CS530
Public Key Cryptography

Bill Cheng

http://merlot.usc.edu/cs530-s10

Public Key Cryptography

- aka asymmetric cryptography
- Based on some NP-complete problem
  - traveling salesman problem
    - *n* cities, connected
    - find shortest tour, all cities must be visited
    - solution complexity is *n!*
  - unique factorization
    - factor an integer into product of prime numbers (unique solution)
  - discrete logarithms
    - for any integers *b*, *n*, *y*: Find *x* such that $b^x \bmod n = y$
    - modular arithmetic produces folding

A Short Note on Primes

- Why are public keys (and private keys) so large?
  - because key space is *sparse*
- What is the probability that some large number *p* is prime?
  - about 1 in *1/ln(p)*
    - 2 digit numbers: 25 primes (1 in 4)
    - 10 digit numbers: 1 in 23 are primes
    - 100 digit numbers: 1 in 230 are primes
    - but... the more digits, the more primes!
  - when $p \approx 2^{512}$ ($\approx 10^{150}$), equals about 1 in 355
    - about 1 in 355^2 numbers $\approx 2^{1024}$ is product of two primes (and therefore valid RSA modulo)

RSA

- Rivest, Shamir, Adleman
- Generate two primes: p, q
  - let $n = pq$
  - choose e, a small number, relatively prime to $(p-1)(q-1)$
  - choose d (< n) such that $ed \equiv 1 \bmod (p-1)(q-1)$
- RSA public-key is <e, n> (e is called the *public exponent*)
  RSA private-key is <d, n> (d is called the *private exponent*)
  - n is called the *public modulus*
- Then, $c = m^e \bmod n$ and $m = c^d \bmod n$
  - can also encrypt with d and decrypt with e
    i.e., $c = m^d \bmod n$ and $m = c^e \bmod n$
- Note: encryption is fast (because e is small) and decryption is slow

An Example

- Let $p = 5$, $q = 11$, $e = 3$ (recall that p & q are primes)
  - then $n = 55$ (recall that $n = pq$)
  - pick $e = 3$ (recall that e is relatively prime to $(p-1)(q-1)$)
  - $d = 27$, since $(3)(27) \bmod 40 = 1$
    (recall that $ed \equiv 1 \bmod (p-1)(q-1)$)
- If $m = 7$, then $c = 7^3 \bmod 55 = 343 \bmod 55 = 13$
- Then m should be $= 13^{27} \bmod 55$
- Computing $13^{27} \bmod 55$
  - $13^1 \bmod 55 = 13$, $13^2 \bmod 55 = 4$, $13^4 \bmod 55 = 16$,
    $13^8 \bmod 55 = 36$, $13^{16} \bmod 55 = 31$
  - $27 = 1+2+8+16$
  - $13^{27} \bmod 55 = (13)(4)(36)(31) \bmod 55 =$
    $(1872 \bmod 55)(31) \bmod 55 = 62 \bmod 55 = 7$ (check)

Calculating the Private Exponent

- *ed* ≡ *1 mod (p-1)(q-1)*
  - *d* is the *multiplicative inverse* of *e* modulo *(p-1)(q-1)*
  - multiplicative inverse of *e* is like the *reciprocal* of *e* since $e \cdot (1/e) = 1$
  - let *a* be an integer such that *a < n* has a multiplicative inverse modulo *n* only if *gcd(a,n)=1*
    - *a* has a multiplicative inverse modulo *n* if and only if *gcd(a,n)=1*
- How to compute multiplicative inverses?
  - use the *Extended Euclidean Algorithm*

Euclidean Algorithm

- Input: two non-negative integers a and b with $a \geq b$
  Output: $gcd(a,b)$
  1) while $b > 0$ do:
     1.1) $r \leftarrow a \bmod b,\ a \leftarrow b,\ b \leftarrow r$
  2) return (a)

- Ex: $a = 425$, $b = 153$, $gcd(a,b) = 17$

q	r	a	b
-	-	425	153
2	119	153	119
1	34	119	34
3	17	34	17
2	0	*17*	0

$425 = 2 \cdot 153 + 119$
$153 = 1 \cdot 119 + 34$
$119 = 3 \cdot 34 + 17$
$34 = 2 \cdot 17 + 0$

Extended Euclidean Algorithm [HAC 2.4]

- **Input:** two non-negative integers a_0 and b_0 with $a_0 \geq b_0$
  Output: $d = gcd(a_0,b_0)$ and integers x, y satisfying $a_0x + b_0y = d$
  1) if $b = 0$ then set $d \leftarrow a_0$, $x \leftarrow 1$, $y \leftarrow 0$, and return (d,x,y)
  2) set $a \leftarrow a_0$, $b \leftarrow b_0$, $x_2 \leftarrow 1$, $x_1 \leftarrow 0$, $y_2 \leftarrow 0$, $y_1 \leftarrow 1$
  3) while $b > 0$ do:
     3.1) $q \leftarrow \lfloor a/b \rfloor$, $r \leftarrow a - qb$, $x \leftarrow x_2 - qx_1$, $y \leftarrow y_2 - qy_1$
     3.2) $a \leftarrow b$, $b \leftarrow r$, $x_2 \leftarrow x_1$, $x_1 \leftarrow x$, $y_2 \leftarrow y_1$, $y_1 \leftarrow y$
  4) set $d \leftarrow a$, $x \leftarrow x_2$, $y \leftarrow y_2$, and return (d,x,y)
  - end of each iteration: $a_0x_2 + b_0y_2 = a$

- **Ex:** $a_0 = 425$, $b_0 = 153$, $gcd(a_0,b_0) = 17$, and $425 \cdot 4+153 \cdot (-11) = 17$

q	r	x	y	a	b	x_2	x_1	y_2	y_1
-	-	-	-	425	153	1	0	0	1
2	119	1	-2	153	119	0	1	1	-2
1	34	-1	3	119	34	1	-1	-2	3
3	17	4	-11	34	17	-1	4	3	-11
2	0	-8	25	*17*	0	*4*	-8	*-11*	25

The Table Method

- A simple way to implement the Extended Euclidean Algorithm
  - http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

```
rem[1] = a0;
rem[2] = b0;
x[1] = 0;
x[2] = 1;
y[1] = 1;
y[2] = 0;
for (i=3; rem[i] > 1; i++) {
  rem[i] = rem[i-2] % rem[i-1];
  quo[i] = rem[i-2] / rem[i-1];
  x[i] = -quo[i] * x[i-1] + x[i-2];
  y[i] = -quo[i] * y[i-1] + y[i-2]; /* optional */
}
inverse = x[i];
```

The Table Method (Cont...)

Ex: $a_0 = 425$, $b_0 = 153$, $gcd(a_0,b_0) = 17$, and $425 \cdot 4+153 \cdot (-11) = 17$

q	r	x	y	a	b	x_2	x_1	y_2	y_1
-	-	-	-	425	153	1	0	0	1
2	119	1	-2	153	119	0	1	1	-2
1	34	-1	3	119	34	1	-1	-2	3
3	17	4	-11	34	17	-1	4	3	-11
2	0	-8	25	*17*	0	*4*	-8	*-11*	25

- Table Method:

```
rem[i] = rem[i-2] - quo[i] * rem[i-1]
  x[i] =   x[i-2] - quo[i] *   x[i-1]
  y[i] =   y[i-2] - quo[i] *   y[i-1]
```

i	quo	rem	x	y
1	-	425	0	1
2	-	153	1	0

The Table Method (Cont...)

Ex: $a_0 = 425$, $b_0 = 153$, $gcd(a_0,b_0) = 17$, and $425 \cdot 4+153 \cdot (-11) = 17$

q	r	x	y	a	b	x_2	x_1	y_2	y_1
-	-	-	-	425	153	1	0	0	1
2	119	1	-2	153	119	0	1	1	-2
1	34	-1	3	119	34	1	-1	-2	3
3	17	4	-11	34	17	-1	4	3	-11
2	0	-8	25	*17*	0	*4*	-8	*-11*	25

- **Table Method:**

```
rem[i] = rem[i-2] - quo[i] * rem[i-1]
  x[i] =   x[i-2] - quo[i] *   x[i-1]
  y[i] =   y[i-2] - quo[i] *   y[i-1]
```

i	quo	rem	x	y
1	-	425	0	1
2	-	153	1	0
3	*2*	*119*		

The Table Method (Cont...)

- Ex: $a_0 = 425$, $b_0 = 153$, $gcd(a_0,b_0) = 17$, and $425 \cdot 4+153 \cdot (-11) = 17$

q	r	x	y	a	b	x_2	x_1	y_2	y_1
-	-	-	-	425	153	1	0	0	1
2	119	1	-2	153	119	0	1	1	-2
1	34	-1	3	119	34	1	-1	-2	3
3	17	4	-11	34	17	-1	4	3	-11
2	0	-8	25	*17*	0	*4*	-8	*-11*	25

- Table Method:

```
rem[i] = rem[i-2] - quo[i] * rem[i-1]
  x[i] =   x[i-2] - quo[i] *   x[i-1]
  y[i] =   y[i-2] - quo[i] *   y[i-1]
```

i	quo	rem	x	y
1	-	425	0	1
2	-	153	1	0
3	*2*	*119*	*-2*	

The Table Method (Cont...)

- Ex: $a_0 = 425$, $b_0 = 153$, $gcd(a_0,b_0) = 17$, and $425 \cdot 4+153 \cdot (-11) = 17$

q	r	x	y	a	b	x_2	x_1	y_2	y_1
-	-	-	-	425	153	1	0	0	1
2	119	1	-2	153	119	0	1	1	-2
1	34	-1	3	119	34	1	-1	-2	3
3	17	4	-11	34	17	-1	4	3	-11
2	0	-8	25	*17*	0	*4*	-8	*-11*	25

- Table Method:

```
rem[i] = rem[i-2] - quo[i] * rem[i-1]
  x[i] =   x[i-2] - quo[i] *   x[i-1]
  y[i] =   y[i-2] - quo[i] *   y[i-1]
```

i	quo	rem	x	y
1	-	425	0	1
2	-	153	1	0
3	*2*	*119*	*-2*	*1*

The Table Method (Cont...)

Ex: $a_0 = 425$, $b_0 = 153$, $gcd(a_0,b_0) = 17$, and $425 \cdot 4+153 \cdot (-11) = 17$

q	r	x	y	a	b	x_2	x_1	y_2	y_1
-	-	-	-	425	153	1	0	0	1
2	119	1	-2	153	119	0	1	1	-2
1	34	-1	3	119	34	1	-1	-2	3
3	17	4	-11	34	17	-1	4	3	-11
2	0	-8	25	*17*	0	*4*	-8	*-11*	25

- **Table Method:**

```
rem[i] = rem[i-2] - quo[i] * rem[i-1]
  x[i] =   x[i-2] - quo[i] *   x[i-1]
  y[i] =   y[i-2] - quo[i] *   y[i-1]
```

i	quo	rem	x	y
1	-	425	0	1
2	-	153	1	0
3	2	119	-2	1
4	*1*	*34*	*3*	*-1*

The Table Method (Cont...)

- **Ex:** $a_0 = 425$, $b_0 = 153$, $gcd(a_0,b_0) = 17$, and $425 \cdot 4+153 \cdot (-11) = 17$

q	r	x	y	a	b	x_2	x_1	y_2	y_1
-	-	-	-	425	153	1	0	0	1
2	119	1	-2	153	119	0	1	1	-2
1	34	-1	3	119	34	1	-1	-2	3
3	17	4	-11	34	17	-1	4	3	-11
2	0	-8	25	*17*	0	*4*	-8	*-11*	25

- **Table Method:**

```
rem[i] = rem[i-2] - quo[i] * rem[i-1]
  x[i] =   x[i-2] - quo[i] *   x[i-1]
  y[i] =   y[i-2] - quo[i] *   y[i-1]
```

i	quo	rem	x	y
1	-	425	0	1
2	-	153	1	0
3	2	119	-2	1
4	1	34	3	-1
5	*3*	*17*	*-11*	*4*

The Table Method (Cont...)

Ex: $a_0 = 425$, $b_0 = 153$, $gcd(a_0,b_0) = 17$, and $425 \cdot 4+153 \cdot (-11) = 17$

q	r	x	y	a	b	x_2	x_1	y_2	y_1
-	-	-	-	425	153	1	0	0	1
2	119	1	-2	153	119	0	1	1	-2
1	34	-1	3	119	34	1	-1	-2	3
3	17	4	-11	34	17	-1	4	3	-11
2	0	-8	25	*17*	0	*4*	-8	*-11*	25

- **Table Method:**

```
rem[i] = rem[i-2] - quo[i] * rem[i-1]
  x[i] =   x[i-2] - quo[i] *   x[i-1]
  y[i] =   y[i-2] - quo[i] *   y[i-1]
```

i	quo	rem	x	y
1	-	425	0	1
2	-	153	1	0
3	2	119	-2	1
4	1	34	3	-1
5	3	17	-11	4
6	*2*	*0*	*25*	*-9*

Multiplicative Inverse Example

- What is the multipliactive inverse of *3* modulo *40*?
  - let *a=40* and *b=3*, formulate *ax + by = d*
  - since *3* and *40* are relatively prime, *d = 1*
  - after solving *ax + by = 1 (mod a)*
    - *x* is really irrelavent since *a = 0 (mod a)*
    - *y* is the multiplicative of *b (mod a)*
  - use the Table Method

quo	rem	x
-	40	0
-	3	1
13	1	-13

- the multipliactive inverse of *3 (mod 40)* is $-13 \equiv 27 \pmod{40}$
  - $3 \cdot 27 = 1 \pmod{40}$

Security of RSA

- Avoid known pitfalls
  - *p* and *q* cannot be small
  - always add *salt* (i.e., nonce) to a message
  - introduce *structural constraints* on plaintext messages, e.g., repeat bits in original input message before encryption
    - after decryption, check constraints
    - if constraints not met, do not send back decrypted data
- Breaking RSA is believed to be equivalent to solving the unique factorization problem
  - tools for unique factorization of large products of primes
    - elliptic curve factoring algorithm
    - quadratic sieve or general number field sieve
      - although subexponential, if p and q are large enough, these methods are not considered "computationally feasible" to factor

Other Public Key Cryptosystems

- Diffie-Hellman
  - first public key cryptosystem
  - Diffie and Hellman were often cited as creators of public key cryptosystem
  - security based on the discrete logarithm problem
    - for any integers g, p, n: find k such that $g^k \bmod p = n$
- Parameters of the Diffie-Hellman cryptosystem
  - prime p (the modulus) and g (the generator)
    - $1 \leq g \leq p-2$ and for $i=0,1,2,3,\ldots p-2$, g^i generates all values between 1 through $p-1$
  - every entity picks a private key k
    - its public key $K = g^k \bmod p$
- Diffie-Hellman is not strickly a public key cryptosystem
  - basically a key exchange system

Diffie-Hellman Example

- Diffie-Hellman example
  - Alice has private key x and public key $X = g^{x} \bmod p$
  - Bob has private key y and public key $Y = g^{y} \bmod p$
  - Alice wants to communicate with Bob
    - gets Bob's public key Y and computes $Z = Y^{x} \bmod p$
    - derive a key z from Z using a pre-defined public algorithm (e.g., $m \cdot Y^{x} \bmod p$)
    - encrypts a message with z
  - when Bob gets an encrypted message from Alice
    - gets Alice's public key X and computes $Z' = X^{y} \bmod p$
    - $Z' = Z = g^{xy} \bmod p$
    - derive a symmetric key z' from Z' using a pre-defined public algorithm
    - decrypts Alice's message with $z' = z$

Diffie-Hellman Numeric Example

- Diffie-Hellman numeric example
  - $p = 53$, $g = 17$ (which can be shown to be a generator)
  - $x = 5$, $X = g^{x} \bmod p = 17^{5} \bmod 53 = 40$
  - $y = 7$, $Y = g^{y} \bmod p = 17^{7} \bmod 53 = 6$
  - $X^{y} \bmod p = 40^{7} \bmod 53 = 38$
  - $Y^{x} \bmod p = 6^{5} \bmod 53 = 38$

Other Public Key Cryptosystems (Cont...)

- ElGamal (signature, encryption)
  - ex: (encryption and decryption)
    - choose a prime p, and two random numbers g, $x < p$
    - public key is g, p, and $X = g^{x} \bmod p$
    - private key is x; to obtain from public key requires extracting discrete log
    - to encrypt message m for an entity with private key y and public key $Y = g^{y} \bmod p$, compute $c = m \cdot g^{xy} \bmod p$
      - recall that $g^{xy} \bmod p = X^{y} \bmod p = Y^{x} \bmod p$
    - to decrypt message m, first compute $g^{-xy} \bmod p$
      - $g^{-k} \cdot g^{k} \equiv 1 \bmod p$
    - then compute $c \cdot g^{-xy} \bmod p = m \cdot g^{xy} \cdot g^{-xy} \bmod p = m$
  - mostly used for signatures

Other Public Key Cryptosystems (Cont...)

- **Elliptic curve cryptosystems**
  - $y^2 = x^3 + ax^2 + bx + c$
  - **elliptic curves were featured in Fermat's Last Theorem proof**
    - **Fermat's Last Theorem:**
      - **cannot find x, y, z, such that $x^n + y^n = z^n$ if $n > 2$**
  - **elliptic curves (e.g., $mod\ n$ or arithmatic in $GF(2^8)$) used to implement existing public-key systems (e.g., RSA, ElGamal)**
    - **allow for shorter keys and greater efficiency**
    - **application in battery operated devices**

Combining Public-key and Secret-key Algorithms

- Public-key algorithms are orders of magnitude slower than secret-key algorithms
  - not practical to encrypt a large document using public-key cryptography
- Bulk data encryption
  - combine public-key (e.g., RSA) and secret-key (e.g., 3DES)
    - generate session key (random)
    - encrypt session key with receiver's RSA public key
      - session key must be smaller than the public modulus
    - 3DES encrypt data with session key
    - receiver decrypts with RSA private key to get session key, then decrypts data with session key
  - basically a method of ***key exchange***

Another Way to Exchange Keys

- ***Diffie-Hellman key exchange***
  - choose large prime p, and generator g
    - for any n in $(1, p-1)$, there exists a k such that $g^k \equiv n \bmod p$
  - Alice, Bob select secret values x, y, respectively
  - Alice sends $X = g^x \bmod p$
  - Bob sends $Y = g^y \bmod p$
  - both compute $g^{xy} \bmod p$, a shared secret
    - can be used as keying material

- Diffie-Hellman key exchange is vulnerable to the *man-in-the-middle* attack
  - Eve selects z and computes $Z = g^z \bmod p$
  - Eve establishes a channel with Alice using $g^{xz} \bmod p$
  - Eve establishes a channel with Bob using $g^{yz} \bmod p$
  - Alice and Bob cannot know that Eve is decrypting and re-encrypting messages