

RSA Cryptosystem

Bits	PCs	Memory
430	1	128MB
760	215,000	4GB
1,020	342×10^6	170GB
1,620	1.6×10^{15}	120TB

Outline

- ◆ Euler's theorem
- ◆ RSA cryptosystem
 - Definition
 - Example
 - Security
 - Correctness
- ◆ Algorithms for RSA
 - Modular power
 - Modular inverse
 - Randomized primality testing

Euler's Theorem

- ◆ The multiplicative group for \mathbf{Z}_n , denoted with $\mathbf{Z}_{n'}^*$, is the subset of elements of \mathbf{Z}_n relatively prime with n
- ◆ The totient function of n , denoted with $\phi(n)$, is the size of \mathbf{Z}_n^*

- ◆ Example

$$\mathbf{Z}_{10}^* = \{ 1, 3, 7, 9 \} \quad \phi(10) = 4$$

- ◆ If p is prime, we have

$$\mathbf{Z}_p^* = \{ 1, 2, \dots, (p - 1) \} \quad \phi(p) = p - 1$$

Euler's Theorem

For each element x of $\mathbf{Z}_{n'}^*$ we have $x^{\phi(n)} \bmod n = 1$

- ◆ Example ($n = 10$)

$$3^{\phi(10)} \bmod 10 = 3^4 \bmod 10 = 81 \bmod 10 = 1$$

$$7^{\phi(10)} \bmod 10 = 7^4 \bmod 10 = 2401 \bmod 10 = 1$$

$$9^{\phi(10)} \bmod 10 = 9^4 \bmod 10 = 6561 \bmod 10 = 1$$

RSA Cryptosystem

◆ Setup:

- $n = pq$, with p and q primes
- e relatively prime to $\phi(n) = (p - 1)(q - 1)$
- d inverse of e in $\mathbb{Z}_{\phi(n)}$

◆ Keys:

- Public key: $K_E = (n, e)$
- Private key: $K_D = d$

◆ Encryption:

- Plaintext M in \mathbb{Z}_n
- $C = M^e \bmod n$

◆ Decryption:

- $M = C^d \bmod n$

◆ Example

■ Setup:

- ◆ $p = 7, q = 17$
- ◆ $n = 7 \cdot 17 = 119$
- ◆ $\phi(n) = 6 \cdot 16 = 96$
- ◆ $e = 5$
- ◆ $d = 77$

■ Keys:

- ◆ public key: (119, 5)
- ◆ private key: 77

■ Encryption:

- ◆ $M = 19$
- ◆ $C = 19^5 \bmod 119 = 66$

■ Decryption:

- ◆ $C = 66^{77} \bmod 119 = 19$

Complete RSA Example

◆ Setup:

- $p = 5, q = 11$
- $n = 5 \cdot 11 = 55$
- $\phi(n) = 4 \cdot 10 = 40$
- $e = 3$
- $d = 27$ ($3 \cdot 27 = 81 = 2 \cdot 40 + 1$)

◆ Encryption

- $C = M^3 \pmod{55}$

◆ Decryption

- $M = C^{27} \pmod{55}$

<i>M</i>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
<i>C</i>	1	8	27	9	15	51	13	17	14	10	11	23	52	49	20	26	18	2
<i>M</i>	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
<i>C</i>	39	25	21	33	12	19	5	31	48	7	24	50	36	43	22	34	30	16
<i>M</i>	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
<i>C</i>	53	37	29	35	6	3	32	44	45	41	38	42	4	40	46	28	47	54

Security

- ◆ The security of the RSA cryptosystem is based on the widely believed difficulty of factoring large numbers
 - The best known algorithm takes exponential time
- ◆ A challenge sponsored by RSA Security offers prizes for the factorization of large numbers
- ◆ In 1999, a 512-bit challenge number was factored in 4 months using 35.7 CPU-years on the following computers:
 - 160 175-400 MHz SGI and Sun
 - 8 250 MHz SGI Origin
 - 120 300-450 MHz Pentium II
 - 4 500 MHz Digital/Compaq

- ◆ On December 3, 2003, a team of researchers factored the RSA-576 challenge number
- ◆ As of April 2004, the prize for factoring RSA-2048 is \$200,000

```
25195908475657893494027183240048398571429282126204
03202777713783604366202070759555626401852588078440
69182906412495150821892985591491761845028084891200
72844992687392807287776735971418347270261896375014
97182469116507761337985909570009733045974880842840
17974291006424586918171951187461215151726546322822
16869987549182422433637259085141865462043576798423
38718477444792073993423658482382428119816381501067
48104516603773060562016196762561338441436038339044
14952634432190114657544454178424020924616515723350
77870774981712577246796292638635637328991215483143
81678998850404453640235273819513786365643912120103
97122822120720357
```

- ◆ Estimated resources needed to factor a number within one year

Bits	PCs	Memory
430	1	128MB
760	215,000	4GB
1,020	342×10^6	170GB
1,620	1.6×10^{15}	120TB

Correctness

◆ We show the correctness of the RSA cryptosystem for the case when the plaintext M does not divide n

◆ Namely, we show that

$$(M^e)^d \bmod n = M$$

◆ Since $ed \bmod \phi(n) = 1$, there is an integer k such that

$$ed = k\phi(n) + 1$$

◆ Since M does not divide n , by Euler's theorem we have

$$M^{\phi(n)} \bmod n = 1$$

◆ Thus, we obtain

$$(M^e)^d \bmod n =$$

$$M^{ed} \bmod n =$$

$$M^{k\phi(n) + 1} \bmod n =$$

$$MM^{k\phi(n)} \bmod n =$$

$$M (M^{\phi(n)})^k \bmod n =$$

$$M (M^{\phi(n)} \bmod n)^k \bmod n =$$

$$M (1)^k \bmod n =$$

$$M \bmod n =$$

$$M$$

◆ See the book for the proof of correctness in the case when the plaintext M divides n

Algorithmic Issues

◆ The implementation of the RSA cryptosystem requires various algorithms

◆ Overall

- Representation of integers of arbitrarily large size and arithmetic operations on them

◆ Encryption

- Modular power

◆ Decryption

- Modular power

◆ Setup

- Generation of random numbers with a given number of bits (to generate candidates p and q)
- Primality testing (to check that candidates p and q are prime)
- Computation of the GCD (to verify that e and $\phi(n)$ are relatively prime)
- Computation of the multiplicative inverse (to compute d from e)

Modular Power

- ◆ The repeated squaring algorithm speeds up the computation of a modular power $a^p \bmod n$
- ◆ Write the exponent p in binary

$$p = p_{b-1}p_{b-2} \cdots p_1p_0$$

- ◆ Start with

$$Q_1 = a^{p_{b-1}} \bmod n$$

- ◆ Repeatedly compute

$$Q_i = ((Q_{i-1})^2 \bmod n) a^{p_{b-i}} \bmod n$$

- ◆ We obtain

$$Q_b = a^p \bmod n$$

- ◆ The repeated squaring algorithm performs $O(\log p)$ arithmetic operations

- ◆ Example

- $3^{18} \bmod 19$ ($18 = 10010$)

- $Q_1 = 3^1 \bmod 19 = 3$

- $Q_2 = (3^2 \bmod 19)3^0 \bmod 19 = 9$

- $Q_3 = (9^2 \bmod 19)3^0 \bmod 19 = 81 \bmod 19 = 5$

- $Q_4 = (5^2 \bmod 19)3^1 \bmod 19 = (25 \bmod 19)3 \bmod 19 = 18 \bmod 19 = 18$

- $Q_5 = (18^2 \bmod 19)3^0 \bmod 19 = (324 \bmod 19) \bmod 19 = 17 \cdot 19 + 1 \bmod 19 = 1$

p_{5-i}	1	0	0	1	0
$2^{p_{5-i}}$	3	1	1	3	1
Q_i	3	9	5	18	1

Modular Inverse

Theorem

Given positive integers a and b , let d be the smallest positive integer such that

$$d = ia + jb$$

for some integers i and j .

We have

$$d = \gcd(a,b)$$

◆ Example

- $a = 21$
- $b = 15$
- $d = 3$
- $i = 3, j = -4$
- $3 = 3 \cdot 21 + (-4) \cdot 15 = 63 - 60 = 3$

◆ Given positive integers a and b , the extended Euclid's algorithm computes a triplet (d,i,j) such that

- $d = \gcd(a,b)$
- $d = ia + jb$

◆ To test the existence of and compute the inverse of $x \in \mathbb{Z}_n$, we execute the extended Euclid's algorithm on the input pair (x,n)

◆ Let (d,i,j) be the triplet returned

- $d = ix + jn$

Case 1: $d = 1$

i is the inverse of x in \mathbb{Z}_n

Case 2: $d > 1$

x has no inverse in \mathbb{Z}_n

Pseudoprimality Testing

- ◆ The number of primes less than or equal to n is about $n / \ln n$
- ◆ Thus, we expect to find a prime among $O(b)$ randomly generated numbers with b bits each
- ◆ Testing whether a number is prime (primality testing) is believed to be a hard problem
- ◆ An integer $n \geq 2$ is said to be a base- x pseudoprime if
 - $x^{n-1} \bmod n = 1$ (Fermat's little theorem)
- ◆ Composite base- x pseudoprimes are rare:
 - A random 100-bit integer is a composite base-2 pseudoprime with probability less than 10^{-13}
 - The smallest composite base-2 pseudoprime is 341
- ◆ Base- x pseudoprimality testing for an integer n :
 - Check whether $x^{n-1} \bmod n = 1$
 - Can be performed efficiently with the repeated squaring algorithm

Randomized Primality Testing

◆ Compositeness witness function $witness(x, n)$ with error probability q for a random variable x

Case 1: n is prime

$witness w(x, n) = false$

Case 2: n is composite

$witness w(x, n) = false$ with probability $q < 1$

◆ Algorithm *RandPrimeTest* tests whether n is prime by repeatedly evaluating $witness(x, n)$

◆ A variation of base- x pseudoprimality provides a suitable compositeness witness function for randomized primality testing (Rabin-Miller algorithm)

Algorithm *RandPrimeTest*(n, k)

Input integer n , confidence parameter k and composite witness function $witness(x, n)$ with error probability q

Output an indication of whether n is composite or prime with probability 2^{-k}

$t \leftarrow k / \log_2(1/q)$

for $i \leftarrow 1$ **to** t

$x \leftarrow random()$

if $witness(x, n) = true$

return “ n is composite”

return “ n is prime”