CNT4406/5412 Network Security

Public Key Algorithms

Zhi Wang

Florida State University

Fall 2014

Introduction

- Each principal has a pair of public and secret numbers (e, d)
  - ➠ public key is announced to the public
  - ➠ private key is kept secret

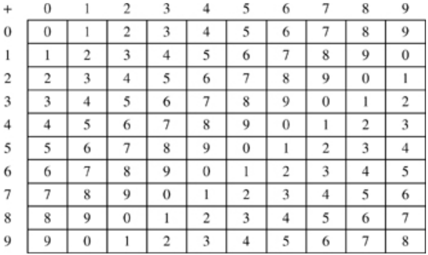
Introduction

- Each principal has a pair of public and secret numbers (e, d)
  - ➠ public key is announced to the public
  - ➠ private key is kept secret
- Public key algorithms are different in design
  - ➠ Diffie-Hellman allows establishment of a shared secret

	encryption	signature	key exchange
RSA	y	y	y
Diffie-Hellman	n	n	y
DSA	n	y	n

Modular Addition

- Modular addition is reversible for all numbers $a < n$
  ➠ Caesar cipher uses modular addition



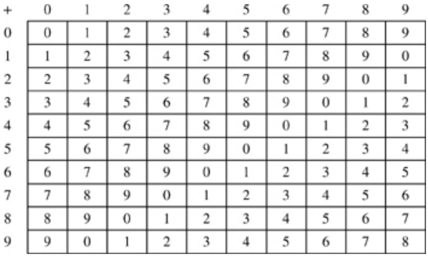
+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

Modular Addition

- Modular addition is reversible for all numbers $a < n$

  ➠ Caesar cipher uses modular addition
- Additive inverse of a is $-a \mod n$

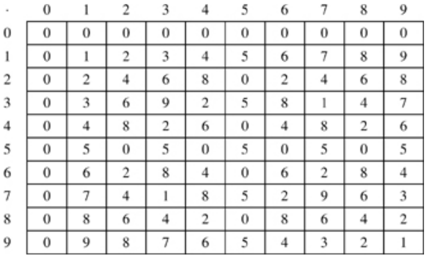
  ➠ e.g., 7 is 3 's additive inverse in $\mod 10$



+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

Modular Multiplication

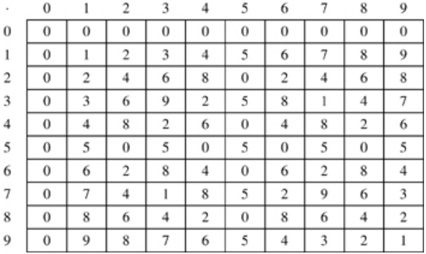
- It is reversible for numbers relatively-prime to and less than *n*
  ➠ multiplicative inverse can be calculated by Euclid's algorithm



·	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	5	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

Modular Multiplication

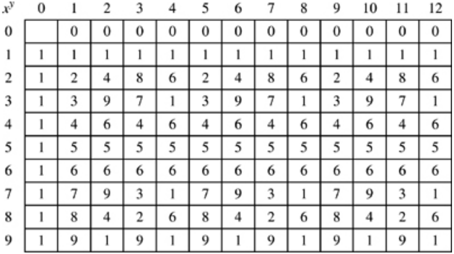
- It is reversible for numbers relatively-prime to and less than n
  ➠ multiplicative inverse can be calculated by Euclid's algorithm
- Totient function $\phi(n)$: how many numbers less than and relatively-prime to n
  ➠ if n is prime, $\phi(n) = n - 1$
  ➠ if $n = pq$ and p, q are prime, $\phi(n) = (p-1)(q-1) = \phi(p)\phi(q)$



·	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	5	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

Modular Exponentiation

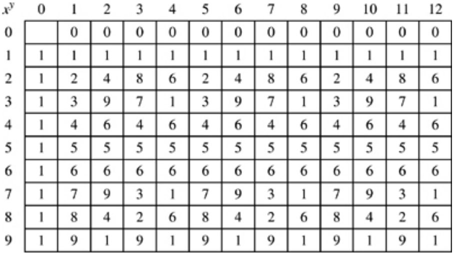
- Euler's theorem: $x^y \mod n = x^{y \mod \phi(n)} \mod n$ if n is prime or a product of two distinct primes
  - e.g., $n = 10, \phi(n) = 4, x^1 = x^5 = x^9 \mod 10$



x^y	0	1	2	3	4	5	6	7	8	9	10	11	12
0		0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	6	2	4	8	6	2	4	8	6
3	1	3	9	7	1	3	9	7	1	3	9	7	1
4	1	4	6	4	6	4	6	4	6	4	6	4	6
5	1	5	5	5	5	5	5	5	5	5	5	5	5
6	1	6	6	6	6	6	6	6	6	6	6	6	6
7	1	7	9	3	1	7	9	3	1	7	9	3	1
8	1	8	4	2	6	8	4	2	6	8	4	2	6
9	1	9	1	9	1	9	1	9	1	9	1	9	1

Modular Exponentiation

- Euler's theorem: $x^y \mod n = x^{y \mod \phi(n)} \mod n$ if n is prime or a product of two distinct primes
  ➠ e.g., $n = 10, \phi(n) = 4, x^1 = x^5 = x^9 \mod 10$
- Exponentiative inverse: $yz = 1 \mod \phi(n) \rightsquigarrow (x^y)^z = x^{yz} = x$
  ➠ $yz = 1 \mod \phi(n)$: z is y 's multiplicative inverse $\mod \phi(n)$



x^y	0	1	2	3	4	5	6	7	8	9	10	11	12
0		0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	6	2	4	8	6	2	4	8	6
3	1	3	9	7	1	3	9	7	1	3	9	7	1
4	1	4	6	4	6	4	6	4	6	4	6	4	6
5	1	5	5	5	5	5	5	5	5	5	5	5	5
6	1	6	6	6	6	6	6	6	6	6	6	6	6
7	1	7	9	3	1	7	9	3	1	7	9	3	1
8	1	8	4	2	6	8	4	2	6	8	4	2	6
9	1	9	1	9	1	9	1	9	1	9	1	9	1

RSA (Rivest, Shamir, Adleman)

- RSA provides both encryption and digital signature

RSA (Rivest, Shamir, Adleman)

- RSA provides both encryption and digital signature
- Variable key length (512 bits or greater) and variable block size
  - ➠ plaintext block must be shorter than the key size
  - ➠ ciphertext block has the same length as the key size

RSA (Rivest, Shamir, Adleman)

- RSA provides both encryption and digital signature
- Variable key length (512 bits or greater) and variable block size
  - ➠ plaintext block must be shorter than the key size
  - ➠ ciphertext block has the same length as the key size
- Basis: factorization of large numbers is hard

RSA (Rivest, Shamir, Adleman)

- RSA provides both encryption and digital signature
- Variable key length (512 bits or greater) and variable block size
  - ➠ plaintext block must be shorter than the key size
  - ➠ ciphertext block has the same length as the key size
- Basis: factorization of large numbers is hard
- RSA is slow, mostly used to encrypt/sign short messages
  - ➠ e.g., shared session keys or message digests

Key Generation

- Choose two large primes, p and q (about 256 bits each)
  ➠ never reveal p and q

Key Generation

- Choose two large primes, p and q (about 256 bits each)
  ➠ never reveal p and q
- Let $n = p \times q$ ($\phi(n) = ???$)
  ➠ factoring n (512 bit) into p and q is hard

Key Generation

- Choose two large primes, p and q (about 256 bits each)
  ➠ never reveal p and q
- Let $n = p \times q$ ($\phi(n) = ???$)
  ➠ factoring n (512 bit) into p and q is hard
- Public key is $< e, n >$, e relatively prime to $\phi(n)$,
  private key is $< d, n >$, $ed = 1 \mod \phi(n)$

Operations

- Public key $< e, n >$, private key $< d, n >$

Operations

- Public key $< e, n >$, private key $< d, n >$
- Encryption of $m < n$: $c = m^e \mod n$, decryption: $m = c^d \mod n$

Operations

- Public key $< e, n >$, private key $< d, n >$
- Encryption of $m < n$: $c = m^e \mod n$,
  decryption: $m = c^d \mod n$
- Signing $m < n$: $s = m^d \mod n$
  verification: $m = s^e \mod n$

Operations

- Public key $< e, n >$, private key $< d, n >$
- Encryption of $m < n$: $c = m^e \mod n$,
  decryption: $m = c^d \mod n$
- Signing $m < n$: $s = m^d \mod n$
  verification: $m = s^e \mod n$
- Who are the principles of these operations???

Example

$$p = 23, q = 11 \leadsto n = pq = 253, \phi(n) = (p-1)(q-1) = 220$$

Example

$p = 23, q = 11 \rightsquigarrow n = pq = 253, \phi(n) = (p-1)(q-1) = 220$

$e = 39$ (relatively prime to 220) $\rightsquigarrow$ public key: $< 39, 253 >$

Example

$p = 23, q = 11 \rightsquigarrow n = pq = 253, \phi(n) = (p-1)(q-1) = 220$

$e = 39$ (relatively prime to 220) $\rightsquigarrow$ public key: $< 39, 253 >$

$d = e^{-1} \mod 220 = 79 \rightsquigarrow$ private key: $< 79, 253 >$

Example

$p = 23, q = 11 \rightsquigarrow n = pq = 253, \phi(n) = (p-1)(q-1) = 220$

$e = 39$ (relatively prime to 220) $\rightsquigarrow$ public key: $< 39, 253 >$

$d = e^{-1} \mod 220 = 79 \rightsquigarrow$ private key: $< 79, 253 >$

➠ $m = 80$

- encryption: $c = m^e \mod n = 80^{39} \mod 253 = 37$
- decryption: $m = c^d \mod n = 37^{79} \mod 253 = 80$

Example

$p = 23, q = 11 \rightsquigarrow n = pq = 253, \phi(n) = (p-1)(q-1) = 220$

$e = 39$ (relatively prime to 220) $\rightsquigarrow$ public key: $< 39, 253 >$

$d = e^{-1} \mod 220 = 79 \rightsquigarrow$ private key: $< 79, 253 >$

➠ $m = 80$

- encryption: $c = m^e \mod n = 80^{39} \mod 253 = 37$
- decryption: $m = c^d \mod n = 37^{79} \mod 253 = 80$
- signature: $s = m^d \mod n = 80^{79} \mod 253 = 224$
- verification: $m = s^e \mod n = 224^{39} \mod 253 = 80$

Why RSA Works

$n = pq, \phi(n) = (p-1)(q-1)$, and $de = 1 \mod \phi(n)$

$\rightsquigarrow x^{de} = x \mod n$ (Euler's theorem, $x \in Z_n$)

Why RSA Works

$n = pq, \phi(n) = (p-1)(q-1)$, and $de = 1 \mod \phi(n)$

$\rightsquigarrow x^{de} = x \mod n$ (Euler's theorem, $x \in Z_n$)

➠ encryption: x^e, decryption: $(x^e)^d = x^{ed} = x$

➠ signature and verification are the reverse

Why RSA is Secure

- Public key $< e, n >$ is a public information
- Factoring large number n into $p \times q$ is difficult
  ➠ if factored $\leadsto \phi(n) = (p-1)(q-1)$
  $\leadsto d = e^{-1} \mod \phi(n) \leadsto < d, n >$

Why RSA is Secure

- Public key $< e, n >$ is a public information
- Factoring large number n into $p \times q$ is difficult
  - ➠ if factored $\rightsquigarrow \phi(n) = (p-1)(q-1)$ $\rightsquigarrow d = e^{-1} \mod \phi(n) \rightsquigarrow < d, n >$
  - ➠ 1024 bits are consider secure for now, 2048 bits are better

Implementation

- Basic operation: exponentiating with big numbers
- Generating RSA keys:
  ➠ finding big primes p and q, and selecting d and e

Exponentiating

To compute $a^x \mod t$, use repeated squaring and do modular reduction at each step

Exponentiating

To compute $a^x \mod t$, use repeated squaring and do modular reduction at each step

Example

$a = 123, x = 54 = 110110_2, t = 678, a^{54} = (((((a)^2a)^2)^2a)^2a)^2$

1_2	123

Exponentiating

To compute $a^x \mod t$, use repeated squaring and do modular reduction at each step

Example

$a = 123, x = 54 = 110110_2, t = 678, a^{54} = (((((a)^2a)^2)^2a)^2a)^2$

1_2		123
10_2	$\Lsh$	$123^2 = 123 \times 123 = 15129 = 213 \mod 678$
11_2	$+1$	$123^3 = 213 \times 123 = 26199 = 435 \mod 678$

Exponentiating

To compute $a^x \mod t$, use repeated squaring and do modular reduction at each step

Example

$a = 123, x = 54 = 110110_2, t = 678, a^{54} = (((((a)^2 a)^2)^2 a)^2 a)^2$

$$\begin{array}{lcl}
1_2 & & 123 \\
10_2 & \Lsh & 123^2 = 123 \times 123 = 15129 = 213 \mod 678 \\
11_2 & +1 & 123^3 = 213 \times 123 = 26199 = 435 \mod 678 \\
110_2 & \Lsh & 123^6 = 435 \times 435 = 189225 = 63 \mod 678
\end{array}$$

Exponentiating

To compute $a^x \mod t$, use repeated squaring and do modular reduction at each step

Example

$$a = 123, x = 54 = 110110_2, t = 678, a^{54} = (((((a)^2 a)^2)^2 a)^2 a)^2$$

1_2		123
10_2	$\Lsh$	$123^2 = 123 \times 123 = 15129 = 213 \mod 678$
11_2	$+1$	$123^3 = 213 \times 123 = 26199 = 435 \mod 678$
110_2	$\Lsh$	$123^6 = 435 \times 435 = 189225 = 63 \mod 678$
1100_2	$\Lsh$	$123^{12} = 63 \times 63 = 3969 = 579 \mod 678$
1101_2	$+1$	$123^{13} = 579 \times 123 = 71217 = 27 \mod 678$

Exponentiating

To compute $a^x \mod t$, use repeated squaring and do modular reduction at each step

Example

$a = 123, x = 54 = 110110_2, t = 678, a^{54} = (((((a)^2 a)^2)^2 a)^2 a)^2$

$$\begin{array}{lcl}
1_2 & & 123 \\
10_2 & \Lsh & 123^2 = 123 \times 123 = 15129 = 213 \mod 678 \\
11_2 & +1 & 123^3 = 213 \times 123 = 26199 = 435 \mod 678 \\
110_2 & \Lsh & 123^6 = 435 \times 435 = 189225 = 63 \mod 678 \\
1100_2 & \Lsh & 123^{12} = 63 \times 63 = 3969 = 579 \mod 678 \\
1101_2 & +1 & 123^{13} = 579 \times 123 = 71217 = 27 \mod 678 \\
11010_2 & \Lsh & 123^{26} = 27 \times 27 = 729 = 51 \mod 678 \\
11011_2 & +1 & 123^{27} = 51 \times 123 = 6273 = 171 \mod 678
\end{array}$$

Exponentiating

To compute $a^x \mod t$, use repeated squaring and do modular reduction at each step

Example

$a = 123, x = 54 = 110110_2, t = 678, a^{54} = (((((a)^2a)^2)^2a)^2a)^2$

1_2		123
10_2	$\Lsh$	$123^2 = 123 \times 123 = 15129 = 213 \mod 678$
11_2	$+1$	$123^3 = 213 \times 123 = 26199 = 435 \mod 678$
110_2	$\Lsh$	$123^6 = 435 \times 435 = 189225 = 63 \mod 678$
1100_2	$\Lsh$	$123^{12} = 63 \times 63 = 3969 = 579 \mod 678$
1101_2	$+1$	$123^{13} = 579 \times 123 = 71217 = 27 \mod 678$
11010_2	$\Lsh$	$123^{26} = 27 \times 27 = 729 = 51 \mod 678$
11011_2	$+1$	$123^{27} = 51 \times 123 = 6273 = 171 \mod 678$
110110_2	$\Lsh$	$123^{54} = 171 \times 171 = 29241 = 87 \mod 678$

Exponentiating

Pseudo code to compute $a^x \mod t$, assuming x has k bits

```
$r = a$
for $i = k - 1$ to 1:
    $r = r \times r \mod t$
    if $x_i == 1$:
        $r = r \times a \mod t$
return $r$
```

Exponentiating: Timing Attacks

- Timing attack: to recover the private key from the running time of the decryption algorithm ($m = c^d \mod n$)

---

[*] http://www.cs.sjsu.edu/faculty/stamp/students/article.html

Exponentiating: Timing Attacks

- Timing attack: to recover the private key from the running time of the decryption algorithm ($m = c^d \mod n$)
  ➠ the attack proceeds bit by bit (assuming he knows c and m):

---

* http://www.cs.sjsu.edu/faculty/stamp/students/article.html

Exponentiating: Timing Attacks

- Timing attack: to recover the private key from the running time of the decryption algorithm ($m = c^d \mod n$)
  ➠ the attack proceeds bit by bit (assuming he knows c and m):
  ➠ $r = r \times a \mod t$ is only executed if $d_i = 1$
  for some c and m combination, this step is extremely slow

---

[*] http://www.cs.sjsu.edu/faculty/stamp/students/article.html

Exponentiating: Timing Attacks

- Timing attack: to recover the private key from the running time of the decryption algorithm ($m = c^d \mod n$)
  - ➠ the attack proceeds bit by bit (assuming he knows c and m):
  - ➠ $r = r \times a \mod t$ is only executed if $d_i = 1$

    for some c and m combination, this step is extremely slow
  - ➠ attackers can determine bits of d by comparing time*

---

* http://www.cs.sjsu.edu/faculty/stamp/students/article.html

Exponentiating: Timing Attacks

- Timing attack: to recover the private key from the running time of the decryption algorithm ($m = c^d \mod n$)
  - ➠ the attack proceeds bit by bit (assuming he knows c and m):
  - ➠ $r = r \times a \mod t$ is only executed if $d_i = 1$

    for some c and m combination, this step is extremely slow
  - ➠ attackers can determine bits of d by comparing time*
- To mitigate, use *blinding*: multiply the ciphertext by a random number before decryption

---

* http://www.cs.sjsu.edu/faculty/stamp/students/article.html

Finding Big Primes

- Infinite number of primes, but thin out when getting bigger
  - ➠ probability of a random number n being prime is $\frac{1}{\ln n}$
  - ➠ e.g., 1 in 23 for a ten-digit number, 1 in 230 for hundred-digits

Finding Big Primes

- Infinite number of primes, but thin out when getting bigger
  - ➠ probability of a random number n being prime is $\frac{1}{\ln n}$
  - ➠ e.g., 1 in 23 for a ten-digit number, 1 in 230 for hundred-digits
- Method: choose a random number then test if it is a prime

Finding Big Primes

Theorem

Fermat's theorem: if p *is prime and* $0 < a < p$, $a^{p-1} = 1 \mod p$

Finding Big Primes

Theorem

Fermat's theorem: if p is prime and $0 < a < p$, $a^{p-1} = 1 \mod p$

- It is a specialization of Euler's theorem: a relatively prime to $n \rightsquigarrow a^{\phi(n)} = 1 \mod n$

Finding Big Primes

Theorem

Fermat's theorem: if p is prime and $0 < a < p$, $a^{p-1} = 1 \mod p$

- It is a specialization of Euler's theorem: a relatively prime to $n \rightsquigarrow a^{\phi(n)} = 1 \mod n$
- Probability of p is not prime but $a^{p-1} = 1 \mod p$ is $\frac{1}{10^{13}}$
  ➠ test multiple a to increase confidence
  ➠ Carmichael numbers are special cases

Finding Big Primes

Theorem

Fermat's theorem: if p is prime and $0 < a < p$, $a^{p-1} = 1 \mod p$

- It is a specialization of Euler's theorem: a relatively prime to $n \rightsquigarrow a^{\phi(n)} = 1 \mod n$
- Probability of p is not prime but $a^{p-1} = 1 \mod p$ is $\frac{1}{10^{13}}$
  - ➠ test multiple a to increase confidence
  - ➠ Carmichael numbers are special cases

Miller and Rabin test

If n is a prime, the only $\mod n$ square roots of 1 are 1 and -1, but many square roots if n is not a power of a prime (exercise: why??)

Finding d and e

- Choose a number that is relatively-prime to $\phi(n)$ as e

  ➠ e is public and can be a small number such as 3 or 65537

Finding *d* and *e*

- Choose a number that is relatively-prime to $\phi(n)$ as *e*
  ➠ *e* is public and can be a small number such as 3 or 65537
- Compute *d* using Euclid's algorithm
  ➠ *d* must be big to avoid being searchable

Issues with $e = 3$

- Messages less than $n^{\frac{1}{3}}$ will be encrypted as m^3
  ➠ take cube root of the ciphertext to decrypt

Issues with $e = 3$

- Messages less than $n^{\frac{1}{3}}$ will be encrypted as m^3
  ➠ take cube root of the ciphertext to decrypt
- Same message enctyped and sent to ≥ 3 recipients with $e = 3$
  ➠ plaintext can be revealed by using Chinese Remainder Theorem

Issues with $e = 3$

- Messages less than $n^{\frac{1}{3}}$ will be encrypted as m^3
  ➠ take cube root of the ciphertext to decrypt
- Same message enctyped and sent to ≥ 3 recipients with $e = 3$
  ➠ plaintext can be revealed by using Chinese Remainder Theorem
  ➠ to address it, use random/individualized padding

RSA Threats with $e = 3$

- Cube root problem: forge signature on any messages
  ➠ assume message are padded on the right with random numbers

RSA Threats with $e = 3$

- Cube root problem: forge signature on any messages
  ➠ assume message are padded on the right with random numbers
  ➠ to forge signature, digest the message to h, pad it on the right with zeros, then set the signature to $r = h^{\frac{1}{3}}$

RSA Threats with $e = 3$

- Cube root problem: forge signature on any messages
  ➠ assume message are padded on the right with random numbers
  ➠ to forge signature, digest the message to h, pad it on the right with zeros, then set the signature to $r = h^{\frac{1}{3}}$
  ➠ signature is forged because $r^3 = h$ (padded with random numbers)

RSA Threats: Smooth Numbers

- Smooth number is the product of reasonably small primes
- Smooth number threat
  ➠ RSA signs a message by $m^d \mod n$

RSA Threats: Smooth Numbers

- Smooth number is the product of reasonably small primes
- Smooth number threat
  ➠ RSA signs a message by $m^d \mod n$
  ➠ with signature on m_1 and m_2, the attacker can forge signature on: $m_1 \times m_2, \frac{m_1}{m_2}, m_1^j, m_1^j \times m_2^k, ...$

RSA Threats: Smooth Numbers

- Smooth number is the product of reasonably small primes
- Smooth number threat
  - ➠ RSA signs a message by $m^d \mod n$
  - ➠ with signature on m_1 and m_2, the attacker can forge signature on: $m_1 \times m_2, \frac{m_1}{m_2}, m_1^j, m_1^j \times m_2^k, \ldots$
  - ➠ "small" primes provide flexible building blocks

    attackers can forge signatures on any product from his collection

Public Key Cryptography Standards (PKCS)

- Correct use of RSA could be tricky
- PKCS is the operational standards to avoid pitfalls

Public Key Cryptography Standards (PKCS)

- Correct use of RSA could be tricky
- PKCS is the operational standards to avoid pitfalls

  ➠ encrypting guessable message

Public Key Cryptography Standards (PKCS)

- Correct use of RSA could be tricky
- PKCS is the operational standards to avoid pitfalls
  - ➠ encrypting guessable message
  - ➠ signing smooth number

Public Key Cryptography Standards (PKCS)

- Correct use of RSA could be tricky
- PKCS is the operational standards to avoid pitfalls
  - ➠ encrypting guessable message
  - ➠ signing smooth number
  - ➠ multiple recipients of a message when $e = 3$

Public Key Cryptography Standards (PKCS)

- Correct use of RSA could be tricky
- PKCS is the operational standards to avoid pitfalls
  - ➠ encrypting guessable message
  - ➠ signing smooth number
  - ➠ multiple recipients of a message when $e = 3$
  - ➠ encrypting messages $\leq n^{\frac{1}{3}}$ when $e = 3$

Public Key Cryptography Standards (PKCS)

- Correct use of RSA could be tricky
- PKCS is the operational standards to avoid pitfalls
  - ➠ encrypting guessable message
  - ➠ signing smooth number
  - ➠ multiple recipients of a message when $e = 3$
  - ➠ encrypting messages $\leq n^{\frac{1}{3}}$ when $e = 3$
  - ➠ signing messages with random padding on the right when $e = 3$

PKCS #1 Encryption

- How does PKCS #1 address the following pitfalls?
  - encrypting guessable message
  - multiple recipients of a message when $e = 3$
  - encrypting messages $\leq n^{\frac{1}{3}}$ when $e = 3$

0	2	at least 8 random non-zero octets	0	data

PKCS #1 Signature

- How does PKCS #1 address the following pitfalls?
  - ➠ signing smooth number
  - ➠ signing messages with random padding on the right when $e = 3$

0	1	at least 8 octets of ff_{16}	0	ASN.1-encoded digest type/value

PKCS #1 Signature

- How does PKCS #1 address the following pitfalls?
  - signing smooth number
  - signing messages with random padding on the right when $e = 3$
- Why 8 octets of ff_{16} instead of random bytes?

0	1	at least 8 octets of ff_{16}	0	ASN.1-encoded digest type/value

Diffie-Hellman

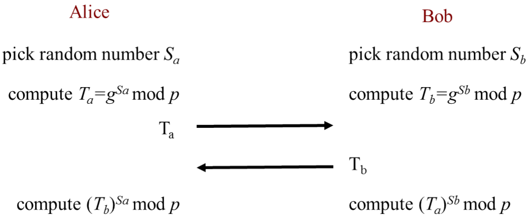
- Diffie-Hellman is designed to negotiate a shared secret key using only public communication
  - ➠ i.e. not suitable for public key encryption

Diffie-Hellman

- Diffie-Hellman is designed to negotiate a shared secret key using only public communication
  - ➠ i.e. not suitable for public key encryption
- Diffie-Hellman does not provide authentication of the principles
  - ➠ you could negotiate a key with a complete stranger

Diffie-Hellman Protocol

- Publicly publish two numbers, p and g
  - ➠ p is a large prime (about 512 bits), and $g < p$



Alice

pick random number S_a

compute $T_a = g^{Sa} \bmod p$

$T_a \longrightarrow$

Bob

pick random number S_b

compute $T_b = g^{Sb} \bmod p$

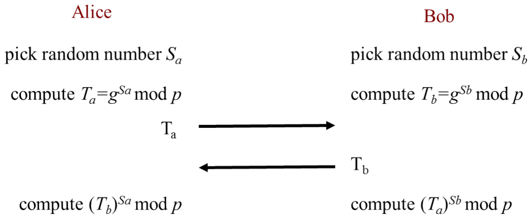
$\longleftarrow T_b$

Alice: compute $(T_b)^{Sa} \bmod p$

Bob: compute $(T_a)^{Sb} \bmod p$

Diffie-Hellman Protocol

- Publicly publish two numbers, p and g
  - ➠ p is a large prime (about 512 bits), and $g < p$
- Alice and Bob exchange two numbers T_a and T_b
- They agree upon $g^{S_a S_b} \mod p$ after DH exchange



Alice

pick random number S_a

compute $T_a = g^{Sa} \bmod p$

T_a

compute $(T_b)^{Sa} \bmod p$

Bob

pick random number S_b

compute $T_b = g^{Sb} \bmod p$

T_b

compute $(T_a)^{Sb} \bmod p$

Example

- Let $p = 353, g = 3, S_a = 97, S_b = 233$

Example

- Let $p = 353, g = 3, S_a = 97, S_b = 233$
- Alice computes $T_a = g^{S_a} \mod p = 3^{97} \mod 353 = 40$
  Bob computes $T_b = g^{S_b} \mod p = 3^{233} \mod 353 = 248$

Example

- Let $p = 353, g = 3, S_a = 97, S_b = 233$
- Alice computes $T_a = g^{S_a} \mod p = 3^{97} \mod 353 = 40$
  Bob computes $T_b = g^{S_b} \mod p = 3^{233} \mod 353 = 248$
- Alice and Bob exchanges T_a and T_b

Example

- Let $p = 353, g = 3, S_a = 97, S_b = 233$
- Alice computes $T_a = g^{S_a} \mod p = 3^{97} \mod 353 = 40$
  Bob computes $T_b = g^{S_b} \mod p = 3^{233} \mod 353 = 248$
- Alice and Bob exchanges T_a and T_b
- Alice computes $K = T_b^{S_a} \mod p = 248^{97} \mod 353 = 160$
  Bob computes $K = T_a^{S_b} \mod p = 40^{233} \mod 353 = 160$

Diffie-Hellman Offline Mode

The same as Diffie-Hellman, but Bob pre-selects his S_b and publishes T_b

- Bob publishes $< p_b, g_b, T_b >$

Diffie-Hellman Offline Mode

The same as Diffie-Hellman, but Bob pre-selects his S_b and publishes T_b

- Bob publishes $< p_b, g_b, T_b >$
- Alice picks a random S_a, and computes $K_{ab} = T_b^{S_a} \mod p_b$
- Alice encrypts the message with K_{ab}
- Alice sends ciphertext and $T_a = g_b^{S_a} \mod p_b$ to Bob

Diffie-Hellman Offline Mode

The same as Diffie-Hellman, but Bob pre-selects his S_b and publishes T_b

- Bob publishes $< p_b, g_b, T_b >$
- Alice picks a random S_a, and computes $K_{ab} = T_b^{S_a} \mod p_b$
- Alice encrypts the message with K_{ab}
- Alice sends ciphertext and $T_a = g_b^{S_a} \mod p_b$ to Bob
- Bob computes $K_{ab} = T_a^{S_b}$, and decrypt the message with it

Why DH is Secure

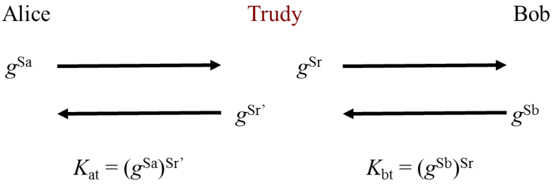
- Discrete logarithms problem is difficult
  - ➠ given $g^S \mod p$, g, and p, it is computationally difficult to get S
  - ➠ no guarantee, but remember Fundamental Tenet of Cryptograph?

Why DH is Secure

- Discrete logarithms problem is difficult
  - ➠ given $g^S \mod p$, g, and p, it is computationally difficult to get S
  - ➠ no guarantee, but remember Fundamental Tenet of Cryptograph?
- For “obscure mathematical reasons:”
  - ➠ p and $\frac{p-1}{2}$ should be prime
  - ➠ $g^{\frac{p-1}{2}} = -1 \mod p$

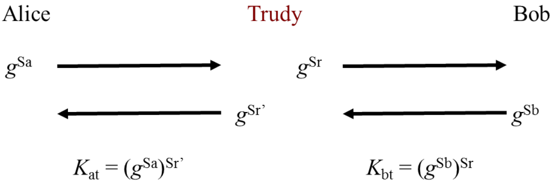
Man-in-the-Middle Attack

- Alice and Bob both negotiated a key with Trudy



Man-in-the-Middle Attack

- Alice and Bob both negotiated a key with Trudy
- Trudy forwards messages between Alice and Bob
  - Alice → Bob: $E(K_{bt}, D(K_{at}, c_{ab}))$
  - Bob → Alice: $E(K_{at}, D(K_{bt}, c_{ba}))$



Defense Against MITM Attacks

- Published DH numbers
  - ➠ everybody agrees upon a p and g, and publishes his g^S
  - ➠ grab the other's g^S then compute the secret
  - ➠ eliminate the need for the first two messages in DH protocol

Defense Against MITM Attacks

- Published DH numbers
  - ➠ everybody agrees upon a p and g, and publishes his g^S
  - ➠ grab the other's g^S then compute the secret
  - ➠ eliminate the need for the first two messages in DH protocol
- Authenticated Diffie-Hellman
  - ➠ share a secret or publish one's public key in advance
  - ➠ there are various ways to mix Diffie-Hellman and authentication

DSS (Digital Signature Standard)

- An algorithm designed by NIST for digital signature
  - ➠ the algorithm is known as DSA (digital signature algorithm)
  - ➠ messages are digested first, the digest is then signed
  - ➠ the sister function, SHS, is a message digesting function

DSS (Digital Signature Standard)

- An algorithm designed by NIST for digital signature
  - ➠ the algorithm is known as DSA (digital signature algorithm)
  - ➠ messages are digested first, the digest is then signed
  - ➠ the sister function, SHS, is a message digesting function
- Signing is faster than verification
  - ➠ e.g., for use in the smart cards

Digital Signature Algorithm

To generate DSA signature

1. Generate p and q (public)
   - ➠ q: 160-bit prime, p: 512-bit prime, and $p = kq + 1$

Digital Signature Algorithm

To generate DSA signature

1. Generate p and q (public)
   - ➠ q: 160-bit prime, p: 512-bit prime, and $p = kq + 1$
2. Generate g such that $g^q = 1 \mod p$ (public)
   - ➠ pick random number h, let $g = h^{\frac{p-1}{q}} = h^k$ (Fermat's theorem)

Digital Signature Algorithm

To generate DSA signature

1. Generate p and q (public)
   - ➠ q: 160-bit prime, p: 512-bit prime, and $p = kq + 1$
2. Generate g such that $g^q = 1 \mod p$ (public)
   - ➠ pick random number h, let $g = h^{\frac{p-1}{q}} = h^k$ (Fermat's theorem)
3. Choose a long-term public/private key pair $< T, S >$
   - ➠ pick a random number $S < q$, let $T = g^S \mod p$

Digital Signature Algorithm

To generate DSA signature

1. Generate p and q (public)
   - ➠ q: 160-bit prime, p: 512-bit prime, and $p = kq + 1$
2. Generate g such that $g^q = 1 \mod p$ (public)
   - ➠ pick random number h, let $g = h^{\frac{p-1}{q}} = h^k$ (Fermat's theorem)
3. Choose a long-term public/private key pair $< T, S >$
   - ➠ pick a random number $S < q$, let $T = g^S \mod p$
4. Choose a per message public/private key pair $< T_m, S_m >$
   - ➠ pick a random S_m, let $T_m = ((g^{S_m} \mod p) \mod q)$

Digital Signature Algorithm...

To generate DSA signature

5. Calculate message digest d_m

Digital Signature Algorithm...

To generate DSA signature

5. Calculate message digest d_m
6. Compute the signature $X = S_m^{-1}(d_m + ST_m) \mod q$

Digital Signature Algorithm...

To generate DSA signature

5. Calculate message digest d_m
6. Compute the signature $X = S_m^{-1}(d_m + ST_m) \mod q$
7. The signed message include
   - m: the message, T_m: per-message publish key, X: signature
   - the public key is $< p, q, g, T >$

Digital Signature Algorithm...

To generate DSA signature

5. Calculate message digest d_m
6. Compute the signature $X = S_m^{-1}(d_m + ST_m) \mod q$
7. The signed message include
   - m: the message, T_m: per-message publish key, X: signature
   - the public key is $< p, q, g, T >$

Digital Signature Algorithm...

To verify DSA signature

public information: $< p, q, g, T, T_m, m, X >$

1. calculate $X^{-1} \mod q$ and d_m

Digital Signature Algorithm...

To verify DSA signature

public information: $< p, q, g, T, T_m, m, X >$

1. calculate $X^{-1} \mod q$ and d_m
2. $x = d_m X^{-1} \mod q$
   $y = T_m X^{-1} \mod q$
   $z = (g^x T^y \mod p) \mod q$

Digital Signature Algorithm...

To verify DSA signature

public information: $< p, q, g, T, T_m, m, X >$

1. calculate $X^{-1} \mod q$ and d_m
2. $x = d_m X^{-1} \mod q$
   $y = T_m X^{-1} \mod q$
   $z = (g^x T^y \mod p) \mod q$
3. if $z = T_m$, the signature is verified

Why DSA Works

Let $v = (d_m + ST_m)^{-1} \mod q$:

Why DSA Works

Let $v = (d_m + ST_m)^{-1} \mod q$:

➠ $X^{-1} = (S_m^{-1}(d_m + ST_m))^{-1} = S_m(d_m + ST_m)^{-1} = S_m v \mod q$

Why DSA Works

Let $v = (d_m + ST_m)^{-1} \mod q$:

➠ $X^{-1} = (S_m^{-1}(d_m + ST_m))^{-1} = S_m(d_m + ST_m)^{-1} = S_m v \mod q$

➠ $x = d_m X^{-1} = d_m S_m v \mod q$

Why DSA Works

Let $v = (d_m + ST_m)^{-1} \mod q$:

- $X^{-1} = (S_m^{-1}(d_m + ST_m))^{-1} = S_m(d_m + ST_m)^{-1} = S_m v \mod q$
- $x = d_m X^{-1} = d_m S_m v \mod q$
- $y = T_m X^{-1} = T_m S_m v \mod q$

Why DSA Works

Let $v = (d_m + ST_m)^{-1} \mod q$:

➠ $X^{-1} = (S_m^{-1}(d_m + ST_m))^{-1} = S_m(d_m + ST_m)^{-1} = S_m v \mod q$

➠ $x = d_m X^{-1} = d_m S_m v \mod q$

➠ $y = T_m X^{-1} = T_m S_m v \mod q$

➠ $z = g^x T^y = g^{d_m S_m v} g^{S T_m S_m v} = g^{(d_m + ST_m) S_m v}$
$= g^{S_m} = T_m \mod p \mod q$

DSA Pitfalls

Private key (S) can be revealed if

- per-message private key S_m is leaked
  ➠ $X_m = S_m^{-1}(d_m + ST_m) \rightsquigarrow (X_m S_m - d_m) T_m^{-1} \mod q = S \mod q$

DSA Pitfalls

Private key (S) can be revealed if

- per-message private key S_m is leaked
  ➠ $X_m = S_m^{-1}(d_m + ST_m) \leadsto (X_m S_m - d_m) T_m^{-1} \mod q = S \mod q$
- two messages are signed with the same per-message private key
  ➠ $(X_m - X'_m)^{-1}(d_m - d'_m) \mod q = S_m \mod q$

Summary

- Modular arithmetic
- RSA
- Diffie-Hellman
- DSA

- Next lecture: authentication