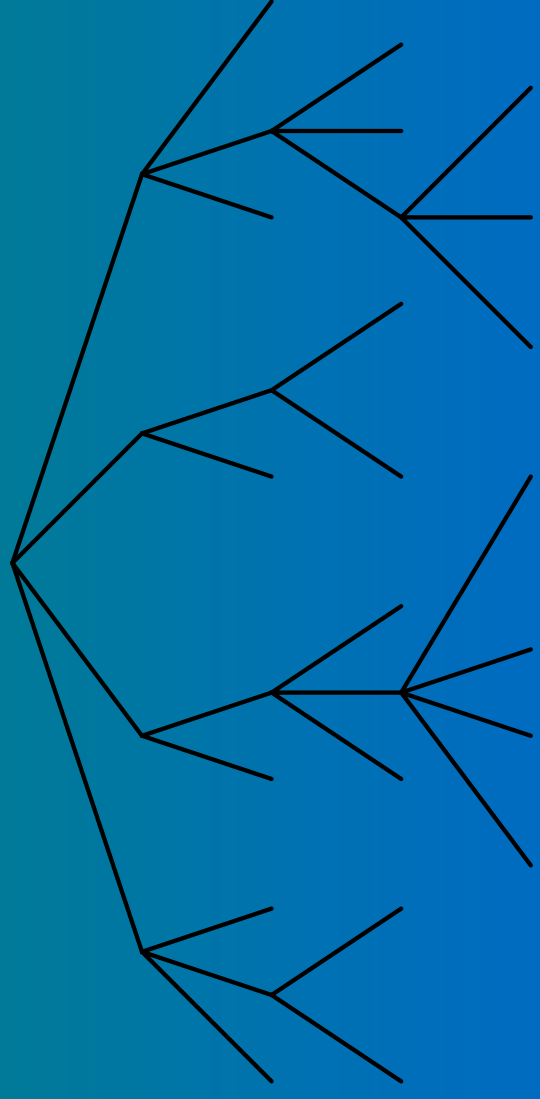
Is Remote Coin Flipping Possible?

Two-part answer:

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

A Protocol Flow Tree

A:
B:
A:
B:



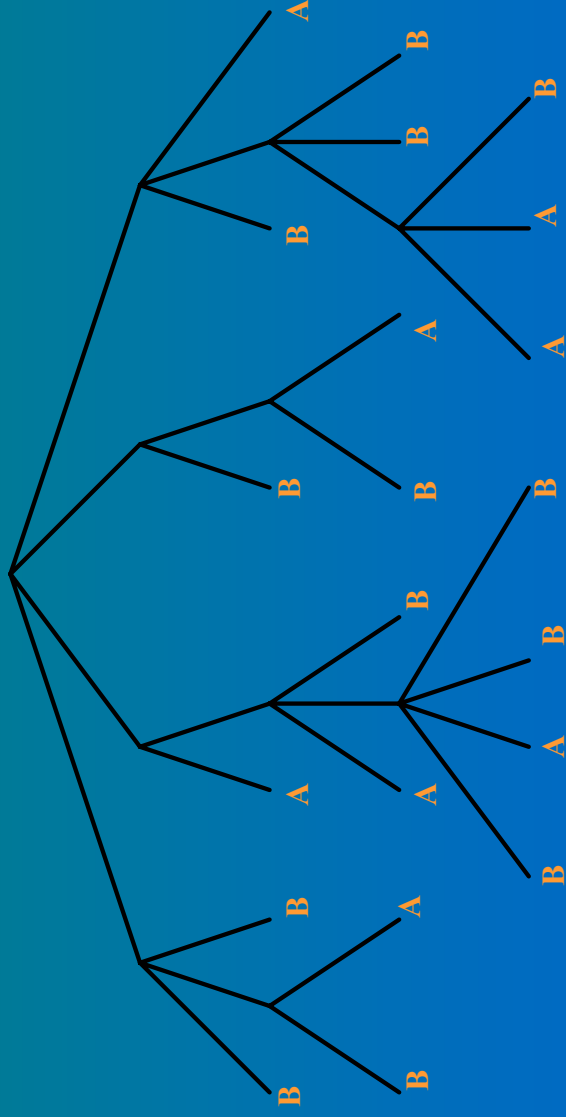
A Protocol Flow Tree

A:

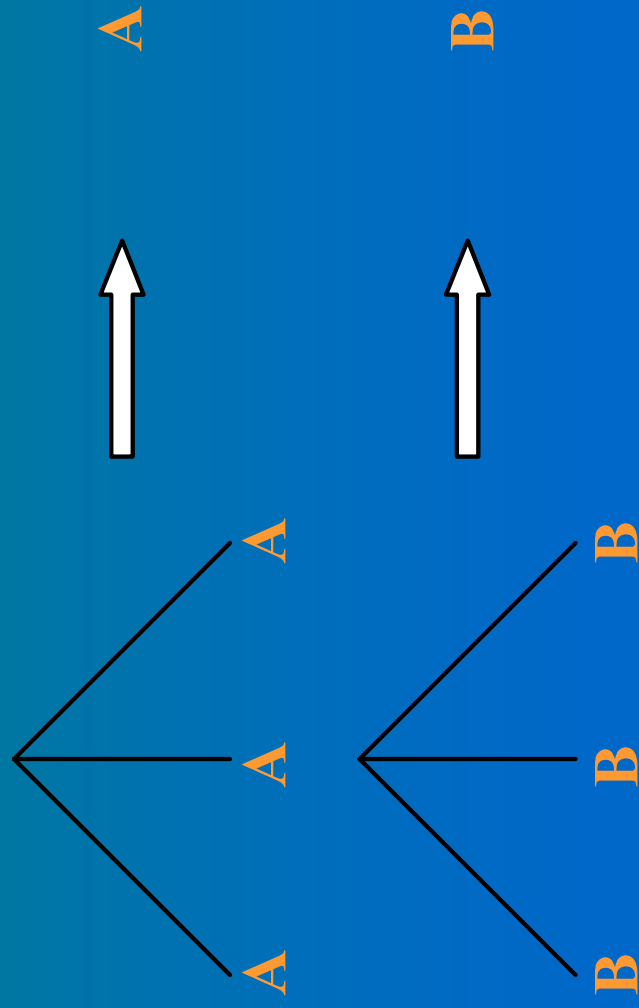
B:

A:

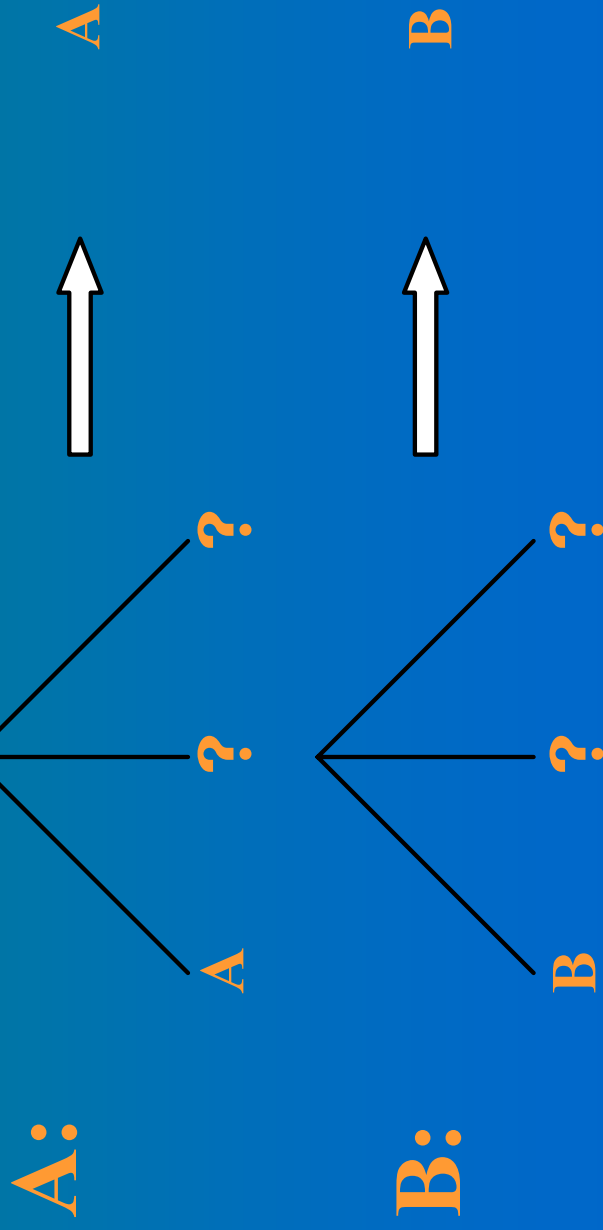
B:



Pruning the Tree



Pruning the Tree



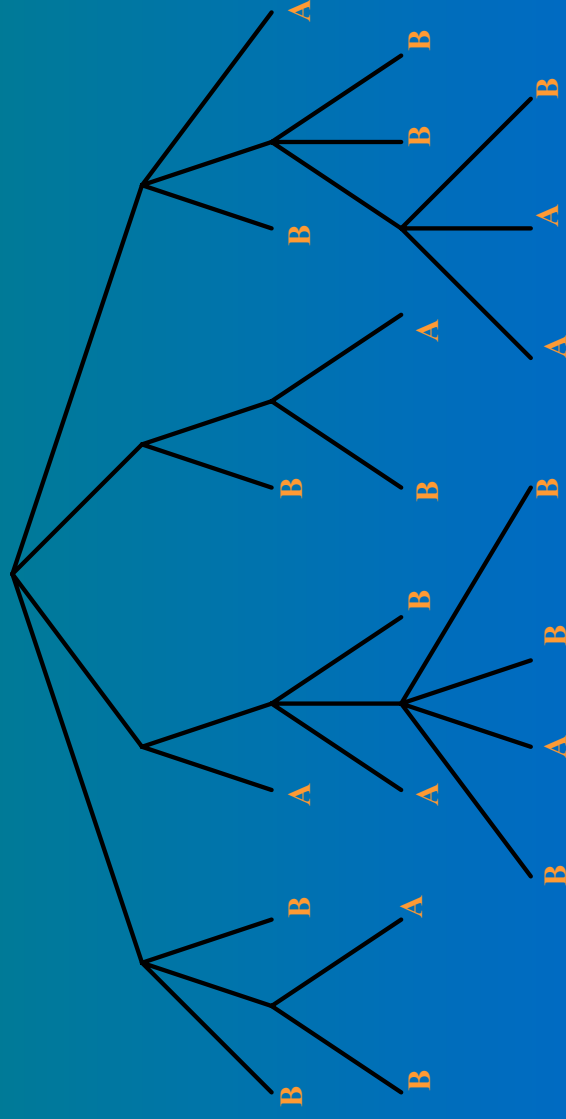
A Protocol Flow Tree

A:

B:

A:

B:



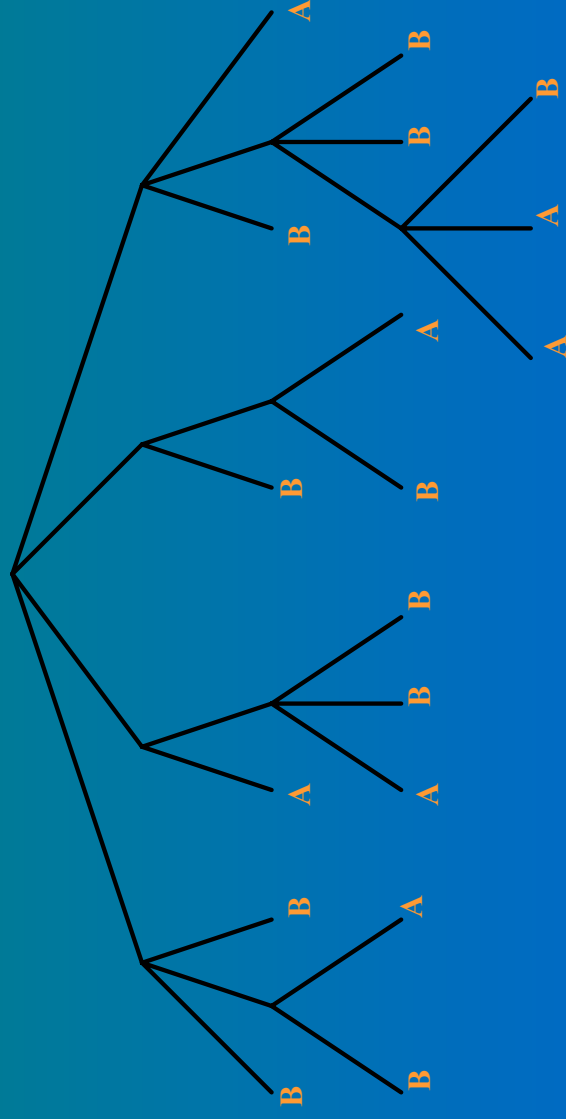
A Protocol Flow Tree

A:

B:

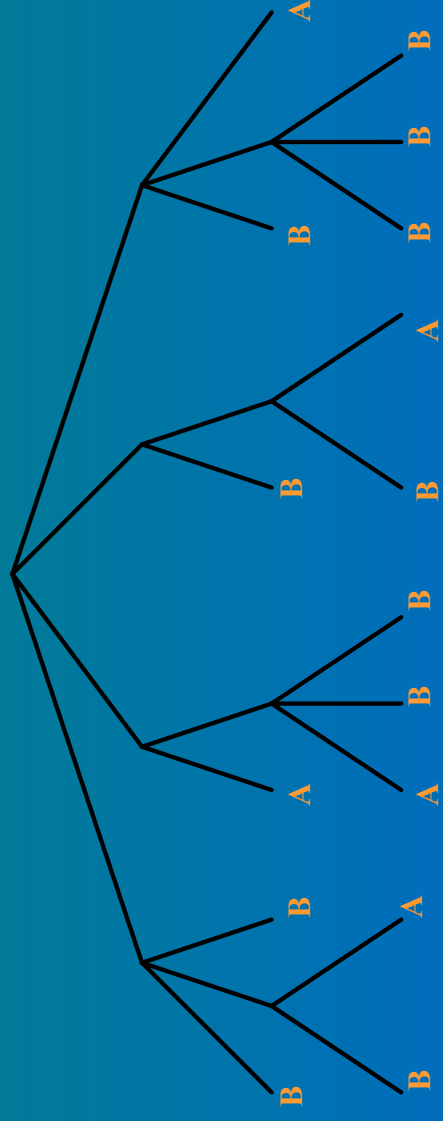
A:

B:



A Protocol Flow Tree

A:
B:
A:
B:



A Protocol Flow Tree

A:

B:

A:

B:



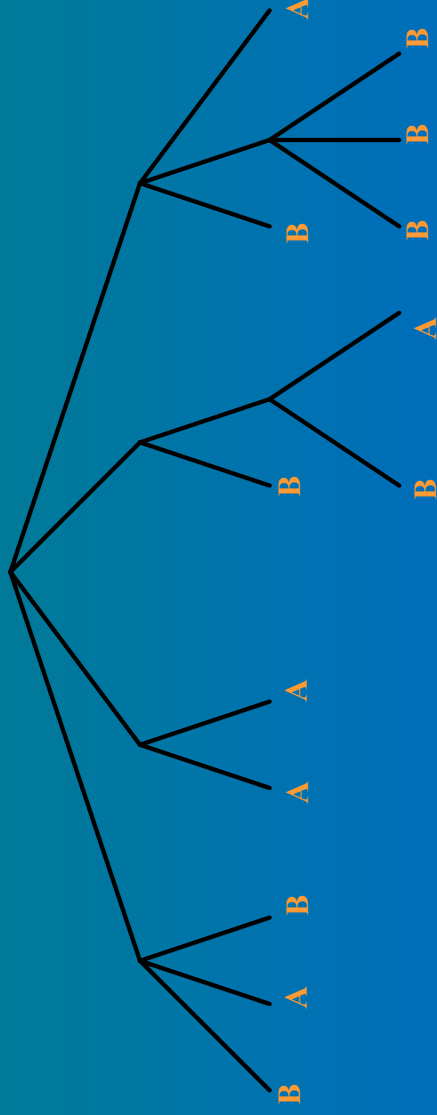
A Protocol Flow Tree

A:

B:

A:

B:



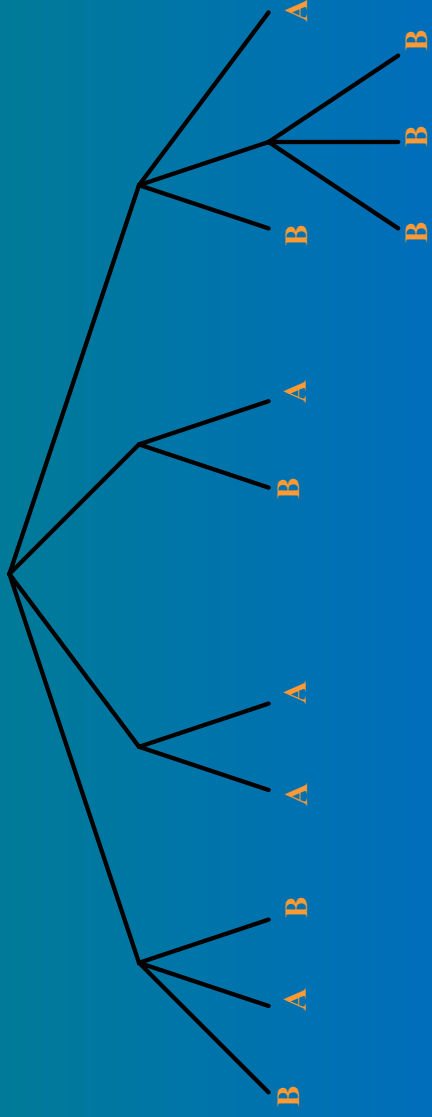
A Protocol Flow Tree

A:

B:

A:

B:



A Protocol Flow Tree

A:

B:

A:

B:



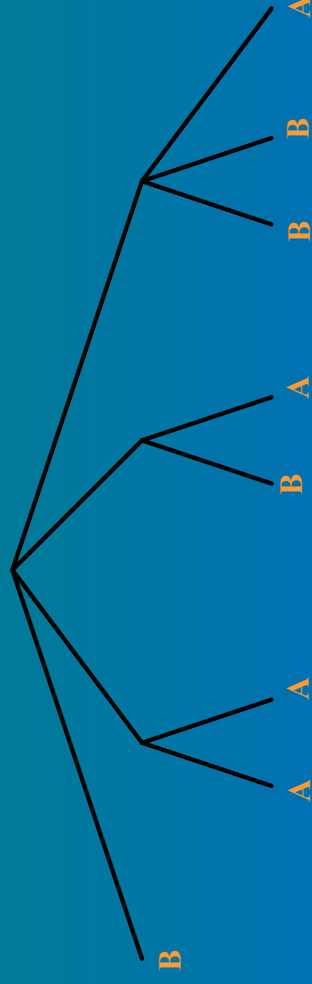
A Protocol Flow Tree

A:

B:

A:

B:



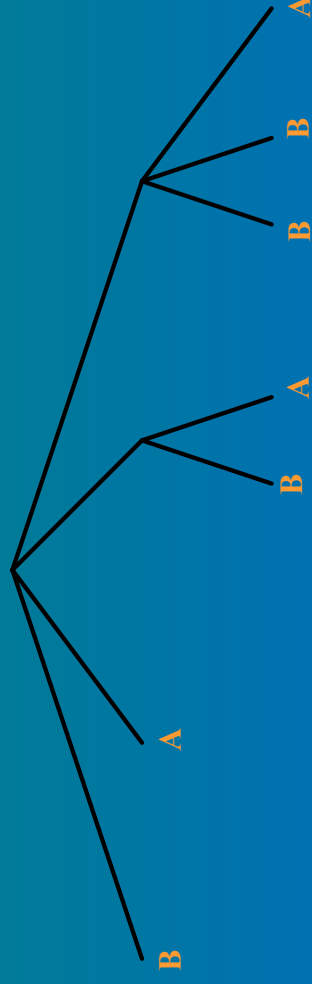
A Protocol Flow Tree

A:

B:

A:

B:



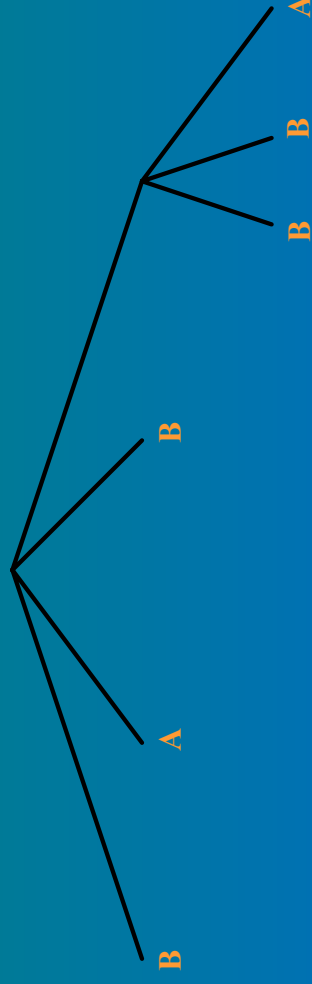
A Protocol Flow Tree

A:

B:

A:

B:



A Protocol Flow Tree

A:

B:

A:

B:



A Protocol Flow Tree

A:

B:

A:

B:

A

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

The INTEGERS

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

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

Even

How to Remotely Flip a Coin

The INTEGERS

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

Fact 2

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

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

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 $N = PQ$.

Bob

How to Remotely Flip a Coin

Alice

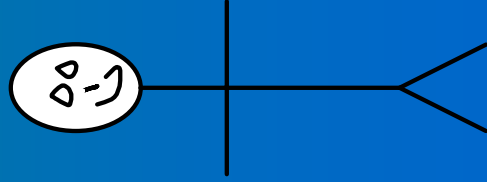
- Randomly select a bit $b \in \{\pm 1\}$ and two *large* integers P and Q – both of type b .
- Compute $N = PQ$.
- Send N to Bob.

Bob

How to Remotely Flip a Coin

Alice

Bob



N



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

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.

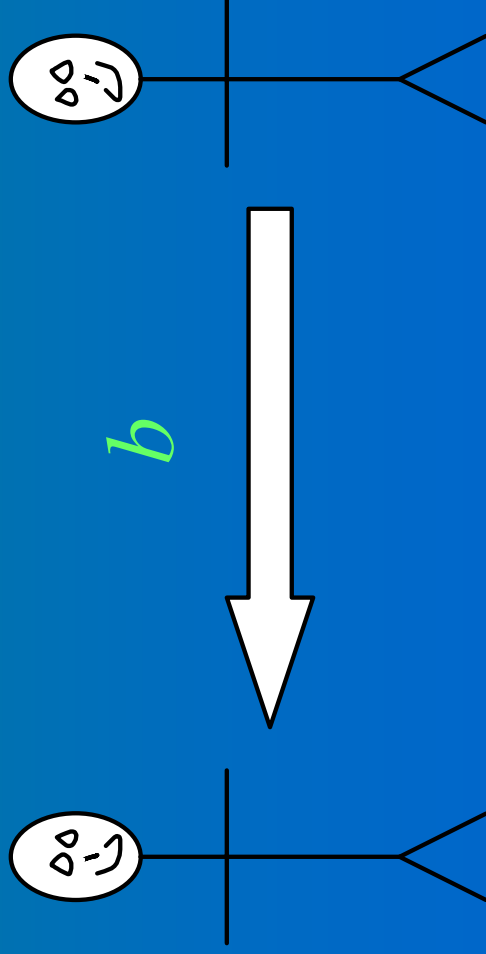
Bob

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

How to Remotely Flip a Coin

Alice

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

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

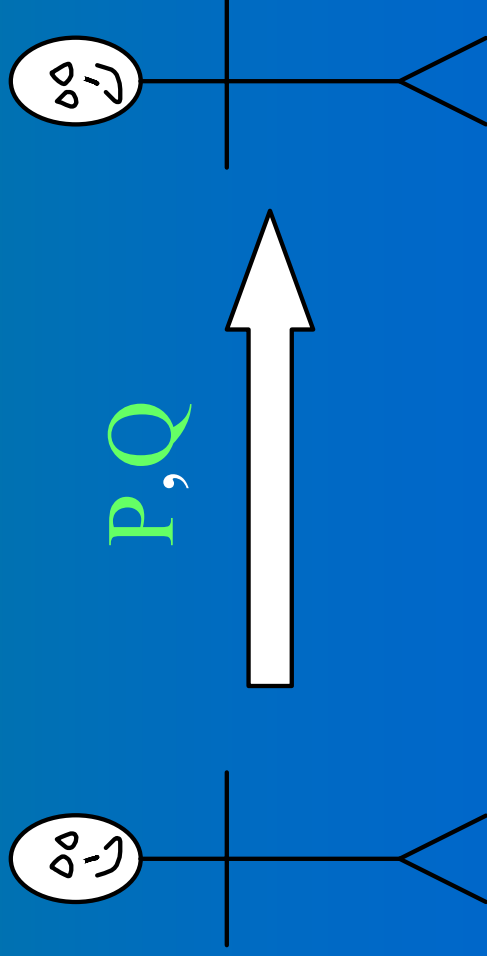
- After receiving N from Alice, guess the value of b and send this guess to Alice.

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

How to Remotely Flip a Coin

Alice

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.

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

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

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: $Z = X \cdot Y$

One-Way Functions

Two basic classes of one-way functions

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

One-Way Functions

Two basic classes of one-way functions

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

The Fundamental Equation

$$Z \equiv Y^X \pmod{N}$$

The Fundamental Equation

$$Z \equiv Y^X \pmod{N}$$

When Z is unknown, it can be efficiently computed.

The Fundamental Equation

$$Z \equiv Y^X \pmod{N}$$

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

The Fundamental Equation

$$Z \equiv Y^X \pmod{N}$$

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

The Fundamental Equation

$$Z \equiv Y^X \pmod{N}$$

... *unless* the factorization of N is known.

The Fundamental Equation

$$Z \equiv Y^X \pmod{N}$$

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

Implementation

$$Z \equiv Y^X \pmod{N}$$

How to compute $Y^X \bmod N$

How to compute $Y^X \bmod N$

Compute Y^X and then reduce mod N .

How to compute $Y^X \bmod N$

Compute Y^X and then reduce mod N .

- If X , Y , and N each are 1,000-bit integers, Y^X consists of $\sim 2^{1010}$ bits.

How to compute $Y^X \bmod N$

Compute Y^X and then reduce mod N .

- If X , Y , and N each are 1,000-bit integers, Y^X consists of $\sim 2^{1010}$ 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 $Y^X \bmod N$

- 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 $\sim 2^{900}$ 32-bit multiplications per second to complete the computation before the sun burns out.

How to compute $Y^X \bmod N$

How to compute $Y^X \bmod N$

Multiplication by Repeated Doubling

How to compute $Y^X \bmod N$

Multiplication by Repeated Doubling

To compute $X \cdot Y$,

How to compute $Y^X \bmod N$

Multiplication by Repeated Doubling

To compute $X \cdot Y$,

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

How to compute $Y^X \bmod N$

Multiplication by Repeated Doubling

To compute $X \cdot Y$,
compute $Y, 2Y, 4Y, 8Y, \dots$
and sum up those values dictated by the binary
representation of X .

How to compute $Y^X \bmod N$

Multiplication by Repeated Doubling

To compute $X \cdot Y$,

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

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

Example: $26Y = 2Y + 8Y + 16Y$.

How to compute $Y^X \bmod N$

How to compute $Y^X \bmod N$

Exponentiation by Repeated Squaring

How to compute $Y^X \bmod N$

Exponentiation by Repeated Squaring

To compute Y^X ,

How to compute $Y^X \bmod N$

Exponentiation by Repeated Squaring

To compute Y^X ,

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

How to compute $Y^X \bmod N$

Exponentiation by Repeated Squaring

To compute Y^X ,
compute $Y, Y^2, Y^4, Y^8, Y^{16}, \dots$
and multiply those values dictated by the binary
representation of X .

How to compute $Y^X \bmod N$

Exponentiation by Repeated Squaring

To compute Y^X ,
compute $Y, Y^2, Y^4, Y^8, Y^{16}, \dots$
and multiply those values dictated by the binary
representation of X .

Example: $Y^{26} = Y^2 \cdot Y^8 \cdot Y^{16}$.

How to compute $Y^X \bmod N$

We can now perform a 1,000-bit modular exponentiation using $\sim 1,500$ 1,000-bit modular multiplications.

- 1,000 squarings: $y, y^2, y^4, \dots, y^{2^{1000}}$
- ~ 500 “ordinary” multiplications

Sliding Window Method

One way to speed up modular exponentiation is by precomputation of many small products.

For instance, if you have $y, y^2, y^3, \dots, y^{15}$ computed in advance, you can multiply by (for example) y^{13} without having to multiply individually by y, y^4 , and y^8 .

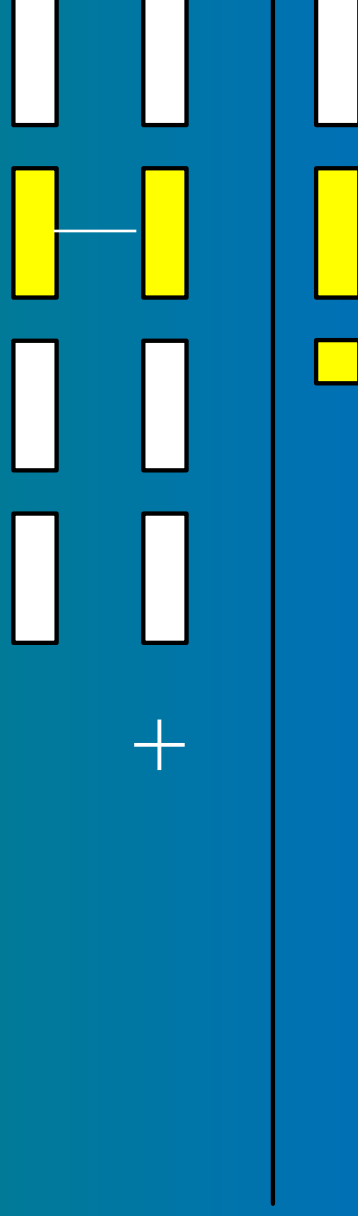
Large-Integer Operations

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

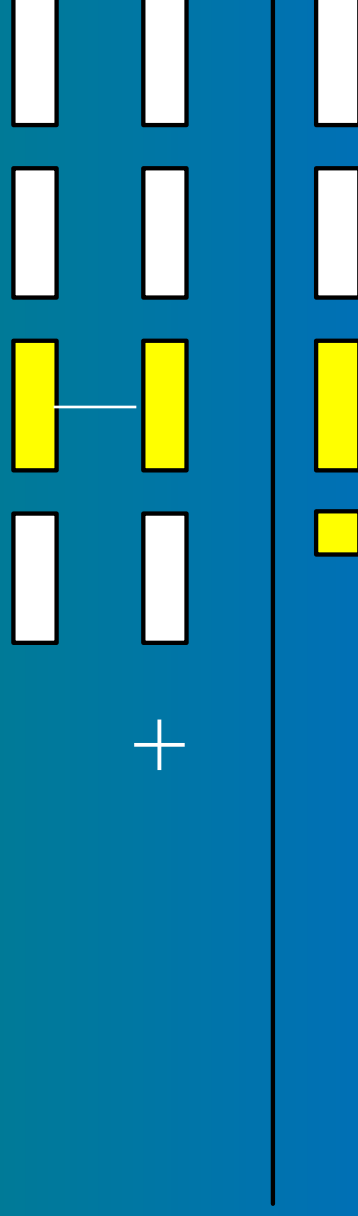
Large-Integer Addition



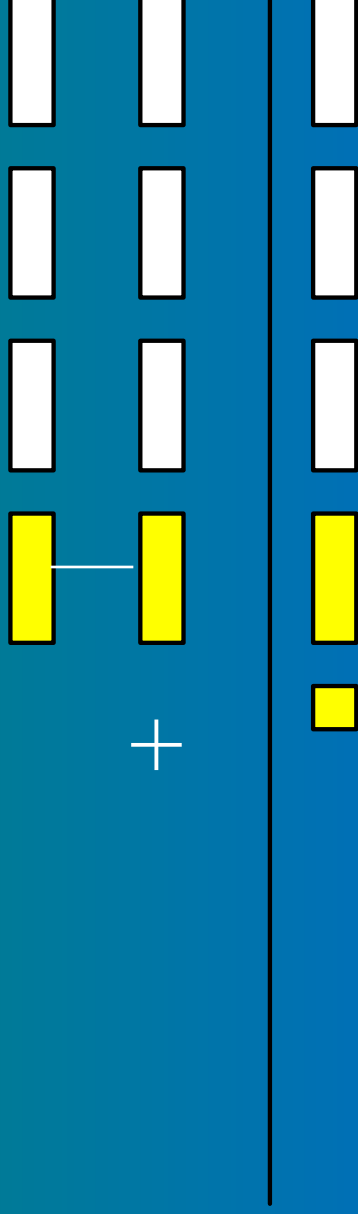
Large-Integer Addition



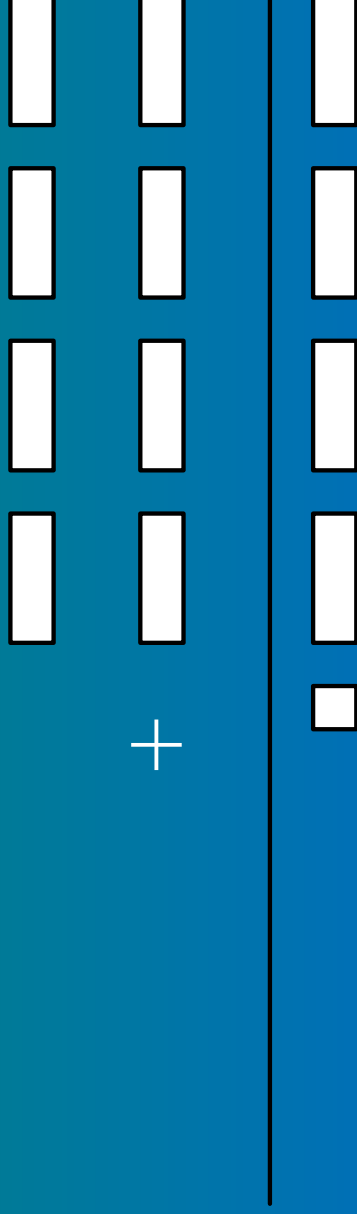
Large-Integer Addition



Large-Integer Addition



Large-Integer Addition



Large-Integer Addition

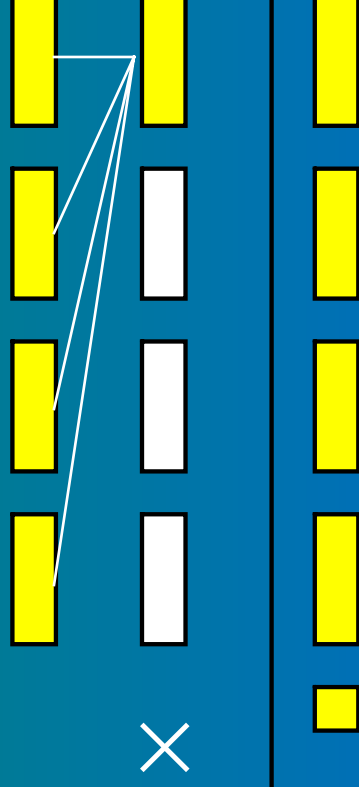
In general, adding two large integers – each consisting of n small blocks – requires $O(n)$ small-integer additions.

Large-integer subtraction is similar.

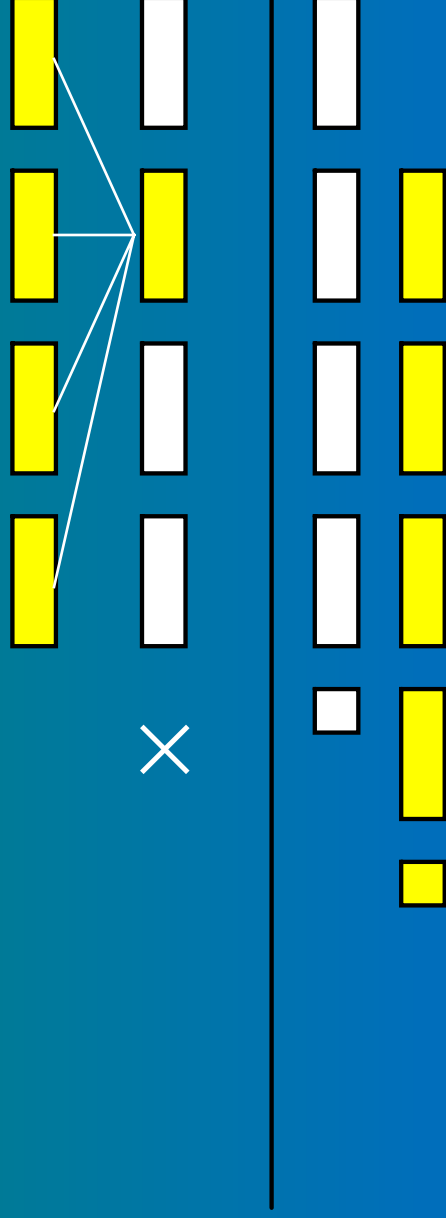
Large-Integer Multiplication

$$\begin{array}{r} \square \square \square \square \\ \times \square \square \square \square \\ \hline \end{array}$$

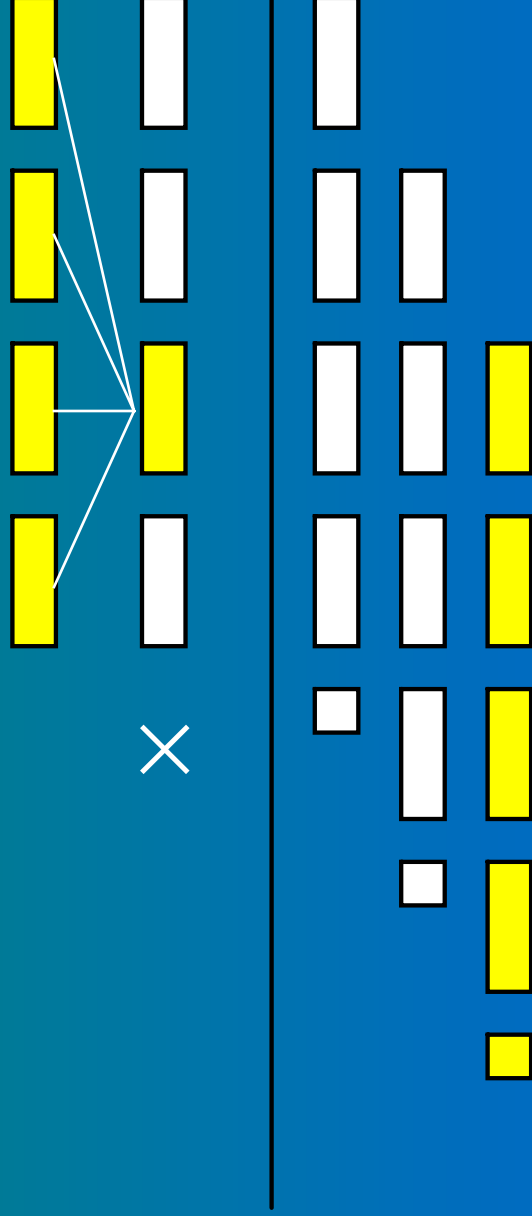
Large-Integer Multiplication



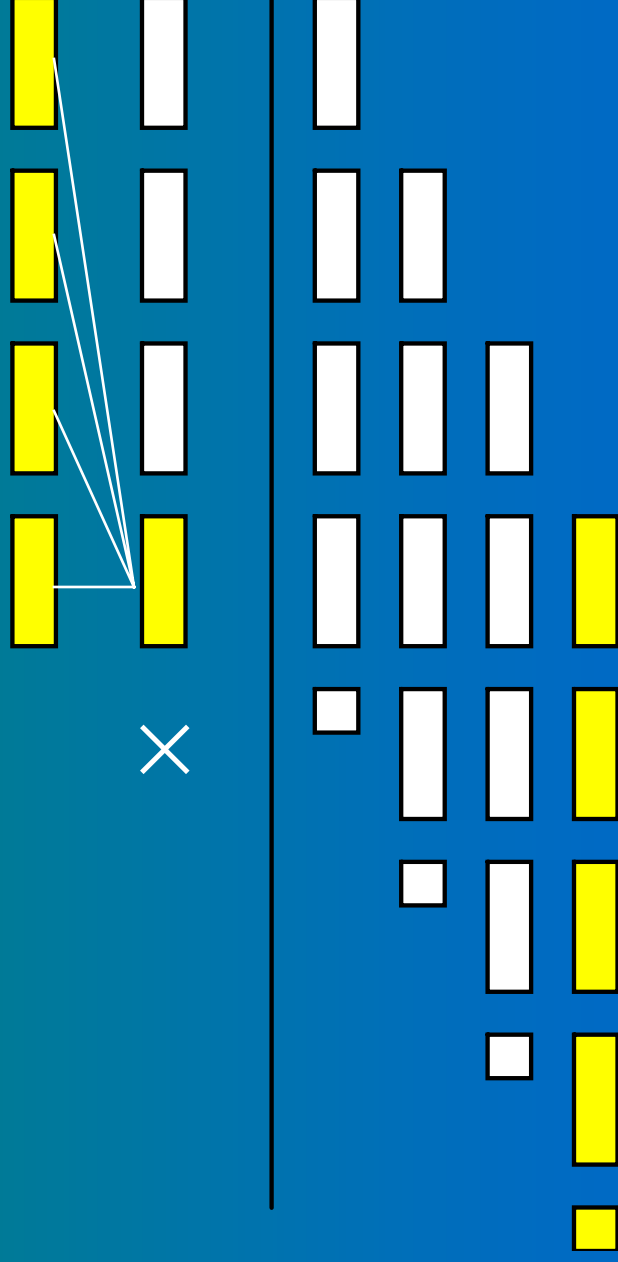
Large-Integer Multiplication



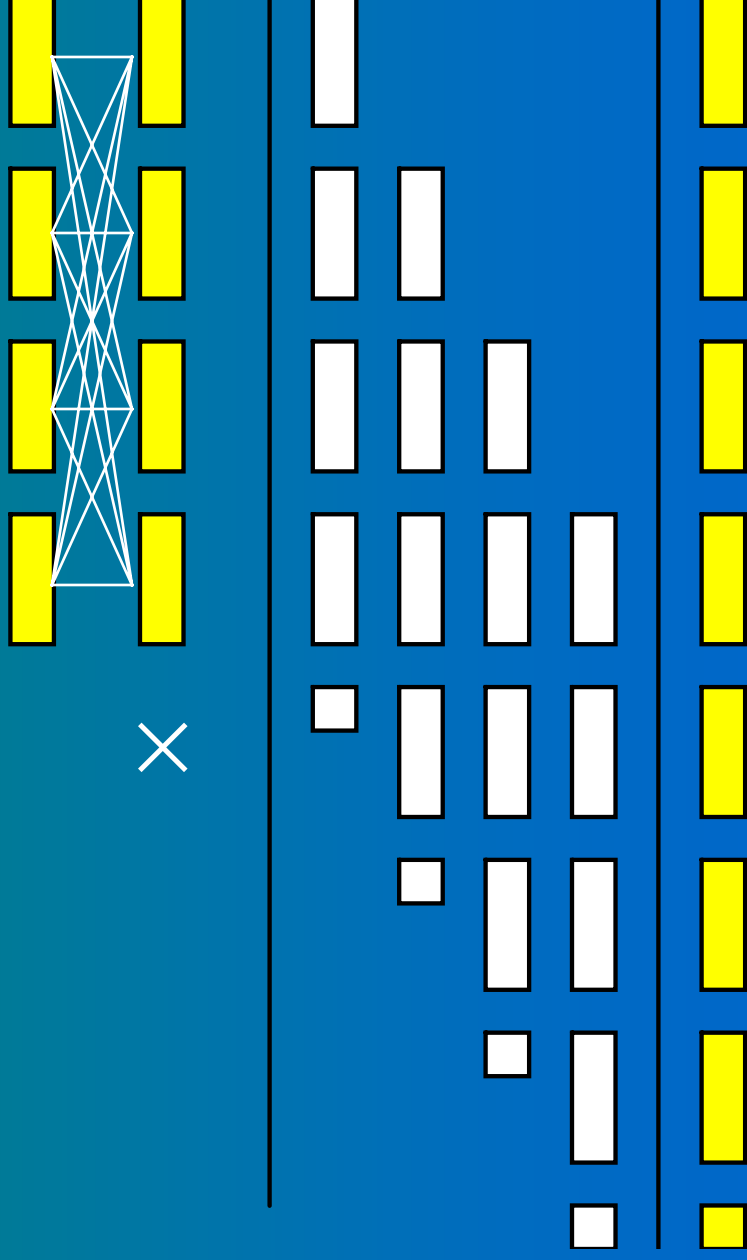
Large-Integer Multiplication



Large-Integer Multiplication



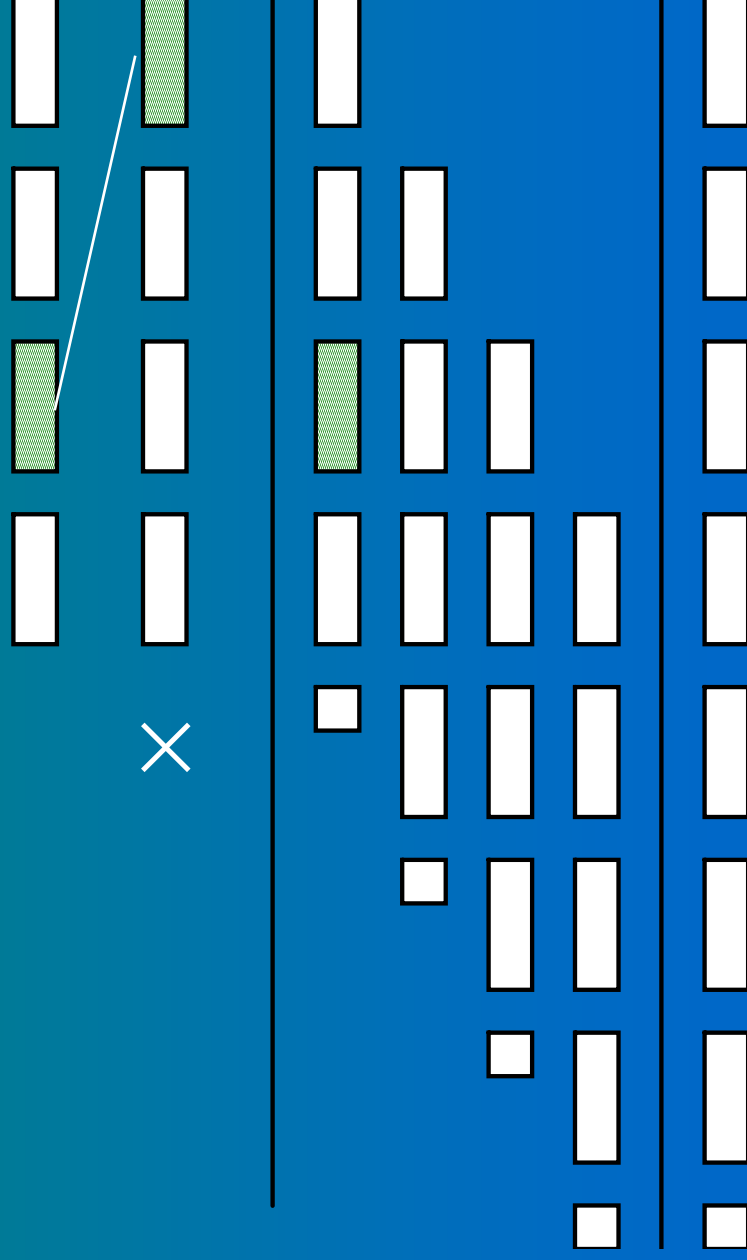
Large-Integer Multiplication



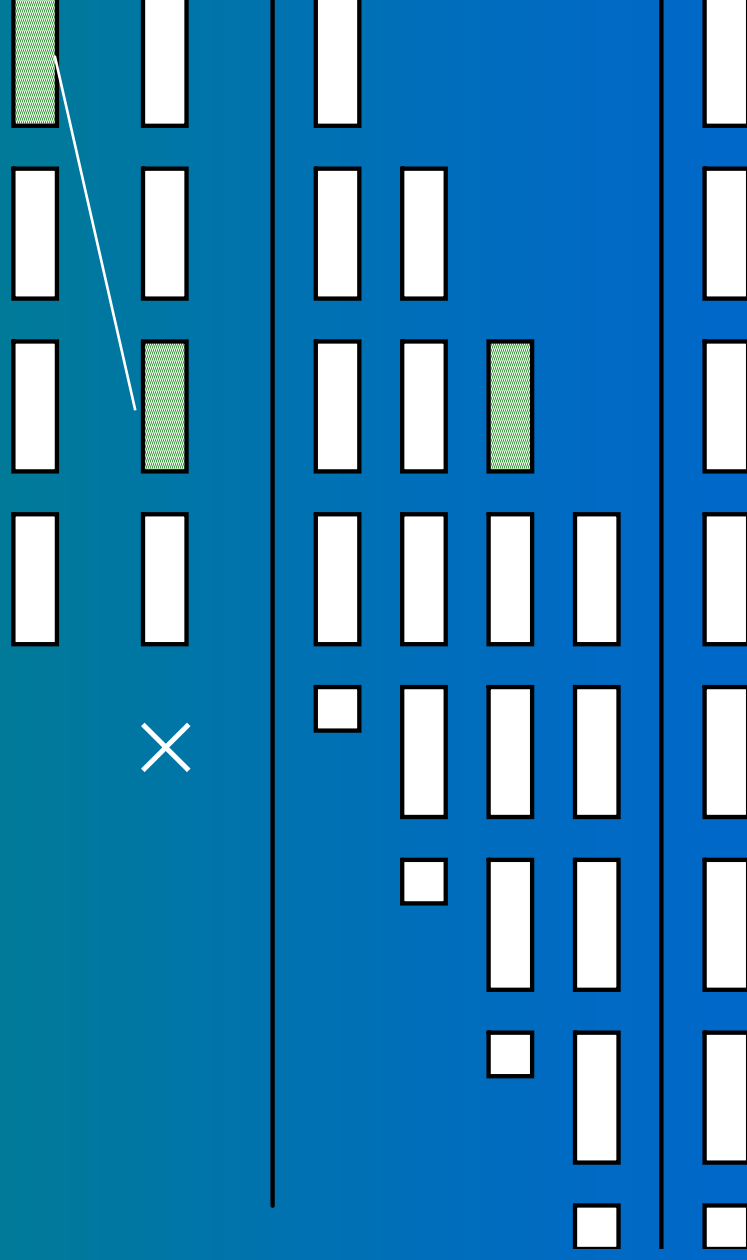
Large-Integer Multiplication

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.

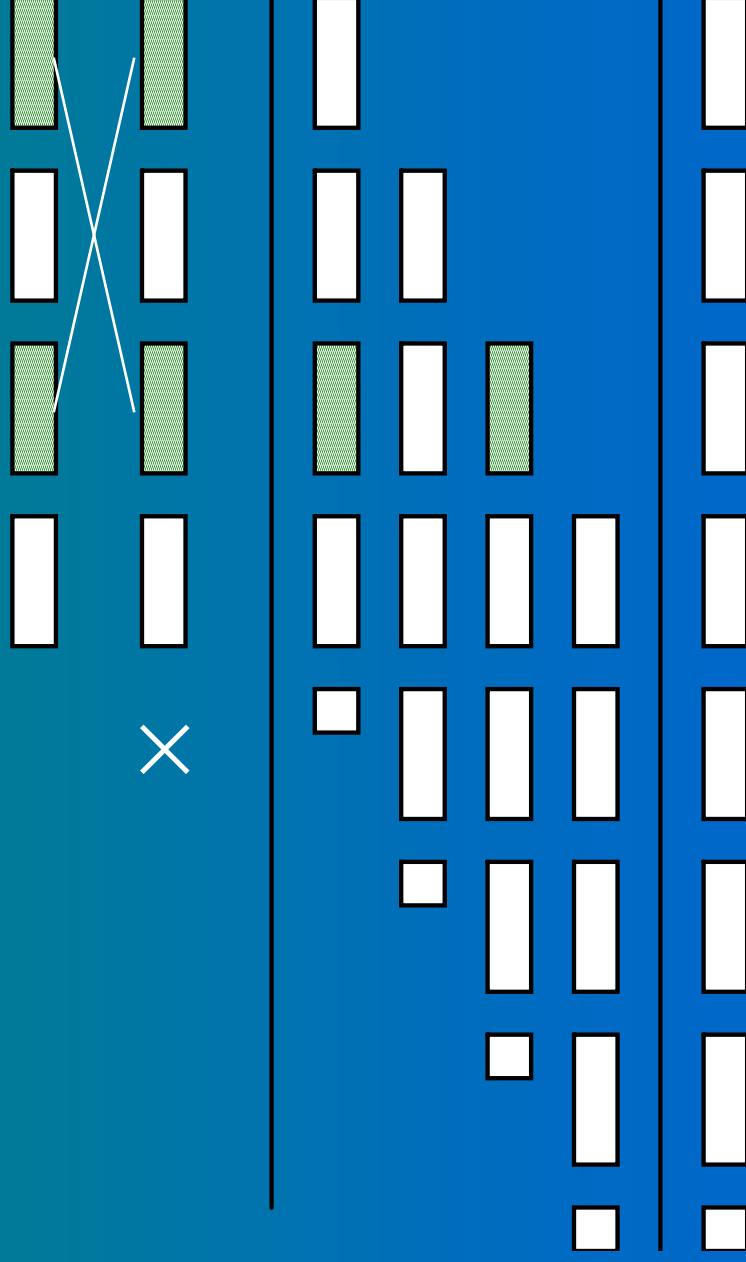
Large-Integer Squaring



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 $Y^X \bmod N$

- 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

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

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

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

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

$$(A \cdot 2^k + B)(C \cdot 2^k + D) = AC \cdot 2^{2k} + (AD + BC) \cdot 2^k + BD$$

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 $X = A \pmod{P}$ and $X = B \pmod{Q}$ then (as long as P and Q have no common factors) X can be derived as

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

Chinese Remaindering

If $N = PQ$, then a computation $\bmod N$ can be accomplished by performing the same computation $\bmod P$ and again $\bmod Q$ and then using Chinese Remaindering to derive the answer to the $\bmod N$ 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 $(A \cdot B) \bmod N$ requires much more than twice the time to compute $A \cdot B$.

Division is slow and cumbersome.

Modular Reduction

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

Division is disgusting.

Modular Reduction

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

Division is slow and cumbersome.

Modular Reduction

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

Division is dreadful.

Modular Reduction

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

Division is slow and cumbersome.

Modular Reduction

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

Division is wretched.

Modular Reduction

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

Division is slow and cumbersome.

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

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

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

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

$$Z \equiv Y^X \pmod{N}$$

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 $A = Y^a \bmod N$.

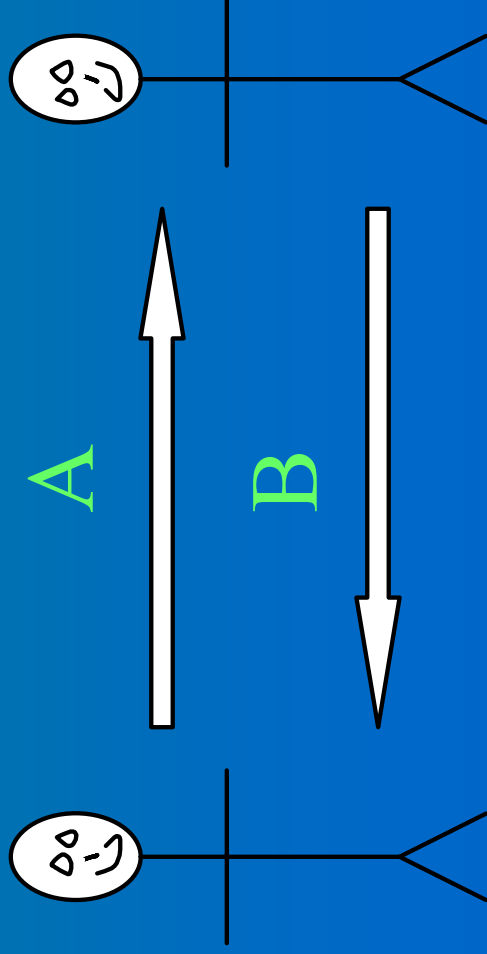
Bob

- 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

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

One-Way Trap-Door Functions

$$Z \equiv Y^X \pmod{N}$$

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

- Select two large random primes **P** & **Q**.

Anyone

RSA Public-Key Cryptosystem

Alice

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

Anyone

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

$$Z = Y^X \text{ mod } N.$$

RSA Public-Key Cryptosystem

Alice

- Select two large random primes P & Q .
- Publish the product $N=PQ$.
- To send message Y to Alice, compute $Z=Y^X \bmod N$.
- Send Z and X to Alice.

Anyone

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 mod 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

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

Some RSA Details

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

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

whenever

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

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

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

<u>Name</u>	<u>Public Key</u>	<u>Encryption</u>
Alice	N_A	$E_A(Y) = Y^E \pmod{N_A}$
Bob	N_B	$E_B(Y) = Y^E \pmod{N_B}$
Carol	N_C	$E_C(Y) = Y^E \pmod{N_C}$
:	:	:

Public Key Directory

<u>Name</u>	<u>Public Key</u>	<u>Encryption</u>
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	$E_C(Y) = Y^E \bmod N_C$
:	:	:

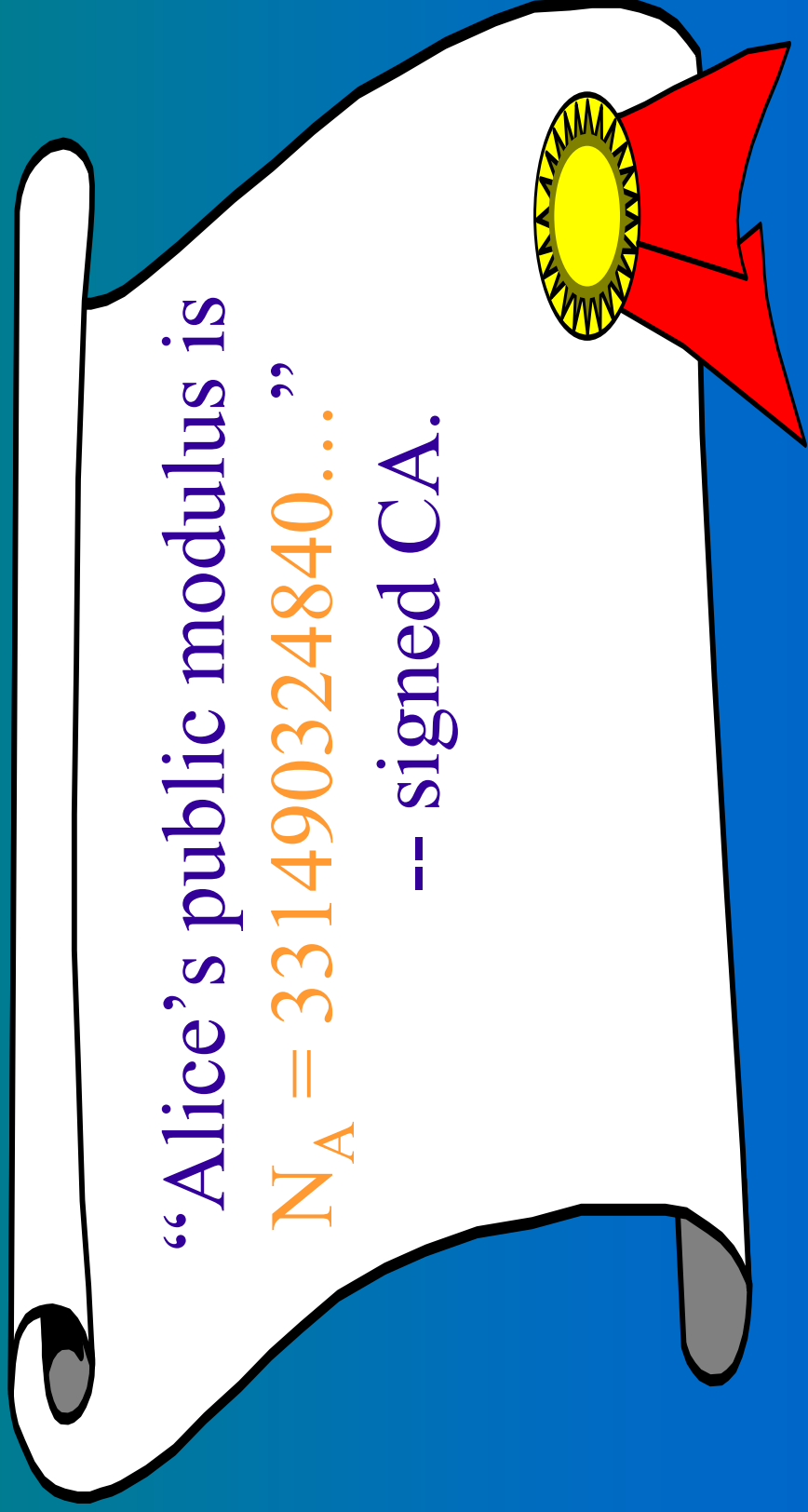
(Recall that E is commonly fixed to be $E=65537$.)

Certificate Authority

“Alice’s public modulus is

$N_A = 331490324840\dots$ ”

-- signed CA.



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?

Bob sends Alice the authentication string $y = \text{“I owe Bob \$1,000,000 - signed Alice.”}$

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

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

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

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 XX XX ... XX 00 YY YY ... YY

random
non-zero
bytes

message

“Man-in-the-Middle” Attacks

Alice \longleftrightarrow Bob

Alice \longleftrightarrow Eve \longleftrightarrow Bob

The Practical Side

Practical Aspects of Modern Cryptography

January 3, 2006

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

Practical Aspects of Modern Cryptography

January 3, 2006

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.