Announcements

- **Homework:** Required to show your work
  - Remember 3/2/1/0
- **Project #1**
  - Today we’ll work through the rest of the math

Practice

<table>
<thead>
<tr>
<th>a^1</th>
<th>(a^2)</th>
<th>(a^3)</th>
<th>(a^4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>1</td>
<td>61</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ a^{61} = 1 \cdot 11 \cdot 5 \cdot 3 = 3 \]

Key points:
- Represent exponent in binary
- Break up the problem into factors (one per binary digit)
- Compute the factors by repeated squaring
- Use the substitution rule

Objectives

- **Part 1:**
  - Introduce Fermat’s Little Theorem
  - Understand and analyze the Fermat primality tester
- **Part 2:**
  - Discuss GCD and Multiplicative Inverses, modulo N
  - Prepare to Introduce Public Key Cryptography
  - This adds up to a lot of ideas!

Part 1: Primality Testing

Fermat’s Little Theorem

If \( p \) is prime, then \( a^{p-1} \equiv 1 \pmod{p} \)
for any \( a \) such that \( 1 \leq a < p \)

How do you wish you could use this theorem?

Examples:

- \( p = 3, a = 2 \)
  \[ a^{p-1} \equiv 2^2 \equiv 4 \equiv 1 \pmod{3} \]
- \( p = 7, a = 4 \)
  \[ a^{p-1} \equiv 4^6 \equiv 4^{1+5} \equiv 4^1 \cdot (4^5) \equiv 1 \cdot 2^3 \equiv 1 \pmod{7} \]
Logic Review

\( a \Rightarrow b \) (a implies b)

Which is equivalent to the above statement?
- \( b \Rightarrow a \)  \( \times \)
- \( \neg a \Rightarrow \neg b \)  \( \times \)
- \( \neg b \Rightarrow \neg a \)  **Contrapositive**

Contrapositive of Fermat's Little Theorem

If \( p \) and \( a \) are integers such that \( 1 \leq a < p \) and \( a^{p-1} \mod p \neq 1 \), then \( p \) is not prime.

First Prime Number Test

\[
\text{function } \text{primality}\left(N\right) \\
\text{Input: Positive integer } N \\
\text{Output: yes/no} \\
\begin{align*}
\text{// a is random positive integer between 1 and N-1} \\
\text{a = uniform(1..N-1)} \\
\text{// } a^{N-1} \mod N \\
\text{if } \left( \text{modexp}(a, N-1, N) == 1 \right): \\
\text{\quad return "possibly prime"} \\
\text{else:} \\
\text{\quad return "not prime" // certain}
\end{align*}
\]

False Witnesses

- If primality\( (N) \) returns "possibly prime", then \( N \) might or might not be prime, as the answer indicates
- Consider 15: \( N=15 \), \( a=4 \)
  - \( 4^{15} \mod 15 = 1 \)
  - but 15 clearly isn’t prime!
  - 4 is called a false witness of 15

- Given a non-prime \( N \), we call a number \( a \) where \( a^{N-1} \mod N = 1 \) a “false witness” (to the claim that \( N \) is prime)

Relatively Prime

- Two numbers \( a \) and \( N \) are relatively prime iff their greatest common divisor is 1.
  - 3 and 5?  \( \checkmark \)
  - 4 and 8?  \( \times \)
  - 4 and 9?  \( \checkmark \)
- Consider the Carmichael numbers:
  - They pass the test (i.e., \( a^{N-1} \mod N = 1 \)) for all \( a \) relatively prime to \( N \) (\( 1 \leq a < N \))
False Witnesses

- Ignoring Carmichael numbers,

- How common are false witnesses?
  - Lemma: If \( a^{n-1} \mod n = 1 \) for some \( a \) relatively prime to \( n \), then it must hold for at least half the choices of \( a < n \)

State of Affairs

- Summary:
  - If \( n \) is prime, then \( a^{n-1} \mod n = 1 \) for all \( a < n \)
  - If \( n \) is not prime, then \( a^{n-1} \mod n = 1 \) for at most half the values of \( a < n \)

  - Allows us to put a bound on how often our primality() function is wrong.

Correctness

- **Question #1**: Is the “Fermat test” correct?
  - No

- **Question #1’**: How correct is the Fermat test?
  - The algorithm is \( \frac{1}{2} \)-correct with one-sided error.
    - The algorithm has 0.5 probability of saying “yes \( N \) is prime” when \( N \) is not prime.
    - But when the algorithm says “no \( N \) is not prime”, then \( N \) must not be prime (by contrapositive of Fermat’s Little Theorem)

Amplification

- Repeat the test
  - Decrease the probability of error:

<table>
<thead>
<tr>
<th>1st run</th>
<th>2nd run</th>
<th>P(Error)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C )</td>
<td>( P )</td>
<td>( C )</td>
</tr>
<tr>
<td>( P )</td>
<td>( C )</td>
<td>( P )</td>
</tr>
</tbody>
</table>

- *Amplification of stochastic advantage*

**P(Correct)**

- \( k \) trials gives \( 1/(2^k) \) probability of being incorrect when the answer is "prime"

\[
P(\text{Correct}) = 1 - P(\text{Error}) = 1 - \frac{1}{2^k}
\]

Modified Primality Test

```python
function primality2(N):
    Input: Positive integer N
    Output: yes/no
    for i = 1 to k do:
        a = uniform(1..N-1)
        if (modexp(a, N-1, N) == 1):
            // possibly prime; do nothing
        else:
            return "not prime"
    return yes
```

\[\text{with correctness } 1 - \frac{1}{2^k}\]
2. Greatest Common Divisor

- **Euclid's rule:**
  - \( \text{gcd}(x, y) = \text{gcd}(x \mod y, y) = \text{gcd}(y, x \mod y) \)

- Can compute \( \text{gcd}(x, y) \) for large \( x, y \) by modular reduction until we reach the base case! 
  \[ \text{gcd}(a, 0) = a \]

```
function Euclid (a,b)
  Input: Two integers a and b with a \geq b \geq 0 (n-bit integers)
  Output: gcd(a,b)
  if b=0: return a
  return Euclid(b, a mod b)
```

Example

- \( \text{gcd}(25, 11) \)
  - \( = \text{gcd}(11, 3) \)
  - \( = \text{gcd}(3, 2) \)
  - \( = \text{gcd}(2, 1) \)
  - \( = \text{gcd}(1, 0) = 1 \)

3 Questions

- 1. Is it Correct?
- 2. How long does it take?
- 3. Can we do better?

Analysis

```
function Euclid (a,b)
  Input: Two integers a and b with a \geq b \geq 0 (n-bit integers)
  Output: gcd(a,b)
  if b=0: return a
  return Euclid(b, a mod b)
```
Bezout's Identity

- For two integers $a$, $b$ and their GCD $d$, there exist integers $x$ and $y$ such that:
  
  $$ax + by = d$$

Extended Euclid Algorithm

```python
function extended-Euclid (a, b)
Input: Two positive integers $a$ & $b$ with $a \geq b \geq 0$ (n-bits)
Output: Integers $x$, $y$, $d$ such that $d = \gcd(a, b)$
  and $ax + by = d$
  
  if $b=0$: return $(1,0,a)$
  $(x', y', d) = \text{extended-Euclid}(b, \, a \mod b)$
  return $(y', x' - \text{floor}(a/b)y', \, d)$
```

Example

- Note: there’s a great worked example of how to use the extended-Euclid algorithm on Wikipedia here: http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

- And another linked from the reading column on the schedule for today

Finding Multiplicative Inverses (Modulo $N$)

- Modular division theorem: For any $a \mod N$, $a$ has a multiplicative inverse modulo $N$ if and only if $a$ is relatively prime to $N$.

- Significance of extended-Euclid algorithm:
  - When two numbers, $a$ and $N$, are relatively prime
  - extended-Euclid algorithm produces $x$ and $y$ such that $\frac{ax + by}{d} = 1$
  - Thus, $x \equiv 1 \pmod{N}$
  - Because $h \cdot y \equiv 0 \pmod{N}$ for all integers $y$
  - Then $x$ is the multiplicative inverse of $a \mod N$

- I.e., I can use extended-Euclid to compute the multiplicative inverse of $a \mod N$

Multiplicative Inverses

- In the rationals, what’s the multiplicative inverse of $a$?
  
  $$a \cdot \frac{1}{a} = 1$$

- In modular arithmetic $(\text{modulo } N)$, what is the multiplicative inverse?
  
  $$a \cdot x \equiv 1 \pmod{N}$$

  e.g.
  
  $3 \cdot 5 \equiv 1 \pmod{7}$
  $2 \cdot 4 \equiv 1 \pmod{7}$

Multiplicative Inverses

- The multiplicative inverse mod $N$ is exactly what we will need for RSA key generation

- Notice also: when $a$ and $N$ are relatively prime, we can perform modular division in this way
Next

- RSA

Assignment

- HW #3: 1.9, 1.18, 1.20

- Finish your project #1 whiteboard experience on time.

- Finish project #1 for the early bonus and the win!