Lecture 8 Encryption, Decryption

Basic concepts, symmetric-key algorithm, public-key algorithm, security protocol

1. Basic concepts

1) symbols

p: plain text (p1, p2, ..., pn) // before encryption

c: cipher text (c1, c2, ..., cm) // after encryption

enc: encryption algorithm

dec: decryption algorithm

e: encryption key

d: decryption key

c = enc(p, e) // encryption. e is an encryption key

p = dec(c, d) // decryption. d is a decryption key

2) basic encryption technique

substitution, transposition

ex0) shift cipher (Caesar's cipher)

plain: a b c d .... x y z

cipher: d e f g ... a b c

shift amount = 3.

Encryption key is the shift amount. Key space size = 25

ex1) substitution

plain: a b c d .......z

cipher: d f e r .......k

Encryption key is the cipher line. Key space size = 26! (> 4\*10^26)

Brute force attack for ex1 is virtually impossible even if we know the encryption algorithm because of the large key space, but frequency attack is possible. Each letter has a known frequency in English sentences.

'a': 8.167%, 'b':1.492%, 'e':12.702%, 't': 9.056%, ...

We can match a cipher letter which has 8.1667% frequency to 'a', etc.

2. Symmetric-key algorithm

In symmetric-key algorithm, encryption key is equal to or closely related to the decryption key. Closely related means the other key can be easily computed from one of the keys.

DES, AES, Blowfish, RC4, IDEA, ....

DES(Data Encryption Standard)

ref: "The DES Algorithm Illustrated" :

http://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm

0) Select a 64-bit key. Derive 16 48-bit sub-keys from it: K[1], K[2], …, K[16]

1) Divide plain text into 64-bit blocks

2) For each 64-bit block, T

T0 = IP(T)

T1 = E(T0)

T2 = E(T1)

...............

T16 = E(T15)

T16' = R16L16 -- reverse left half and right half

T17 = FP(T16')

T17 is the encrypted one for plain text T.

3) IP is the initial permutation and FP is the final permutation.

4) T[j] = E(T[j-1])

a) divide T[j-1] into L[j-1], the left half, and R[j-1], the right half

b) L[j]=R[j-1]

c) R[j]=L[j-1] xor f(R[j-1], K[j])

where f is

- expand R[j-1] to 48 bits using "expansion permutation"

- R[j-1] xor K[j]

- divide the result into 6-bit block (we will have 8 blocks)

- perform s-box substitution for each block.

each block becomes 4 bit after subtituion

- permute the resulting 32 bit

3. problem of symmetric-key algorithm

- fast, easy to use

- key distribution problem

-- you want to send credit card number to G-market

-- to use DES, you and the G-market need to share the same 64-bit key.

how can you and G-market exchange the key?

-- G-market creates a 64-bit key and send to you by email or by phone?

4. public-key algorithm

Encryption key is different from the decryption key.

Computing the other key from one of the keys is very hard (virtually impossible).

RSA, ElGamal, DSA, elliptic curve, ..........

5. RSA

1) basic concepts

relative prime: two number are relative prime to each other if the GCD of them =1

ex) 21, 10 are relative prime => GCD(21, 10) = 1

phi function: phi(N) is the number of relative prime to N in [1..N-1]

ex) phi(8) = 4 because we have {1, 3, 5, 7} for relative primes.

phi(7) = 6

phi(N) = N - 1, if N is a prime number.

phi(P\*Q) = (P-1)\*(Q-1), if P and Q are prime numbers

ex) phi(15) = phi(3\*5) = 2\*4=8 //{1, 2, 4, 7, 8, 11, 13, 14}

We can easily compute phi for a number x by finding two prime factors for x.

2) algorithm

a) select two prime numbers p and q (about 200 digits each)

b) n = p\*q

c) phi1 = (p-1)(q-1) // phi1 is the number of relative prime numbers for n

d) select e = relative prime to phi1, and < phi1

e) compute d such that d\*e = 1 (mod (p-1)(q-1))

d = e^(phi2-1) mod phi1,

where phi2 is the number of relative prime numbers for phi1

Now

n: modulus

e: public exponent

d: private exponent

(n, e): public key

(n, d): private key

f) encryption: C = M^e mod n, where M is the plain text. Usually a number of consecutive letters are grouped together to produce a number, which will be used as a plain text. The resulting number should be less than n. For example, if we are using 26 characters, and each 4 consecutive letters are grouped to produce a plain text, the plain text for letters abcd would be

97\*26^0+98\*26^1+99\*26^2+100\*26^3

(refer http://www.math.mtu.edu/mathlab/COURSES/holt/dnt/phi4.html)

g) decryption: M = C^d mod n

ex) p = 3, q = 5 ==> n=15

phi(n)=(3-1)(5-1)=8 // they are 1, 2, 4, 7, 8, 11, 13, 14

To count relative primes for 15

1) 1, 2, 3, 4, ..........., 15 // 15 candidates

2) remove multiple of 3 // these numbers can't be relative prime numbers

// there are 5 of them

remove multiple of 5 // these numbers can't be relative prime numbers

// there are 3 of them

3) number 15 will be removed twice, so add 1

4) 15 - (5+3) + 1 = 8

No suitable e!!

ex) p=11, q=17 ==> n=187

phi1=phi(187) = (11-1)(17-1)=160

phi2=phi(160)=64

choose e=7

d=e^(phi2-1) mod phi1 =7^63 mod 160 =23

Now try encryption and decryption.

M = 144

C = M^e mod n = 144^7 mod 187 =100

M = C^d mod n = 100^23 mod 187 =144

3) security of RSA

n, e is known. Computing d from n and e is almost impossible. The number of bits of “n” is the key length of this RSA. 1024 bit is considered medium security.

(n,e): public key

(n,d): private key

4) using RSA

- Every one has (e, d) pair.

- Every one publicizes his own public key e (in the form of a public key certificate)

- To send a message "m" secretly to person A, encrypt it with A's public key "e\_A"

c = m^e\_A mod n\_A

- No one can read this except A. A decrypts it by

c^d\_A mod n\_A

- To make it more secure, if B is the sender,

c = enc(enc(m, d\_B), e\_A)

6. Usage of public-key system

1) digital signature by A

(M, enc(H(M), d\_A))

M: document

H(M): hash value of M

enc(H(M), d\_A): encrypt H(M) with the private key of A

The receiver can check the validity of M by comparing its hash value with H(M). the H(M) can be obtained by decrypting enc(H(M), d\_A) with e\_A which is public.

If someone has changed M, the H(M) will not match. If someone, B, make a change in M and provide enc(H(M), d\_B), the decryption with e\_A will fail.

2) public key certificate (X. 509)

CA(Certificate Authority) : provide digital signature to a certificate

A person or an organization (company, web cite, ...) can obtain a certificate from one of the CA's. The CA verifies the information in the certificate and attaches a digital signature on it. X.509 specifies what information should be provided in the certificate.

X.509 field:

version : X.509 version

client's public key: public key of the client (private key is stored in client' PC)

public key algorithm ID

serial number

client info

valid date

CA info

encryption algorithm ID used in digital signature

digital signature

X.509 file can be stored in various formats such as DER, PEM, cer, crt, PFX, ...

ex) A certificate issued to "FreeSoft". CA is "Thawte"

Certificate:

Data:

Version: 1 (0x0)

Serial Number: 7829 (0x1e95)

Signature Algorithm: md5WithRSAEncryption

Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,

OU=Certification Services Division,

CN=Thawte Server CA/emailAddress=server-certs@thawte.com

Validity

Not Before: Jul 9 16:04:02 1998 GMT

Not After : Jul 9 16:04:02 1999 GMT

Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,

OU=FreeSoft, CN=www.freesoft.org/emailAddress=baccala@freesoft.org

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:

33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:

66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:

70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:

16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:

c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:

8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:

d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:

e8:35:1c:9e:27:52:7e:41:8f

Exponent: 65537 (0x10001)

Signature Algorithm: md5WithRSAEncryption

93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:

92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:

ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:

d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:

0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:

5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:

8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:

68:9f

To confirm the validity of the above certificate, we need a certificate of Thawte.

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 1 (0x1)

Signature Algorithm: md5WithRSAEncryption

Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,

OU=Certification Services Division,

CN=Thawte Server CA/emailAddress=server-certs@thawte.com

Validity

Not Before: Aug 1 00:00:00 1996 GMT

Not After : Dec 31 23:59:59 2020 GMT

Subject: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,

OU=Certification Services Division,

CN=Thawte Server CA/emailAddress=server-certs@thawte.com

**Subject Public Key Info**:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:d3:a4:50:6e:c8:ff:56:6b:e6:cf:5d:b6:ea:0c:

68:75:47:a2:aa:c2:da:84:25:fc:a8:f4:47:51:da:

85:b5:20:74:94:86:1e:0f:75:c9:e9:08:61:f5:06:

6d:30:6e:15:19:02:e9:52:c0:62:db:4d:99:9e:e2:

6a:0c:44:38:cd:fe:be:e3:64:09:70:c5:fe:b1:6b:

29:b6:2f:49:c8:3b:d4:27:04:25:10:97:2f:e7:90:

6d:c0:28:42:99:d7:4c:43:de:c3:f5:21:6d:54:9f:

5d:c3:58:e1:c0:e4:d9:5b:b0:b8:dc:b4:7b:df:36:

3a:c2:b5:66:22:12:d6:87:0d

Exponent: 65537 (0x10001)

X509v3 extensions:

X509v3 Basic Constraints: critical

**CA:TRUE**

Signature Algorithm: md5WithRSAEncryption

07:fa:4c:69:5c:fb:95:cc:46:ee:85:83:4d:21:30:8e:ca:d9:

a8:6f:49:1a:e6:da:51:e3:60:70:6c:84:61:11:a1:1a:c8:48:

3e:59:43:7d:4f:95:3d:a1:8b:b7:0b:62:98:7a:75:8a:dd:88:

4e:4e:9e:40:db:a8:cc:32:74:b9:6f:0d:c6:e3:b3:44:0b:d9:

8a:6f:9a:29:9b:99:18:28:3b:d1:e3:40:28:9a:5a:3c:d5:b5:

e7:20:1b:8b:ca:a4:ab:8d:e9:51:d9:e2:4c:2c:59:a9:da:b9:

b2:75:1b:f6:42:f2:ef:c7:f2:18:f9:89:bc:a3:ff:8a:23:2e:

70:47

7. Homework

1) Implement Caesar's cipher system. Write a program that breaks this system.

2) Implement DES

ref: "The DES Algorithm Illustrated" :

http://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm

3) Make an RSA key pair from two small prime numbers (p=11, q=3). Encrypt/decrypt for some example numbers. The example number should be in the range of 0 to 32. Try to encrypt HELLO using this RSA key.

4) Write a program that allows users to generate an RSA key and encrypt/decrypt with this key. Use the fact

x\*y mod n = (x mod n)\*(y mod n) mod n

For example, we want to compute a^b mod m. In pseudocode,

a1 = a mod m

p=1

for(int i=1;i<=b;i++){

p \*= a1

p = p mod m

}

Now p is the result for a^b mod m.

To compute gcd(a, b), use following pseudocode:

for(;;){

if (b==0) break;

t=b;

b=a % b;

a=t;

}

Now the resulting a is the gcd(a, b).

5) Write a program that breaks your RSA system.

6) Write a pair of secure client and server that exchange a password using RSA system.

7) Install openssl in your pc.

download openssl from http://gnuwin32.sourceforge.net/packages/openssl.htm

(get "complete package except source")

install in your pc

go to the installed directory/bin

double click on openssl (you may need to run as "administrator")

8-1) Generate an RSA key pair using openssl.

openssl> genrsa -out mykey.pem 1024

// generate public/private key pair in file "mykey.pem" with keysize 1024 bits.

// default size 512 bits if keysize is not specified

8-2) Convert mykey.pem to a text file, mykey.txt, to look at the contents. Use WordPad to open mykey.txt. Find n, e, and d.

openssl> rsa -in mykey.pem -text -out mykey.txt

// display the contents of "mykey.pem" in plain text in

// output file "mykey.txt"

8-3) Encrypt "hello" with (n, e) to produce ciphertext. What is the size of the ciphertext? Decrypt the ciphertext with (n, d) to recover "hello".

openssl> rsautl -encrypt -inkey mykey.pem -in myplain.txt -out mycipher // encrypting

openssl> rsautl -decrypt -inkey mykey.pem -in mycipher -out mycipher.dec // decrypting

Refer the manual in man/pdf/openssl-mal.pdf.

PEM(Privacy Enhanced Mail) file format transforms a binary file into an ascii file using base64. Each 6 bit in the input file will be converted to a letter in {A-Z, a-z, 0-9, +, -}, and wrapped with boundary lines.

ex) Man ==> 77 97 110 ==> 01001101 01100001 01101110

==> T(010011, 19) W(010110, 22) F(000101, 5) u(101110, 46)

(.der : binary DER encoded certificates

.cer : similar to .der

.key : PKCS#8 keys. the keys are encoded as binary DER or ASCII PEM)

9) Make an X.509 certificate.

9.1) make a config file "myconf.txt"

[req]

string\_mask = nombstr

distinguished\_name = req\_distinguished\_name

prompt = no

[req\_distinguished\_name]

commonName = my CA

stateOrProvinceName = some state

countryName = US

emailAddress = root@somename.somewhere.com

organizationName = mycompany

9.2) Make a certificate for the person/company specified in myconf.txt. The public key of this person/company is given in mykey.pem:

req -config myconf.txt -new -x509 -key mykey.pem -out mycert.pem

9.3) let's read the contents of the certificate

x509 -in mycert.pem -text -out mycert.txt

Who is the owner of this certificate? What is the public key? What is the key size?

Who has signed this certificate?

10) Get a certificate in Internet Explorer or in Chrome. Check the contents of x.509 file.

10.1) Go to "tools>internet options>contents>certificates" to get a copy of a certificate. To view the certificate of a site:

In Chrome, go to some https site such as [www.daum.net](http://www.daum.net) and select Three Dots Menu>More Tools>Developer Tools>Security>View certificate>Details>Copy to File

In IE, go to some https site such as [www.daum.net](http://www.daum.net) and click the padlock symbol, select “View Certificate”>Detail>Copy to File.

10.2) Look at the contents of this certificate (assume the file name is daum.cer) with

x509 -in daum.cer -text -out daum.cer.txt -inform DER

Who is the owner of this certificate? Who has signed this certificate? What is the public key? What is the key size?

(DER: Distinguished Encoding Rules)