Asymmetric Key Encryption

BMEVITMAV52

Information and Network Security

feher.gabor@tmit.bme.hu

Public-key encryption

- Problems with symmetric-key encryption
  - The key should be transferred secretly

- Public-key encryption
  - Asymmetric keys: public and private keys
    - It should be computationally infeasible to get the private key from the public key
  - $C=E_K(P)$, $P=D_{K^*}(C)$ or
  - $C=E_{K^*}(P)$, $P=D_K(C)$
  - The public key is supposed to be known
    - Should be authentic!
    - Resistance to chosen-plaintext attacks

Public-key encryption

- Asymmetric block ciphers
  - Large messages:
    - ECB or CBC
    - CFB, OFB, CTR is not possible (only encryption is used)
  - They are lot more slower than symmetric ones!
- Asymmetric stream ciphers
  - Possible
  - Infrequent, no details here

RSA

- 1977, Inventors: Ron Rivest, Adi Shamir, Len Adleman (MIT)
- The most widely used public-key encryption algorithm

RSA key generation

- 1. p and q are two large prime numbers
  - More than 500 bits
  - Generating large primes: generate and test
    - Provable prime or probably prime
- 2. $n = p \cdot q$ **MODULUS**
- 3. $\varphi(n) = (p-1)(q-1)$
  - Euler's totient, Euler's phi function, phi function: The number of positive integers, which are less than or equal to n and they are also coprime to n
- 4. Chose *e*, where $1 < e < \varphi(n)$ and e coprime to $\varphi(n)$ **Public key component**
- 5. Compute d that $d \cdot e \equiv 1 \pmod{\varphi(n)}$ **Private key component**

$$\varphi(n) = |\{ k \in Z \mid 0 < k \leq n \wedge (n,k) = 1 \}| \quad (n \in N)$$

- Public key:
  - n, e
- Private key:
  - n, d

RSA encryption

- M message is turned into number m
  - $m < n$
- Encryption
  - $c = m^e \bmod n$
- Decryption
  - $m = c^d \bmod n$
    - $c^d \equiv (m^e)^d \equiv m^{ed} \pmod{n}$
      - Since $ed \equiv 1 \pmod{p-1}$ and $ed \equiv 1 \pmod{q-1}$
        Due to the Fermat's little theorem generalization:
        if $x \equiv y \pmod{p-1}$ then $a^x \equiv a^y \pmod{p}$
    - $m^{ed} \equiv m \pmod{p}$ and $m^{ed} \equiv m \pmod{q}$
      - Using the Chinese reminder theorem
    - $m^{ed} \equiv m \pmod{pq}$
    - $c^d \equiv m \pmod{n}$

a=m
x=ed
y=1

RSA example

- Two primes: $p = 61$, $q = 53$
  - This is just an example, thus these primes are small
- $n = pq = 3233$, $\varphi(n) = 3120$
- Let $e = 17$
  - Public key: 17, 3233
- $d = 2753$
  - Private key: 2753, 3233

- Message: $m = 123$
  - Encryption: $c = 123^{17} \bmod 3233 = 855$
  - Decryption: $m = 855^{2753} \bmod 3233 = 123$

Accelerate calculations

- Modular exponentiation
  - $x \equiv y^e \bmod n$
  - Straightforward method
    - Multiply y e-times and then make a division
      O(e) calculations
    - Using large (enormous) numbers this is really slow
  - Memory efficient method
    - $i \equiv (j \cdot k) \bmod n$
    - $i = ((j \bmod n) \cdot (k \bmod n)) \bmod n$
    - Perform modulo operation at each round
    - Repeat $x \equiv (x \cdot y) \bmod n$; e times (recursive, x=1 first)
    - Same complexity: O(e), but smaller memory footprint
  - Using square-and-multiply
    - $x^n =$

      $$x^n = \begin{cases} x & i = 1: \\ (x^2)^{n/2} & \text{even } i: \\ x(x^2)^{(n-1)/2} & \text{odd } i: \end{cases}$$

    - Complexity is O(log e)

$$\underbrace{2 \cdot 2}_{4} \cdot \underbrace{2 \cdot 2}_{4}$$

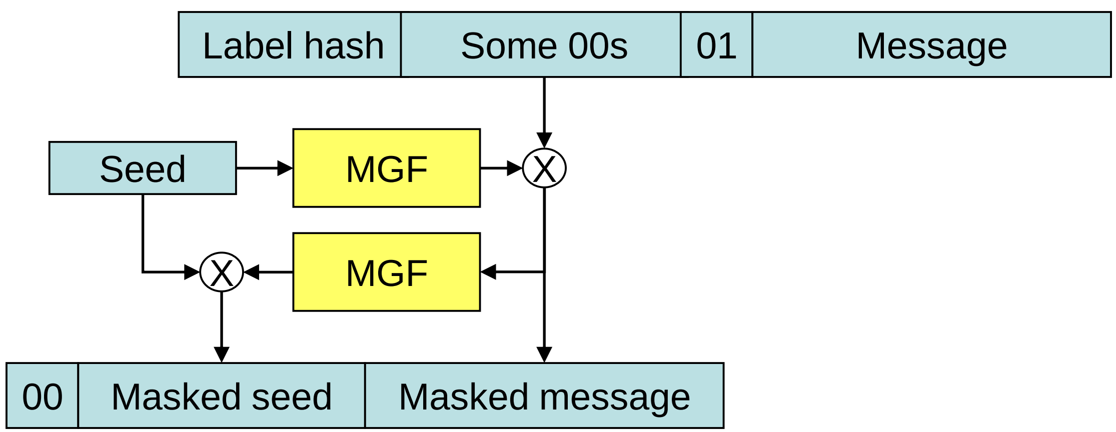
Message padding

- The algorithm has problems
  - If $m = 0$: the result is 1 for every n
  - If $m = 1$: the ciphertext is the same as the plaintext
  - If e is not big enough: for small m no modulo operations performed and thus m can be recovered using the e-th root
    - Example $17^3 < n$ (For reasonable modulus)
- The solution is to change m with padding
  - Add some bits to the message

PKCS #1

- Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography

- RSA message masking (padding)



Faster RSA

Usually this one is used

- Faster decryption
  - Universal exponent
    - Instead of φ(n) use λ(n) = lcm(p-1,q-1) (least common multiple) can be used
      - This results smaller d thus faster decryption
- Faster encryption
  - Use small *e* (but not too small)
    - can be a security risk if the same message is encrypted with different modulus
  - *e* is usually 65537

RSA public key (NEPTUN)

```
Modulus (2048 bits):
c3 65 a6 8f f6 52 f6 43 ea 00 16 6b 34 fe 7b 6d
0c a8 71 7d 80 42 58 50 34 ff 09 5c 5c 9f 96 31
2a b7 62 65 13 5a fb 4a da fb c7 ff 0f 82 e2 a5
1d ce b8 4a 84 20 38 df ba a2 59 86 12 71 0b ac
34 c1 37 94 9a 87 42 bd fa 01 55 7d fc 44 b2 50
7f a6 be 14 2f 4d da 59 bc 31 09 de 98 60 bb f9
98 ae 83 98 0a 3d a0 61 0e 7b d9 38 ab ff ee 5d
82 27 f0 5d c5 20 16 b3 48 35 fc eb 50 c2 c6 27
55 91 63 64 da 5f 20 0e 84 f2 19 6f 95 ce 1d 96
9f 24 0e e9 96 2f bd 5c cb 9e ea d3 e5 c0 1c f8
22 17 57 00 77 fb c3 70 5b 33 7e 66 83 62 d1 a3
e3 97 cd 9e ab 04 d2 b2 8e b4 df 59 9d 1b ca 64
22 97 6e cd 32 c3 a0 57 81 9b 9c 62 cc 88 a9 58
17 11 5f ad 7d 6b e6 6c 94 6b 14 ba 13 c5 16 ed
7e c5 07 cb 91 2d ec 6e 5b 38 a3 5b 17 f5 03 7f
2d aa 48 89 cf 44 7f 26 ce f7 16 5a 5c e1 84 e7

Exponent (24 bits):
65537
```

RSA Algorithm Security

- Problem of factoring very large numbers
  - No efficient algorithm is known

- Possible security problems
  - Unconcealed messages
    - $m^e \bmod n = m$
    - Small encryption exponent
  - Homomorphism
    - $(m_1 m_2)^e \equiv m_1^e \cdot m_2^e \equiv c_1 c_2 \pmod{n}$
    - Problem if the receiver decrypts arbitrary text
      - $(c \cdot r^e)^d \equiv c^d r^{ed} \equiv m \cdot r \pmod{n}$

Blinding

- Input and the function on the input is separated. The function must not know the real input
  - In general
    - y=f(x)
    - Blinding: E(), D() bijection
    - Instead of f(x) we do f(E(x))
    - y=D(f(E(x))), **but it does not work for all f, E and D !**

- RSA
  - In order to avoid side channel attacks (using timing information)
  - $E(x)=xr^e \bmod n$, $f(x)=x^d \bmod n$, $D(x)=x/r \bmod n$

  - Instead of $c^d \; mod \; n$, we calculate $(r^e c)^d \; mod \; n$ and this way the result is *rm mod n* (due to Homomorphism)
  - We always use a new r for each encryption/decryption

ElGamal public-key encryption

- Taher Elgamal in 1984
- Key generation
  - Generate large prime: p
  - g is a generator of a multiplicative group Zp* (1,2, … p-1)
  - a is random, $1 \le a \le p-2$
  - Compute $A = g^a \bmod p$
  - Public key: p, g, A
  - Private key: p, g, a

ElGamal encryption

- Message is turned into m
  - $0 \leq m \leq p-1$
- Encryption:
  - 1. Select a random number: r
  - 2. Compute: $R = g^r \bmod p$
  - 3. Compute: $C = m \cdot A^r \bmod p$
  - The cipher text is R,C
- Decryption:
  - $m = C \cdot R^{p-1-a} \bmod p$
  - $C \cdot R^{p-1-a} \equiv m \cdot A^r \cdot R^{p-1-a} \equiv m \cdot g^{ar} \cdot g^{r(p-1-a)} \equiv m \cdot g^{p-1} \equiv m \pmod p$

ElGamal example

- Key generation:
  - Prime and generator: $p = 2357$, $g = 2$
  - Private key: $a = 1751$
  - Public key: 2357, 2, $2^{1751} \bmod 2357 = 1185$
- Encryption:
  - Message: $m = 2035$
  - Select $r = 1520$
  - Computer $R = 2^{1520} \bmod 2357 = 1430$
  - Ciphertext = $2035 \cdot 1185^{1520} \bmod 2357 = 697$
  - Send 1430 and 697
- Decryption
  - $697 \cdot 1430^{2357-1-1751} \bmod 2357 = 2035$

ElGamal properties

- Encryption is slower than decryption
  - Two exponentiations are used
- Ciphertext is two times longer than the plaintext

Other than RSA

- RSA is very slow
  - Encrypt only a symmetric key! (128-256 bit)

- Faster methods
  - El Gamal method
  - Elliptic curve cryptography

NIST javaslata az egyes kriptorendszerek kulcshosszának összehasonlítására:			
ECC modulus	AES	RSA modulus	RSA:ECC
112	56	512	5:1
161	80	1024	6:1
256	128	3072	12:1
384	192	7680	20:1
512	256	15630	30:1

References

- Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, “Handbook of Applied Cryptography”, CRC Press, ISBN: 0-8493-8523-7
  - http://www.cacr.math.uwaterloo.ca/hac/
- Wikipedia - The free encyclopedia
  - http://www.wikipedia.org/