CS483 Analysis of Algorithms
Lecture 02 – Algorithms with numbers *

Jyh-Ming Lien

January 30, 2008

*this lecture note is based on *Algorithms* by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani and *Introduction to the Design and Analysis of Algorithms* by Anany Levitin.

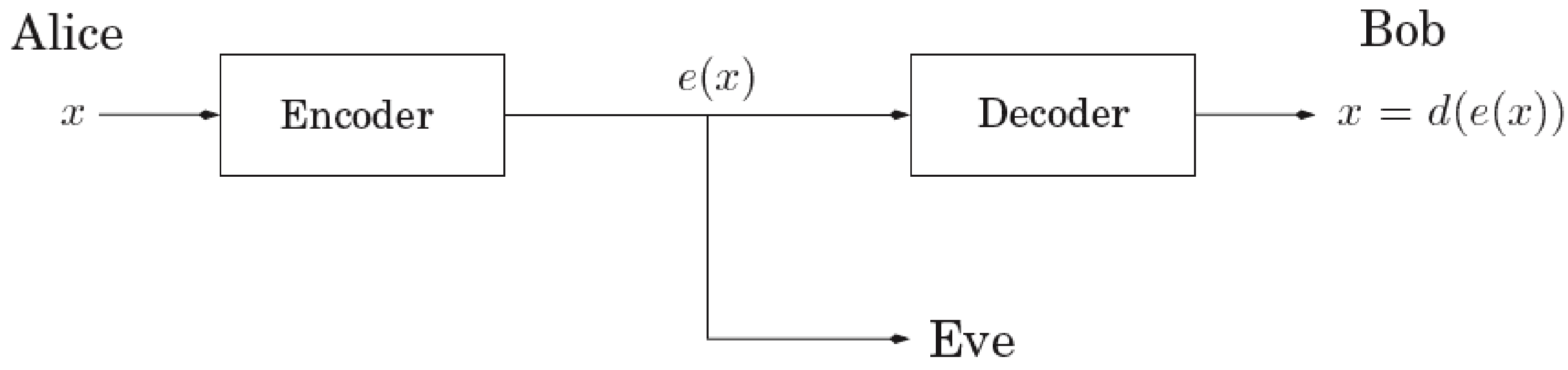
What will we learn today?

- Basic and modulo arithmetic
- Greatest common divisor (GCD)
- Check if a number is prime (an easier problem)
- Prime number factorization (a very hard problem)
- Generate random prime number with arbitrary length
- Cryptography:
  - Private/Public-key cryptography (symmetric/asymmetric cryptography).
  - RSA cryptosystem
  - Based on the fact that primality check can be done much more efficiently than factoring.

Cryptography

Typical setting in cryptography

- The typical setting



  - Alice and Bob wish to communicate in private
  - Eve will try to find out what they are saying
  - When Alice wants to send a message x, she encode it as $e(x)$
  - Bob then applies his decryption function $d(\cdot)$ to get his message $d(e(x)) = x$
  - Hopefully, Eve does not know how to convert $e(x)$ back to e, i.e., $d(\cdot)$

Private-key cryptography

- Alice and Bob choose a secret codebook (key) together
- **Example**: One time pad using *bitwise xor*
  - Encode $e_r(x) = x \oplus r$
  - Decode $e_r(e_r(x)) = (x \oplus r) \oplus r = x \oplus (r \oplus r) = x$
- **Example**:
  - $x = 11110000$
  - $r = 01110010$
  - Encoded message $e_r(x) = 11110000 \oplus 01110010 = 10000010$
  - Decoded message $e_r(e_r(x)) = 10000010 \oplus 01110010 = 11110000$
- Drawbacks of One time pad:
  - r needs to be discarded after use.
  - If r is used twice, $x_1 \oplus R$ and $x_2 \oplus R$, then Eve can easily know $x_1 \oplus x_2$.
- A more secure/popular private-key cryptography: Advanced Encryption Standard (AES) (by Rijmen and Daeme 1998)

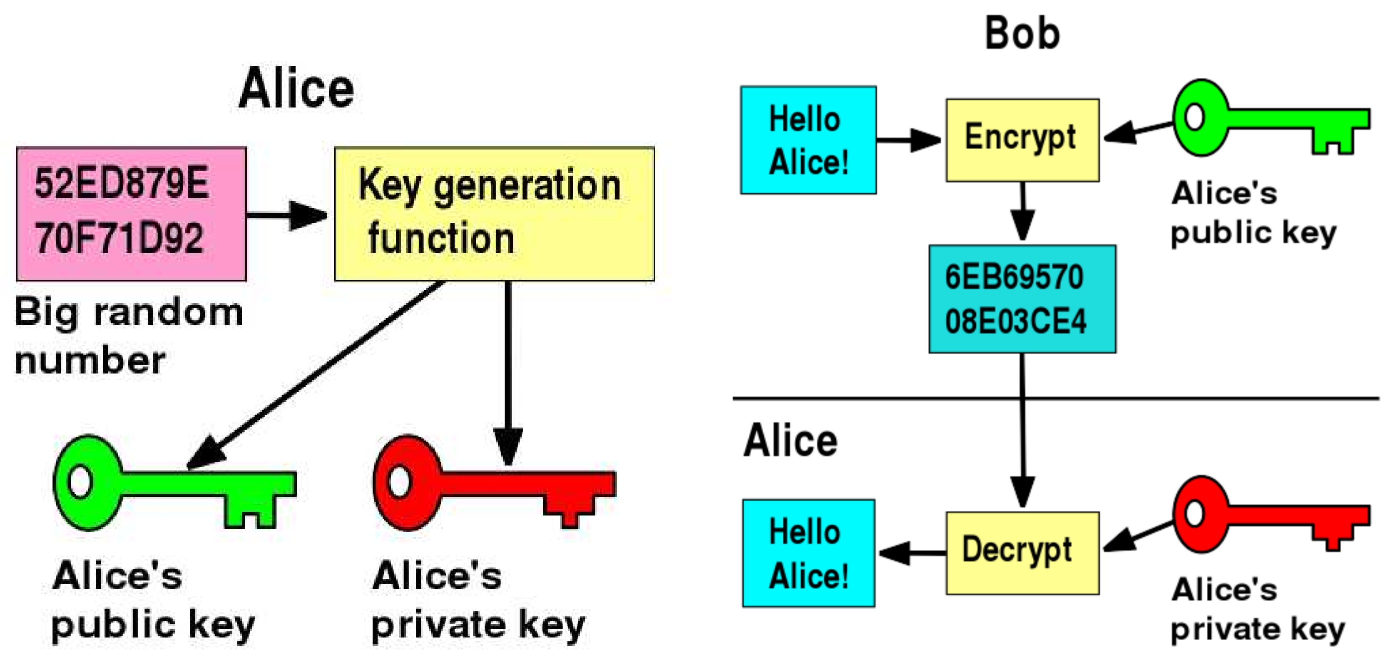
Public-key cryptography (PKC)

- For thousands of years, it was believed that the only way to establish secure communications was to first exchange a secret codebook (private key).
- PKC is a ground breaking idea in cryptography (by Merkle, Diffie and Hellman 1976)

(Ralph Merkle, Martin Hellman, Whitfield Diffie, Public Key Cryptography (PKC) Inventors (c) Chuck Painter/Stanford News Service.)

Public-key cryptography

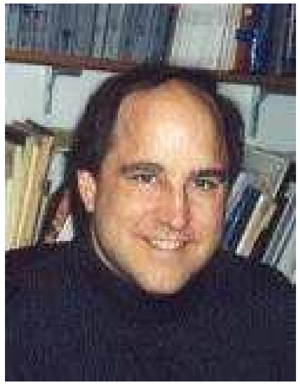
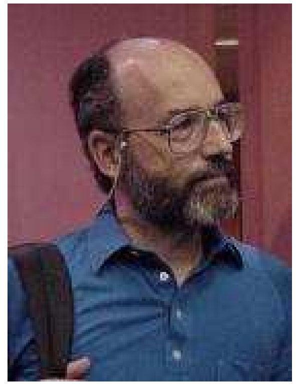
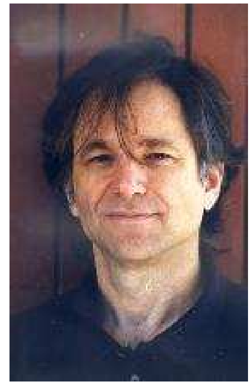
- **Example:**



(Images from Wikipedia)

RSA

- RSA is a type of PKC (by Rivest, Shamir, Adleman 1978)

Ronald Rivest Adi Shamir Len Adleman
(Images from *http://www.livinginternet.com/*)

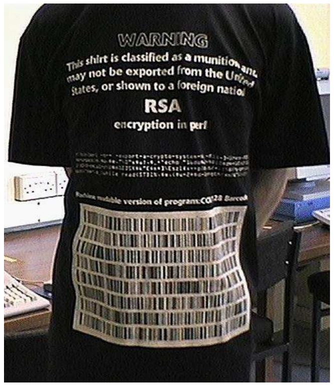
- A brief history of RSA:
  - RSA is inspired by Diffie and Hellman's paper on PKC
  - First publicized by Martin Gardner on Scientific American in 1977
  - NSA attempts to prevent RSA being distributed
  - RSA published on CACM in 1978
  - RSA was written up by Adam Back in 5 line PERL program

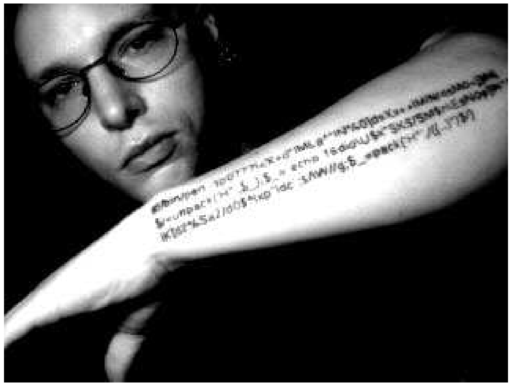
```
-export-a-crypto-system-sig -RSA-3-lines-PERL

#!/bin/perl -sp0777i<X+d*lMLa^*lN%0]dsXx++lMlN/dsM0<j]dsj
$/=unpack('H*',$_);$_=`echo 16dio\U$k"SK$/SM$n\EsN0p[lN*1
lK[d2%Sa2/d0$^Ixp"|dc`;s/\W//g;$_=pack('H*',/((..)*)$/)
```

(3-line version, from http://www.cypherspace.org/adam/rsa/)

RSA

- As usual, the US Government prohibited exporting the code outside of the country
- People started to protest and put the PERL code:
  - in their e-mail signatures,
  - on t-shirts, and
  - on their skins...





(Images from http://www.cypherspace.org/adam/rsa/)

- In Sep 2000, the US patent for RSA expired

RSA

- Making RSA keys
  - Bob picks two prime numbers p and q and lets $N = pq$.
  - Let e be any relative prime to $(p-1)(q-1)$
  - Let $d = (e\%(p-1)(q-1))^{-1}$
  - **Bob's public key**: e and N
  - **Bob's private key**: d
- Communicate using RSA keys
  - Alice encodes a message x: $e(x) = x^e\%N$
  - Bob decodes a message: $d(e(x)) = (e(x))^d\%N$
  - If Eve wants to decode a encrypted message, she will need to
    - Try all possible x until $x^e\%N = e(x)$
    - Try to find p and q from N using prime number factorization
- The security of RSA is based the following simple fact
  - *Given N, e, and $y = x^e\%N$, it is computationally intractable to determine x*

RSA

- RSA is based heavily on number theory
  - modulo arithmetic
  - prime number generation
- What do we need to in RSA?
  - An algorithm to generate prime numbers with arbitrary length
  - An algorithm to compute $x^y \% N$ for arbitrary large x and y
  - An algorithm to compute the inverse of a modulo, i.e., $(x\%N)^{-1}$

Basic Arithmetic

Integer addition

- Example:

$$
\begin{array}{lccccccccl}
\text{Carry:} & 1 & & & 1 & 1 & 1 & & \\
 & & 1 & 1 & 0 & 1 & 0 & 1 & (53) \\
 & & 1 & 0 & 0 & 0 & 1 & 1 & (35) \\
\hline
 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & (88)
\end{array}
$$

- Important observation: The sum of any three single-bit (digit) numbers is at most two bits (digits) long.
- Complexity:
- Can we do better?

Integer multiplication

- What is the time complexity of multiplying two integers using the algorithms we learned in elementary schools?
  Example: how do you compute this: 1101×1011?

$$
\begin{array}{rcccccccl}
 & & & & 1 & 1 & 0 & 1 & \\
 & & & \times & 1 & 0 & 1 & 1 & \\
\hline
 & & & & 1 & 1 & 0 & 1 & \text{(1101 times 1)} \\
 & & & 1 & 1 & 0 & 1 & & \text{(1101 times 1, shifted once)} \\
 & & 0 & 0 & 0 & 0 & & & \text{(1101 times 0, shifted twice)} \\
+ & 1 & 1 & 0 & 1 & & & & \text{(1101 times 1, shifted thrice)} \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \text{(binary 143)}
\end{array}
$$

- Complexity:

- Is there a better way of multiplying two integers than this elementary-school method?

Integer multiplication

- Russian peasant method (This is the method in Al Khwarizmi's book)
- Computing xy
  - If y is even, $x \cdot y = 2(x \cdot \frac{y}{2})$
  - If y is odd, $x \cdot y = x + 2(x \cdot \frac{y-1}{2})$
- Example: 123×77=

$$
xy = \\
\begin{array}{rrl}
y & x & z \\
77 & \cdot 123 & +0 \\
38 & \cdot 246 & +123 \\
19 & \cdot 492 & +123 \\
9 & \cdot 984 & +123 + 492 \\
4 & \cdot 1968 & +123 + 492 + 984 \\
2 & \cdot 3936 & +123 + 492 + 984 \\
1 & \cdot 7872 & +123 + 492 + 984 = 9471
\end{array}
$$

Integer multiplication

- Algorithm

Algorithm 0.1: MULTIPLY(x, y)

```
if y = 1
  then return (x)
z = MULTIPLY(x, ⌊y/2⌋)
if y%2 = 0
  then return (2z)
  else return (x + 2z)
```

- Time complexity:
  $O(n^2)$ given that x and y are both n bits long
- Advantage:
  very fast and easy hardware implementation!
- Can we do better?

Integer division

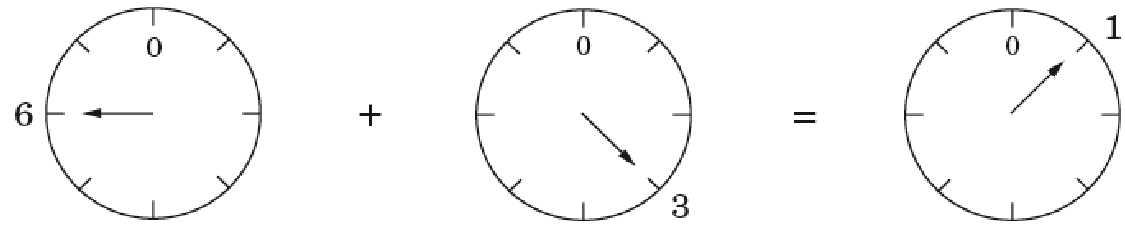
- Computing $(q, r) = x/y$
  - If x is even, $(q', r') = (x/2)/y \Rightarrow (q, r) = (2q', 2r')$
  - If x is odd, $(q', r') = ((x-1)/2)/y \Rightarrow (q, r) = (2q', 2r' + 1)$If $x < y$, $(q, r) = (0, x)$
- Example: $123/17$=

x	y	q	r
123	17	7	4
61	–	3	10
30	–	1	13
15	–	0	15

- Time complexity?

Modular Arithmetic

Figure 1.3 Addition modulo 8.



- N divides x if $x \mod N = 0$
- $x \mod N = x\%N = x - kN$
- If $x\%N = r$, then $(x - r)\%N = 0$
- It is usually convenient to write:

$$(x \equiv y \mod N) \text{ iff } (x \mod N) = (y \mod N).$$

- Example:
  - $31 \equiv 13 \mod 3$
  - $14 \equiv 59 \mod 5$

Modulo Addition/Multiplication

- If $x \equiv x' \mod N$ and $y \equiv y' \mod N$, then:

$$x + y \equiv x' + y' \mod N$$

and

$$xy \equiv x'y' \mod N$$

- More properties:
  - $x + (y + z) \equiv (x + y) + z \mod N$ (associativity)
  - $xy \equiv yx \mod N$ (commutativity)
  - $x(y + z) \equiv xy + xz \mod N$ (distributivity)

Modulo Addition/Multiplication

- Addition: $(x\%N) + (y\%N) = (x + y)\%N$
  - Complexity:

- Multiplication $(x\%N)(y\%N) = (xy\%N)$
  - Complexity:

Modulo Exponentiation

- Exponentiation: $x^y \% N$
  - Brute force: Compute x^y then compute $x^y \% N$
    - ▹ Problem: if x and y are 20 bits long, $x^y = 2^{(19)(524288)}$, which is about 10^7 bits long. In cryptography, x and y can be much longer than this.
  - Incremental: $x\%N \rightarrow x^2\%N \rightarrow x^3\%N \rightarrow \cdots \rightarrow x^y\%N$
    - ▹ Problem: If y is n bits long, we need to perform 2^n multiplications. This means the incremental method has time complexity exponential to n.

Modulo Exponentiation

- Decrease-n-conquer
  - If y is even,

$$x^y = (x^{\lfloor \frac{y}{2} \rfloor})^2$$

$$\Rightarrow x^y \% N = (x^{\lfloor \frac{y}{2} \rfloor} \% N)^2 \% N$$

  - If y is odd,

$$x^y = x \cdot (x^{\lfloor \frac{y}{2} \rfloor})^2$$

$$\Rightarrow x^y \% N = x \cdot (x^{\lfloor \frac{y}{2} \rfloor} \% N)^2 \% N$$

Algorithm 0.2: MODEXP(x, y, N)

if $y = 1$
  then return (x)
$z \leftarrow$ MODEXP$(x, \lfloor \frac{y}{2} \rfloor, N)$
if y is even
  then return $(z^2 \% N)$
  else return $((x \cdot z^2) \% N)$

Greatest Common Divisor & Modular division

Definition

- **Greatest Common Divisor Problem**: Given two non-negative integers m and n, find the largest integer, denoted as $\gcd(m, n)$, that can evenly divide both m and n.
- Example: If $m = 98$ and $n = 42$, then $\gcd(m, n) =$
- How do we design an algorithm to solve this problem?

Solution 1 - Brute force

- **Observation**: the range of $\gcd(m, n)$ is in $[1, \min(m, n)]$

Algorithm 0.3: $\gcd(m, n)$

for $i = \{\min(m, n), \cdots, 1\}$
do $\begin{cases} \textbf{if } m\%i = 0 \text{ and } n\%i = 0 \\ \quad \textbf{then return } (i) \end{cases}$

- How long does the algorithm take?

- Can we do better?

Solution 2 - Prime factorization

- **Observation**: use the strategy that we learned in the middle schools, i.e., "Prime factorization".
- **Example**: $m = 98 = 2 \times 7 \times 7$ and $n = 42 = 2 \times 3 \times 7$
  $\Rightarrow \gcd(m, n) = 2 \times 7 = 14$
- **Algorithm**: $\gcd(m, n)$

Algorithm 0.4: $\gcd(m, n)$

Perform prime factorization for m
Perform prime factorization for n
Find and multiply the common prime factors from m and n

- Well, the "algorithm" above is not really an algorithm yet, because we do not specify:

1. how to perform prime factorization on an integer?
2. how to find the common numbers from two lists of integers?

Solution 2 - Prime factorization

- **Problem**: Given an integer n, find a sequence of prime numbers S, whose multiplication is n.
- Find a list of prime numbers P that are smaller than n

Algorithm 0.5: PRIME FACTORIZATION(n)

$i \leftarrow 2$
while $i < n$
do $\begin{cases} \textbf{if } n\%i = 0 \\ \quad \textbf{then } \begin{cases} S \leftarrow i \\ n \leftarrow \frac{n}{i} \end{cases} \\ \quad \textbf{else } i \leftarrow \text{next prime number} \end{cases}$

Solution 2 - Prime factorization

- **Problem**: Given two lists of numbers, P_m and P_n, find a list of the common numbers P_c from P_m and P_n.
- **Example**: $P_m = \{2, 7, 7\}, P_n = \{2, 3, 7\} \Rightarrow P_c = \{2, 7\}$
- Algorithm

Algorithm 0.6: COMMON ELEMENTS(P_m, P_n)

comment: initially we create an empty list P_c

for each $i \in P_m$

$$\textbf{do} \begin{cases} \textbf{if } i \in P_n \\ \quad \textbf{then} \begin{cases} P_c \leftarrow i \\ \text{remove } i \text{ from } P_n \end{cases} \end{cases}$$

- **Observation 1**: $\gcd(m, n) = \gcd(n, m\%n)$
- **Observation 2**: $\gcd(m, 0) = m$

Proof. We want to show that $\gcd(m, n)$ can divide $m\%n$ evenly. Let $m = x \times \gcd$ and $n = y \times \gcd \Rightarrow m\%n = (m - z \times n) = (x \times \gcd - z \times (y \times \gcd) = (x - z \times y) \times \gcd$. □

(image of Euclid)

- **Example**: $\gcd(98, 42) = \gcd(42, 14) = \gcd(14, 0) = 14$
- Algorithm

Algorithm 0.7: $\gcd(m, n)$

while $n \neq 0$
do $\begin{cases} r = m\%n \\ m = n \\ n = r \end{cases}$
return (m)

Solution 3 - Euclidean Algorithm

- Time complexity of Algorithm 0.7?
  - Hint: If $a \geq b$, then $a\%b < a/2$

An extension of Euclid's algorithm

- GCD is key to dividing in the modular world
- **Lemma**: If d divides both a and b and $d = ax + by$ for some integers x and y, then $d = \gcd(a, b)$.
  - *proof*:
- Example: $\gcd(13, 4) = 1$, $13 \cdot 1 + 4 \cdot (-3) = 1$

Algorithm 0.8: $\textsc{Ext}\text{-}\mathrm{gcd}(a, b)$

if $b = 0$
  then return $(1, 0, a)$
$(x', y', d) = \textsc{Ext}\text{-}\mathrm{gcd}(b, a\%b)$
return $(y', x' - \lfloor a/b \rfloor y', d)$

Solution 3 - Euclidean Algorithm

- Is Algorithm 0.8 correct?

- Time complexity of Algorithm 0.8?

Modulo division

- In real number arithmetic, $b/a = b \cdot 1/a = b \cdot a^{-1}$

- For modulo division, $(b\%N)/(a\%N) = (b\%N)(a^{-1}\%N)$
  - We need to define a^{-1}
  - $x = a^{-1}$ if $ax \equiv 1 \mod N$
  - $ax \equiv 1 \mod N \Rightarrow ax + Ny = 1 \Rightarrow \gcd(a, N) = 1$

- Modular division theorem. For any $a \mod N$, a is invertible if a and N are relatively prime. If a is invertible, a^{-1} can be found in time $O(n^3)$ ($n = \log N$) using the extended Euclid algorithm.

Generate random primes

Primality testing

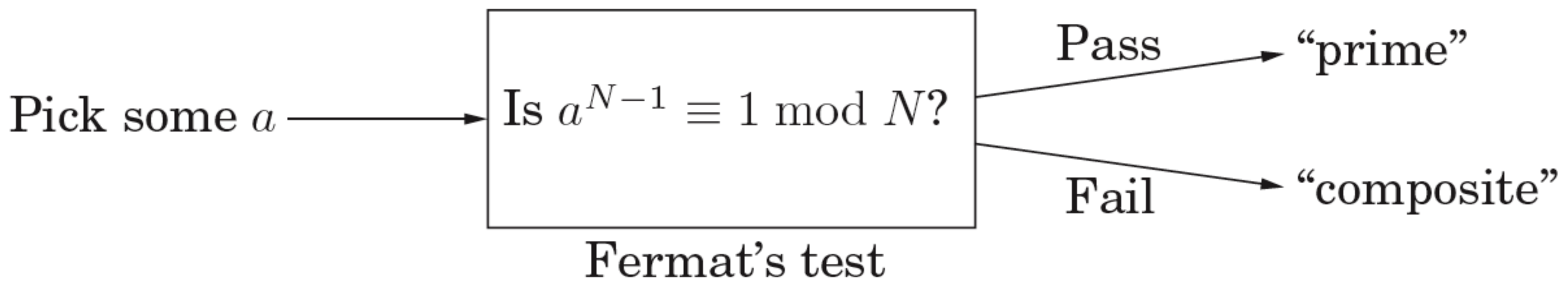
- Given a number p how do we know if p is a prime?
- We wish to answer this without trying to factor p.
- We do this based on Fermat's little theorem (AD 1640)
  - If p is a prime, then for every $1 \leq a < p$,

$$a^{p-1} \equiv 1 \mod p$$

  - *proof.*

Primality testing

- Our 1st attempt



- **Problem**: Note that the theorem is "If p is prime, then ...." But our test above is taking another direction "If $a^{N-1} \equiv 1 \mod N$, then N is prime.
- **Consequence**: Some non-prime (composite) number may have some such a which satisfies the "If" statement above.
  - In fact, there are a set of (very rare) numbers that have *all* such $1 \leq a < p$ which satisfies the "If" statement above. These numbers are called "Carmichael numbers." (We will ignore these numbers for now)

Primality testing

- **Lemma**: If $a^{N-1} \not\equiv 1 \mod N$ for some a which is relatively prime to N, then there must have at least $\frac{N}{2}$ of such $a < N$.
  - *proof*:

- This basically means:
  - If N is prime, $a^{N-1} \equiv 1 \mod N$ for all $a < N$
  - If N is not prime, $a^{N-1} \equiv 1 \mod N$ for $< \frac{N}{2}$ number of $a < N$

Primality testing

- Our strategy: Run our 1st algorithm k times
  - Pr(1st algorithm returns 'yes' and N is prime)=1
  - Pr(1st algorithm returns 'yes' and N is not prime) $\leq \frac{1}{2}$
  - Pr(All k instances of 1st algorithm return 'yes' and N is not prime) $\leq \frac{1}{2^k}$
  - The error decreases 'exponentially'

- Our 2nd attempt

Algorithm 0.9: PRIMIALITY$2(N)$

Pick k positive integers $a_1, a_2, \ldots a_k < N$ at random
if $a_i^{N-1} \equiv 1 \mod N$ for all $i = 1, 2, \ldots, k$
  then return (yes)
  else return (no)

Generate a random prime

- Observation: There are many prime numbers.
  - **Lagrange's prime number theorem**. Let $\pi(x)$ be the number of primes $\leq x$, then $\pi(x) \approx \frac{x}{\ln x}$.
  - Given a n-bit long number N, there are about $\frac{N}{n}$ prime numbers
- Now we describe a brute force method to generate a random prime number:

Algorithm 0.10: RANDOMPRIME(n)

for $i = \{1, 2, 3, \ldots, N\}$

do $\begin{cases} N \leftarrow \text{a random bit stream with length } n \\ \textbf{if } \text{PRIMIALITY2}(N) \\ \quad \textbf{then return } (N) \end{cases}$

- What is the time complexity of RANDOMPRIME?

Conclusion

Back to RSA

- Making RSA keys
  - Two prime numbers p and q and $N = pq$.
  - e be any relative prime to $(p-1)(q-1)$
  - $d = (e\%(p-1)(q-1))^{-1}$
- Communicate using RSA keys
  - Alice encodes a message x: $e(x) = x^e\%N$
  - Bob decodes a message: $d(e(x)) = (e(x))^d\%N$
- Why does it work? We will show that $(x^e\%N)^d = x\%N$
  - *proof*:

Summary

- We talked about
  - Basic/Modulo arithmetic
  - GCD
  - Primality and prime number generation
  - Private/Public key cyrptography
  - RSA
- We've walked through Chapter 1.1-1.4. (Please read 1.5, hashing)