Course Logistics

- Homework 2 revised. Due next Tuesday midnight.
Private key cryptography revisited.

- Key distribution and management is a serious problem! N users - O(N^2) keys!
Public key cryptography

- Key management problem potentially simpler (not really that simple as we will see later!!!).
### A Simple Example

<table>
<thead>
<tr>
<th>Public Key</th>
<th>Private Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nancy</td>
<td>7036027</td>
</tr>
<tr>
<td>Erica</td>
<td>6723952</td>
</tr>
<tr>
<td>Mary</td>
<td>9753658</td>
</tr>
<tr>
<td>Olga</td>
<td>7490469</td>
</tr>
<tr>
<td>Peggy</td>
<td>7123456</td>
</tr>
<tr>
<td>Olivia</td>
<td>6752345</td>
</tr>
<tr>
<td>Kathy</td>
<td>2563859</td>
</tr>
</tbody>
</table>

- Anyone can map from plaintext to ciphertext.
- Decryption easy only with inverted phone book.
One-way functions and trapdoors.

- A function \( f() \) is said to be one-way if given \( x \) it is “easy” to compute \( y = f(x) \), but given \( y \) it is “hard” to compute \( x = f^{-1}(y) \).

- A trap-door one-way function \( f_K() \) is such that to compute
  - \( y = f_K(x) \) is easy if \( K \) and \( x \) are known.
  - \( x = f_K^{-1}(y) \) is easy if \( K \) and \( y \) are known.
  - \( x = f_K^{-1}(y) \) is hard if \( y \) is known but \( K \) is unknown.

- Given a trap-door one-way function one can design a public key cryptosystem.
Encryption and 1-way trap doors

- Two keys:
  - public encryption key $e$
  - private decryption key $d$
- Encryption easy when $e$ is known
- Decryption hard when $d$ is not known
- $d$ provides “trap door”: decryption easy when $d$ is known
- We’ll study the RSA public key encryption scheme. First we need some number theory.
Some Number Theory

- We’ll need some number theory to define a one-way trap-door function:
  - Elementary (Review?):
    - Divisors
    - Prime numbers
    - relative primes
    - Modular arithmetic
  - Advanced (Hand-waving overview)
    - Euler’s totient function
    - Lagrange’s theorem
Divisors

- $x$ divides $y$ (written $x \mid y$) if the remainder is 0 when $y$ is divided by $x$
  - $1 \mid 8$, $2 \mid 8$, $4 \mid 8$, $8 \mid 8$
- The **divisors** of $y$ are the numbers that divide $y$
  - divisors of 8: \{1,2,4,8\}
- For every number $y$
  - $1 \mid y$
  - $y \mid y$
Prime numbers

- A number is prime if its only divisors are 1 and itself:
  - 2, 3, 5, 7, 11, 13, 17, 19, ...
- Fundamental theorem of arithmetic:
  - For every number \( x \), there is a unique set of primes \( \{p_1, \ldots, p_n\} \) and a unique set of positive exponents \( \{e_1, \ldots, e_n\} \) such that
    \[
    x = p_1^{e_1} \ast \ldots \ast p_n^{e_n}
    \]
Common divisors

- The common divisors of two numbers $x,y$ are the numbers $z$ such that $z| x$ and $z| y$
  - common divisors of 8 and 12:
    - intersection of $\{1,2,4,8\}$ and $\{1,2,3,4,6,12\}$
    - $= \{1,2,4\}$
  - greatest common divisor: $\gcd(x,y)$ is the number $z$ such that
    - $z$ is a common divisor of $x$ and $y$
    - no common divisor of $x$ and $y$ is larger than $z$
      - $\gcd(8,12) = 4$
Relative primes

- x and y are relatively prime if they have no common divisors, other than 1.
- Equivalently, x and y are relatively prime if \( \gcd(x, y) = 1 \)
  - 9 and 14 are relatively prime
  - 9 and 15 are not relatively prime
Modular Arithmetic

- Definition: $x$ is congruent to $y \mod m$, if $m$ divides $(x-y)$. Equivalently, $x$ and $y$ have the same remainder when divided by $m$.

- Notation: $x \equiv y \pmod{m}$

- Example: $14 \equiv 5 \pmod{9}$

- We work in $\mathbb{Z}_m = \{0, 1, 2, ..., m-1\}$, the ring of integers modulo $m$ with binary operators $+$ and $*$ defined modulo $m$.

- Example: $\mathbb{Z}_9 = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$

- We abuse notation and often write $=$ instead of $\equiv$
Addition and Multiplication

- Many of the same properties as addition and multiplication of integers:
  - Commutative
  - Associative
  - Distributive
  - Additive inverses

- Some differences:
  - Some elements have multiplicative inverses
Addition in $\mathbb{Z}_m$:

- Addition is well-defined:

  \[
  \begin{align*}
  x &\equiv x' \pmod{m} \\
  y &\equiv y' \pmod{m} \\
  \text{then} \\
  x + y &\equiv x' + y' \pmod{m}
  \end{align*}
  \]

- $3 + 4 = 7 \pmod{9}$.
- $3 + 8 = 2 \pmod{9}$. 
Additive inverses in $\mathbb{Z}_m$

- 0 is the additive identity in $\mathbb{Z}_m$

$$x + 0 \equiv x \pmod{m} \equiv 0 + x \pmod{m}$$

- Additive inverse
  - Every element has unique additive inverse.
  - $4 + 5 = 0 \pmod{9}$.
  - 4 is additive inverse of 5.
Multiplication in $\mathbb{Z}_m$:

- Multiplication is well-defined:

  \[
  \text{if } x \equiv x' \pmod{m} \quad \text{and} \quad y \equiv y' \pmod{m}
  \]

  \[
  \text{then } x \times y \equiv x' \times y' \pmod{m}
  \]

- $3 \times 4 = 3 \mod 9$.
- $3 \times 8 = 6 \mod 9$.
- $3 \times 3 = 0 \mod 9$. 
Multiplicative inverses in $\mathbb{Z}_m$

- 1 is the multiplicative identity in $\mathbb{Z}_m$
  \[ x \cdot 1 \equiv x \pmod{m} \equiv 1 \cdot x \pmod{m} \]

- Multiplicative inverse –
  - SOME, but not ALL elements have unique multiplicative inverse.
  - In $\mathbb{Z}_9$: $3 \cdot 0 \equiv 0$, $3 \cdot 1 \equiv 3$, $3 \cdot 2 \equiv 6$, $3 \cdot 3 \equiv 0$, $3 \cdot 4 \equiv 3$, $3 \cdot 5 \equiv 6$, ..., so 3 does not have a multiplicative inverse.
  - On the other hand, $4 \cdot 2 \equiv 8$, $4 \cdot 3 \equiv 3$, $4 \cdot 4 \equiv 7$, $4 \cdot 5 \equiv 2$, $4 \cdot 6 \equiv 6$, $4 \cdot 7 \equiv 1$, so $4^{-1} \equiv 7$
Which numbers have inverses?

- In $\mathbb{Z}_m$, $x$ has a multiplicative inverse if and only if $x$ and $m$ are relatively prime
  - E.g., 3 and 4 in $\mathbb{Z}_9$
- If $\gcd(x, m) > 1$ then $\text{lcm}(x, m) < xm$, so there is a number $y$, $0 < y < m$ such that $m | xy$.
  - $yx = 0 \pmod{m}$
  - So $x$ does not have a multiplicative inverse
- If $\gcd(x, m) = 1$, as $y$ varies, $y^*x$ takes on $m$ distinct values, so for some value, $y^*x = 1 \pmod{m}$. 
Euler’s totient function

- Given positive integer n, Euler’s totient function $\Phi(n)$ is the number of positive numbers less than n that are relatively prime to n.

- Fact: If p is prime then
  - $\{1, 2, 3, \ldots, p-1\}$ are relatively prime to p.
Euler’s totient function

- Fact: If \( p \) and \( q \) are prime and \( n = pq \) then
  \[
  \Phi(n) = (p - 1)(q - 1)
  \]
- Each number that is not divisible by \( p \) or by \( q \) is relatively prime to \( pq \).
  - E.g. \( p = 5, q = 7 \): \{1, 2, 3, 4, -6, -8, 9, -11, 12, 13, -16, 17, 18, 19, - -22, 23, 24, -26, 27, -29, -31, 32, 33, 34, -\}
  - \((p-1)(q-1) = pq-p-q+1\)
Important Fact

- If $a$ is relatively prime to $n$ then
  $$a^{\Phi(n)} \equiv 1 \mod n$$

- (This is a corollary to a theorem due to Lagrange that states that the order of an element of a multiplicative group divides the order of the group. It’s applied to the group $\mathbb{Z}_n^*$ of residues mod $n$ that are relatively prime to $n$.)
**RSA overview**

- Alice wants people to be able to send her encrypted messages.
- She chooses two (large) prime numbers, p and q and computes \( n = pq \) and \( \Phi(n) \). [“large” = 100 digits +]
- She chooses a number \( e \) such that \( e \) is relatively prime to \( \Phi(n) \) and computes \( d \), the inverse of \( e \) in \( \mathbb{Z}_{\Phi(n)} \)
- She publicizes the pair \((e,n)\) as her public key. She keeps \( d \) secret and destroys \( p, q, \) and \( \Phi(n) \)
- Plaintext and ciphertext messages are elements of \( \mathbb{Z}_n \) and \( e \) is the encryption key.
RSA overview

- Bob wants to send a message $x$ (an element of $\mathbb{Z}_n$) to Alice.
- He looks up her encryption key, $(e,n)$, in a directory.
- The encrypted message is

$$y = E(x) = x^e \mod n$$

- Bob sends $y$ to Alice.
RSA overview

- To decrypt the message

\[ y = E(x) = x^e \mod n \]

she's received from Bob, Alice computes

\[ D(y) = y^d \mod n \]

Claim: \( D(y) = x \)
RSA encryption function is 1-way trap door

- Need to show
  - $D[E[x]] = x$
  - $E[x]$ and $D[y]$ can be computed efficiently if keys are known
  - $E^{-1}[y]$ cannot be computed efficiently without knowledge of the (private) decryption key $d$.

- Also, it should be possible to select keys reasonably efficiently
  - This does not have to be done too often, so efficiency requirements are less stringent.
E and D are inverses: Case 1: $\gcd(x,n)=1$

$$D(y) = y^d \mod n$$

$$\equiv (x^e \mod n)^d$$

$$\equiv (x^e)^d \mod n$$

$$\equiv x^{ed} \mod n$$

$$\equiv x^{\Phi(n)+1} \mod n \quad \text{Because } ed \equiv 1 \mod \Phi(n)$$

$$\equiv (x^{\Phi(n)})^t x \mod n$$

$$\equiv 1^t x \mod n \equiv x \mod n \quad \text{From “important fact”}$$
Theorem (Fermat)

- If $p$ is prime and $x$ is in $\mathbb{Z}_p$ then
  \[ x^p \equiv x \mod p \]
- Proof: If $p|\,x$ then $0^p \equiv 0 \mod p$
- Otherwise,
  \[
  \Phi(p) = p - 1 \\
  x^{p-1} = x^{\Phi(p)} \equiv 1 \mod p \\
  x^p \equiv x \mod p
  \]
Alternative Proof that E and D are inverses

\[ ed = t\Phi(n) + 1 = t(p - 1)(q - 1) + 1 \]

\[ x^{p-1} \equiv 1 \mod p \]

\[ (x^{p-1})^{t(q-1)} \equiv 1 \mod p \]

\[ x^{t\Phi(n)} \equiv 1 \mod p \]

\[ x^{t\Phi(n)+1} \equiv x \mod p \]

\[ x^{ed} \equiv x \mod p \]

\[ p \mid (x^{ed} - x) \]
- By analogous argument \( q \mid (x^{ed} - x) \)
- So
  \[ n \mid (x^{ed} - x) \]
  \[ x^{ed} \equiv x \mod n \]
Tiny RSA example.

- Let $p = 7$, $q = 11$. Then $n = 77$ and $\Phi(n) = 60$.
- Choose $e = 13$. Then $d = 13^{-1} \mod 60 = 37$.
- Let message $= 2$.
- $E(2) = 2^{13} \mod 77 = 30$.
- $D(30) = 30^{37} \mod 77 = 2$
Slightly Larger RSA example.

- Let $p = 47$, $q = 71$. Then $n = 3337$ and
  \[ \Phi(pq) = 46 \times 70 = 3220 \]
- Choose $e = 79$. Then $d = 79^{-1} \mod 3220 = 1019$.
- Let message = 688232... Break it into 3 digit blocks to encrypt.
  - $E(688) = 688^{79} \mod 3337 = 1570$.
  - $E(232) = 232^{79} \mod 3337 = 2756$
  - $D(1570) = 1570^{1019} \mod 3337 = 688$.
  - $D(2756) = 2756^{1019} \mod 3337 = 232$. 
RSA encryption function is 1-way trap door

- Need to show
  - $D[E[x]] = x$
  - $E[x]$ and $D[y]$ can be computed efficiently if keys are known
  - $E^{-1}[y]$ cannot be computed efficiently without knowledge of the (private) decryption key $d$.

- Also, it should be possible to select keys reasonably efficiently
  - This does not have to be done too often, so efficiency requirements are less stringent.
Decryption without trapdoor

- Suppose Oscar intercepts the encrypted message $y$ that Bob has sent to Alice.
- Oscar can look up $(e, n)$ in the public directory (just as Bob did when he encrypted the message).
- If Oscar can compute $d = e^{-1} \mod \Phi(n)$ then he can use the formula $D(y) = y^d \mod n = x$ to recover the plaintext $x$.
- If Oscar can compute $\Phi(n)$, he can compute $d$ (the same way Alice did).
Decryption without trapdoor

- Oscar knows that \( n \) is the product of two primes
- If he can factor \( n \), he can compute \( \Phi(n) \)
- But factoring large numbers is very difficult:
  - Grade school method takes \( O(\sqrt{n}) \) divisions.
  - Prohibitive for large \( n \), such as 200 digits (roughly 512 bits)
  - Better factorization algorithms exist, but they are still too slow for large \( n \)
  - Lower bound for factorization is an open problem
How big should $n$ be?

- Today we need $n$ to be at least 768 bits. Better 1024 or even 2048 bits.
- No other (implementation independent) attack on RSA known.
RSA encryption function is 1-way trap door

- Need to show
  - $D[E(x)] = x$
  - $E[x]$ and $D[y]$ can be computed efficiently if keys are known
  - $E^{-1}[y]$ cannot be computed efficiently without knowledge of the (private) decryption key $d$.

- Also, it should be possible to select keys reasonably efficiently
  - This does not have to be done too often, so efficiency requirements are less stringent.
Efficient exponentiation

Usual approach to computing $x^c$ is inefficient when $c$ is large.

Instead, represent $c$ as bit string $b_{k-1} \ldots b_0$ and use the following algorithm:

$z = 1$

For $i = k-1$ downto 0 do

$z = z^2 \mod n$

if $b_i = 1$ then $z = z^* x \mod n$
Example: $30^{37}$ mod 77

$z = z^2 \mod n$

if $b_i = 1$ then $z = z \times x \mod n$

<table>
<thead>
<tr>
<th>i</th>
<th>b</th>
<th>z</th>
<th>$=1 \times 1 \times 30 \mod 77$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>30</td>
<td>$=30 \times 30 \mod 77$</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>53</td>
<td>$=53 \times 53 \mod 77$</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>37</td>
<td>$=37 \times 37 \times 30 \mod 77$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>71</td>
<td>$=29 \times 29 \mod 77$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>$=71 \times 71 \times 30 \mod 77$</td>
</tr>
</tbody>
</table>
RSA encryption function is 1-way trap door

Need to show

- $D[E[x]] = x$
- $E[x]$ and $D[y]$ can be computed efficiently if keys are known
- $E^{-1}[y]$ cannot be computed efficiently without knowledge of the (private) decryption key $d$.

Also, it should be possible to select keys reasonably efficiently

- This does not have to be done too often, so efficiency requirements are less stringent.
Key selection

- To select keys we need efficient algorithms to:
  - Select large primes
    - Primes are dense so choose randomly.
    - Probabilistic primality testing methods known. Work in logarithmic time.
  - Compute multiplicative inverses
    - Extended Euclidean algorithm
Euclidean Algorithm: $\gcd(r_0, r_1)$

\[
\begin{align*}
  r_0 &= q_1 r_1 + r_2 \\
  r_1 &= q_2 r_2 + r_3 \\
  &\vdots \\
  r_{m-2} &= q_{m-1} r_{m-1} + r_m \\
  r_{m-1} &= q_m r_m + 0 \\
  \gcd(r_0, r_1) &= \gcd(r_1, r_2) = \cdots = \gcd(r_{m-1}, r_m) = r_m
\end{align*}
\]
Computing inverse of a mod n

- Main Idea: Looking for inverse of a mod n means looking for x such that $x \cdot a - y \cdot n = 1$.
- To compute inverse of a mod n, do the following:
  - Compute $\gcd(a, n)$ using Euclidean algorithm.
  - Since a is relatively prime to m (else there will be no inverse) $\gcd(a, n) = 1$.
  - So you can obtain linear combination of $r_m$ and $r_{m-1}$ that yields 1.
  - Work backwards getting linear combination of $r_i$ and $r_{i-1}$ that yields 1.
  - When you get to linear combination of $r_0$ and $r_1$ you are done as $r_0 = n$ and $r_1 = a$. 
Example – Inverse of 15 mod 37

- 37 = 2 * 15 + 7
- 15 = 2 * 7 + 1
- 7 = 7 * 1 + 0

Now,
- 15 – 2 * 7 = 1
- 15 – 2 (37 – 2 * 15) = 1
- 5 * 15 – 2 * 37 = 1

So, inverse of 15 mod 37 is 5 by definition!!
Extended Euclidean Algorithm (Textbook)

- Define:
  
  \[ t_0 = 0 \]
  
  \[ t_1 = 1 \]
  
  \[ t_j = t_{j-2} - q_{j-1} t_{j-1} \mod r_0 \]

- Can prove by induction: for \( 0 \leq j \leq m \)
  
  \[ r_j \equiv t_j r_1 \mod r_0 \]

- So if \( \gcd(r_0, r_1) = 1 \), then \( t_m \equiv r_1^{-1} \mod r_0 \)
Pseudocode for computing $b^{-1} \mod m$

(Textbook)

- Computation of the values $t_j$ can be incorporated into Euclidean algorithm (see code on next slide).
- In this code, the variables $q$, $b0$, and $r$ hold the values of the quotients and remainders; the variables $t0$, $t$, and $temp$ hold the values of $t_{i-1}$, $t_i$, $t_{i+1}$, respectively at the point labelled ** on the $i$th iteration of the loop.
- For our purposes, this will be executed with $b=e$ and $m=\Phi(n)$
Pseudocode for computing $b^{-1} \mod m$

(Textbook)

```plaintext
n0=m  // the modulus
b0=b  // the number we're inverting mod m
t0=0

t=1
q=n0 div b0  /*integer division */
r= n0-q*b0
while r > 0 do
  temp = t0-q*t
  if temp >=0 then temp = temp mod m
  else temp = n-((-temp) mod m)
  // *** see previous slide
  t0 = t
  t = temp
  n0 = b0
  b0 = r
  q = n0 div b0
  r = n0 - q*b0  /*end of loop */
if b0 = 1 then t is b inverse (mod m)
```
Example: $13^{-1} \mod 60$ (Textbook)

<table>
<thead>
<tr>
<th>(n0)</th>
<th>(b0)</th>
<th>(t0)</th>
<th>(t)</th>
<th>(q)</th>
<th>(r)</th>
<th>(\text{temp})</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>13</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>56</td>
</tr>
<tr>
<td>13</td>
<td>8</td>
<td>1</td>
<td>56</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>56</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>51</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>5</td>
<td>51</td>
<td>1</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>51</td>
<td>14</td>
<td>1</td>
<td>1</td>
<td>37</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>14</td>
<td>37</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Key selection

- To select keys we need efficient algorithms to
  - Select large primes
    - Primes are dense so choose randomly.
    - Probabilistic primality testing methods known. Work in logarithmic time.
  - Compute multiplicative inverses
    - Extended Euclidean algorithm
Probabilistic Primality Testing

- A probabilistic (or randomized) algorithm is an algorithm that uses random numbers.
- A yes-biased Monte-Carlo algorithm is a probabilistic algorithm for a decision problem, such that a “yes” answer is always correct, but a “no” answer may be incorrect.
- Error probability $\varepsilon$ means the probability of an incorrect (“no”) is at most $\varepsilon$. 
Solovay-Strassen primality test

- Consider the Jacobi function $J(r,p)$ defined as follows:

\[
J(r, p) = \begin{cases} 
1 & \text{if } r = 1 \\
J(r/2) \times (-1)^{(p^2-1)/8} & \text{if } r \text{ is even} \\
J(p \mod r, r) \times (-1)^{(r-1)(p-1)/4} & \text{if } r \text{ is odd and } r \neq 1
\end{cases}
\]

- Let $\text{Test}(r,p)$ be true iff $J(r, p) \equiv r^{(p-1)/2} \mod p$
and $\gcd(r,p) = 1$.

- Facts:
  - If $p$ is prime then $\text{Test}(r,p)$ is true for all $r$ s.t. $1 \leq r \leq p-1$.
  - If $p$ is an odd composite number then $\text{Test}(r,p)$ is true for at most half of the numbers $r$ s.t. $1 \leq r \leq p-1$. 
Solovay-Strassen primality test

- A yes-biased Monte-Carlo algorithm for the decision problem “Is \( p \) composite?” with error probability \( 1/2 \)
- Let \( p \) be a candidate which we’d like to check
- Algorithm:
  - choose a random integer \( r \), \( 1 \leq r \leq p-1 \)
  - If Test\((r,p)\) is false then answer “yes” (\( p \) is composite, i.e., is not prime)
  - else answer “no” (\( p \) is not composite, i.e. is prime)
- To reduce error probability repeat many times
How Alice can select primes p,q:

```c
int select-a-prime()
{
    while true {
        randomly choose large integer p
        for (i= 0;i<k;i++) {
            if SS says p is composite break
        };
        return p; /*p is prime with high probability */
    };
    /* end of while loop */
}
```

Since primes are “dense”, this will terminate after a reasonable number of iterations.
Key selection

✓ To select keys we need efficient algorithms to
  ✓ Select large primes
    ✓ Primes are dense so choose randomly.
    ✓ Probabilistic primality testing methods known. Work in logarithmic time.
  ✓ Compute multiplicative inverses
    ✓ Extended Euclidean algorithm
RSA encryption function is 1-way trap door

✓ Need to show
  ✓ $D[E(x)] = x$
  ✓ $E[x]$ and $D[y]$ can be computed efficiently if keys are known
  ✓ $E^{-1}[y]$ cannot be computed efficiently without knowledge of the (private) decryption key $d$.

✓ Also, it should be possible to select keys reasonably efficiently
  ✓ This does not have to be done too often, so efficiency requirements are less stringent.
Other Public Key cryptosystems

- **Knapsack**
  - Most versions not considered secure any more.

- **Discrete Log**
  - Depends on intractability of discrete log problem, that is, given $x$ and $x^y \mod n$ find $y$

- **Elliptic Curve.**
  - Involves taking discrete logs in “elliptic curve groups”.
  - Much more “security” than RSA for given key size.
  - Used in WAP.
Further Reading

- Handbook of Applied Cryptography – Menezes et. al. CRC Press.
- North American Crypto archive http://cryptography.org/
- Ron Rivest’s crypto page http://www.toc.lcs.mit.edu/~rivest/crypto-security.html
- Cryptography archive: http://www.austinlinks.com/Crypto/
- AES home page http://csrc.nist.gov/encryption/aes/