Overview

Today:
- Overview/demo of research tools
- The RSA Algorithm - key sizes and factoring

Next:
- Read Sections 2.8, 10.1, and 10.2
- Complete ungraded homework 6
- Remember to be working on graded homework 2 (due next Thurs)
First up... some demos of research tools

Tools being demonstrated:
- Zotero (managing papers, citations, etc.)
- LaTeX and paper format templates
- BibTeX
**Miller-Rabin Primality Testing**: There is an efficient randomized algorithm for testing if large numbers are prime (with very low probability of error).
- So: There is an efficient algorithm for **finding** large random prime numbers.

**Euler’s totient function**: $\phi(n) = \text{number of integers from } 1..n-1 \text{ that are relatively prime to } n$.

**Euler’s Theorem**: For every $a$ and $n$ that are relatively prime (i.e., $\gcd(a,n)=1$),
\[
a^{\phi(n)} \equiv 1 \pmod{n}.
\]
RSA Algorithm

Key Generation:
- Pick two large primes \( p \) and \( q \)
- Calculate \( n = p \times q \) and \( \phi(n) = (p-1) \times (q-1) \)
- Pick a random \( e \) such that \( \gcd(e, \phi(n)) \)
- Compute \( d = e^{-1} \pmod{\phi(n)} \) \[Use extended GCD algorithm!\]
- Public key is \( PU = (n, e) \); Private key is \( PR = (n, d) \)

Encryption of message \( M \in \{0,..,n-1\} \):
\[ E(PU, M) = M^e \mod n \]

Decryption of ciphertext \( C \in \{0,..,n-1\} \):
\[ D(PR, C) = C^d \mod n \]
RSA Algorithm

Key Generation:
- Pick two large primes $p$ and $q$
- Calculate $n = p \times q$ and $\phi(n) = (p-1)(q-1)$
- Pick a random $e$ such that $\gcd(e, \phi(n))$
- Compute $d = e^{-1} \pmod{\phi(n)}$ \[\text{[Use extended GCD algorithm!]}\]
- Public key is $PU = (n, e)$ ; Private key is $PR = (n, d)$

Encryption of message $M \in \{0, \ldots, n-1\}$:
- $E(PU, M) = M^e \mod n$

Decryption of ciphertext $C \in \{0, \ldots, n-1\}$:
- $D(PR, C) = C^d \mod n$

Correctness - easy when $\gcd(M, n) = 1$:
- $D(PR, E(PU, M)) = (M^e)^d \mod n$
  - $= M^{ed} \mod n$
  - $= M^{k\phi(n)+1} \mod n$
  - $= (M^{\phi(n)})^k \cdot M \mod n$
  - $= M$

Also works when $\gcd(M, n) \neq 1$, but slightly harder to show...
RSA Example

Simple example:

\[ p = 73, \quad q = 89 \]
\[ n = p \times q = 73 \times 89 = 6497 \]
\[ \phi(n) = (p-1)(q-1) = 72 \times 88 = 6336 \]
\[ e = 5 \]
\[ d = 5069 \quad \text{[ Note: } 5 \times 5069 = 25,345 = 4 \times 6336 + 1 \text{ ]} \]

Encrypting message \( M = 1234 \):

\[ 1234^5 \mod 6497 = 1881 \]

Decrypting:

\[ 1881^{5069} \mod 6497 = 1234 \]

Note: If time allows in class, more examples using Python!
Status of breaking RSA and factoring

Observation: If we could factor fast, we could break RSA

- How: Factor the public modulus $n$, compute $\phi(n)$, and compute $d$

So factoring is sufficient to break RSA - is it necessary?
Status of breaking RSA and factoring

Observation: If we could factor fast, we could break RSA
- How: Factor the public modulus n, compute $\phi(n)$, and compute $d$

So factoring is **sufficient** to break RSA - is it **necessary**?
- Answer: no one knows!
- This would be a great result if it could be proved…
- Note: Rabin’s PK encryption system is based on a similar concept, and it has been shown that breaking it is equivalent to factoring
  - Rabin’s scheme isn’t used because it is very inefficient - bit-by-bit

<table>
<thead>
<tr>
<th>What we know</th>
<th>What we’d like</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast factoring ⇒ Break RSA</td>
<td>Break RSA ⇒ Fast factoring</td>
</tr>
<tr>
<td>Why? Look at logical contrapositive: Can’t factor fast ⇒ Can’t break RSA</td>
<td></td>
</tr>
</tbody>
</table>
How fast can we factor?

Consider an algorithm with running time $\Theta \left(2^{c \cdot n^\alpha} \cdot (\log n)^{1-\alpha}\right)$

With $\alpha = 1$: This is $2^{c \cdot n}$ -- pure exponential time
With $\alpha = 0$: This is $2^{c \cdot \log(n)} = n^c$ -- pure polynomial time

Algorithm discovery for factoring has generally involved lowering $\alpha$

- $\alpha = 1$: Brute-force search for factors (exponential time)
- $\alpha = \frac{1}{2}$: Quadratic Sieve (1981) - still the best for $n<300$ bits or so
- $\alpha = \frac{1}{3}$: General Number Field Sieve (1990) - best for large numbers

But: Constants also matter (esp. the $c$ in the exponent!)...

What are the real-world speeds and consequences?
Comparable Key Sizes
From NIST publication 800-57a

**Issue**: PK algorithms based on mathematical relationships, and can be broken with algorithms that are faster than brute force.

We spent time getting a feel for how big symmetric cipher keys needed to be ➔ How big do keys in a public key system need to be?

From NIST pub 800-57a:

<table>
<thead>
<tr>
<th>Security Strength</th>
<th>Symmetric Key Algorithms</th>
<th>FFC (e.g., DSA, D-H)</th>
<th>IFC (e.g., RSA)</th>
<th>ECC (e.g., ECDSA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤ 80</td>
<td>2TDEA&lt;sup&gt;21&lt;/sup&gt;</td>
<td>$L = 1024$</td>
<td>$k = 1024$</td>
<td>$f = 160-223$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$N = 160$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>3TDEA</td>
<td>$L = 2048$</td>
<td>$k = 2048$</td>
<td>$f = 224-255$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$N = 224$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>AES-128</td>
<td>$L = 3072$</td>
<td>$k = 3072$</td>
<td>$f = 256-383$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$N = 256$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>192</td>
<td>AES-192</td>
<td>$L = 7680$</td>
<td>$k = 7680$</td>
<td>$f = 384-511$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$N = 384$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>AES-256</td>
<td>$L = 15360$</td>
<td>$k = 15360$</td>
<td>$f = 512+$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$N = 512$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>