The RSA Cryptosystem

Gianluca Dini
Dept. of Ingegneria dell'Informazione
University of Pisa
Email: gianluca.dini@.unipi.it

Version: 2023-04-19

1

The RSA Cryptosystem

BASICS

Apr-23 The RSA Cryptosystem 2

2

RSA in a nutshell

- Rivest-Shamir-Adleman, 1978
  - Rivest, R.; Shamir, A.; Adleman, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM 21 (2): 120–126, February 1978.
- The most widely used asymmetric crypto-system
- Patented until 2000 in US
- Many applications
  - Encryption of small pieces (e.g., key transport)
  - Digital Signatures
- Underlying one-way function: integer factorization problem

Apr-23 The RSA Cryptosystem 3

3

RSA one-way function

- One-way function y = f(x)
  - y = f(x) is easy
  - x = $f^{-1}(y)$ is hard
- RSA one-way function
  - Multiplication is easy
  - Factoring is hard

Apr-23 The RSA Cryptosystem 4

4

Mathematical setting

- RSA encryption and decryption is done in the integer ring $\mathbb{Z}_n = \{0, 1, \ldots, n\text{-}1\}$
  - PT and CT are elements in $\mathbb{Z}_n$
  - Modular computation plays a central role

Apr-23 5

5

Key Generation

1. Choose two large, distinct primes p, q
2. Compute modulus n = p × q
3. Compute Euler's Phi function $\phi(n) = (p-1) \times (q-1)$
4. Randomly select the public (encryption) exponent e, $1 < e < \phi(n)$, s.t. $\gcd(e, \phi(n)) = 1$
5. Compute the unique private (decryption) exponent d, $1 < d < \phi$, such that $e \cdot d \equiv 1 \pmod{\phi}$
6. Private key = (d, n), Public key = (e, n)

Apr-23 The RSA Cryptosystem 6

6

RSA Key Generation

- Comments
  - Primes p and q are 100÷200 decimal digits
    - Nowadays, p and q are 1024 bit
  - Condition $\gcd(e, \Phi(n)) = 1$ guarantees that d exists and is unique
  - At the end of key generation, p and q must be deleted
  - Two parts of the algorithm are nontrivial:
    - Step 1
    - Steps 4-5 (step 5 is crucial for RSA correctness)

Apr-23 The RSA Cryptosystem 7

7

RSA Encryption and Decryption Algorithm

- Encryption algorithm: to generate the ciphertext y from the plaintext $x \in [0, n – 1]$
  - Obtain receiver's authentic public key (n, e)
  - Compute $\mathbf{y = x^e \bmod n}$
- Decryption algorithm: to obtain the plaintext x from the ciphertext $y \in [0, n - 1]$
  - Compute $\mathbf{x = y^d \bmod n}$

Apr-23 The RSA Cryptosystem 8

8

Example with artificially small numbers

Key generation

- Let p = 47 e q = 71
  $n = p \times q = 3337$
  $\phi = (p-1) \times (q-1) = 46 \times 70 = 3220$
- Let e = 79
  $ed = 1 \bmod \phi$
  $79 \times d = 1 \bmod 3220$
  $d = 1019$

Encryption

Let $m = 9666683$

Divide m into blocks $m_i < n$

$m_1 = 966$; $m_2 = 668$; $m_3 = 3$

Compute

$c_1 = 966^{79} \bmod 3337 = 2276$

$c_2 = 668^{79} \bmod 3337 = 2423$

$c_3 = 3^{79} \bmod 3337 = 158$

$c = c_1c_2c_3 = 2276\ 2423\ 158$

Decryption

$m_1 = 2276^{1019} \bmod 3337 = 966$

$m_2 = 2423^{1019} \bmod 3337 = 668$

$m_3 = 158^{1019} \bmod 3337 = 3$

$m = 966\ 668\ 3$

The RSA Cryptosystem

PROOF OF RSA

RSA consistency: proof

- We need to prove that decryption is the inverse operation of encryption, $D_{privK}(E_{pubK}(x)) = x$
- Step 1
  - $d \cdot e = 1 \bmod \Phi(n)$
  - By definition of mod operator $d \cdot e = 1 + t \cdot \Phi(n)$ for some integer t
  - Insert this expression in the decryption: $y^d \equiv x^{ed} \equiv x^{1+t\cdot\Phi(n)} \equiv x \cdot x^{t\cdot\Phi(n)} \equiv x\cdot(x^{\Phi(n)})^t \bmod n$
- Step 2: prove that $x \equiv x\cdot(x^{\Phi(n)})^t \bmod n$
  - Recall
    - Euler's Theorem: if gcd(x, n) = 1 then $1 \equiv x^{\Phi(n)} \bmod n$
    - Minor generalization $1 \equiv 1^t \equiv (x^{\Phi(n)})^t \bmod n$

RSA consistency: proof

- Step 2
  - case 1: gcd(x, n) = 1
    - Euler's theorem holds ➔ $x\cdot(x^{\Phi(n)})^t \equiv x\cdot 1 \equiv x \bmod n$ Q.E.D.
  - case 2: gcd(x, n) $\neq$ 1
    - Since p and q are primes (and x < n) then either $x = r\cdot p$ or $x = s\cdot q$ with r < p and s < q
    - Assume $x = r\cdot p$ then gcd(x, q) = 1 ➔ Euler's Theorem holds in this form $1 \equiv (x^{\Phi(n)})^t \bmod q$
      - Proof: $(x^{\Phi(n)})^t \equiv (x^{(p-1)(q-1)})^t \equiv ((x^{\Phi(q)})^t)^{p-1} \equiv 1^{(p-1)} \equiv 1 \bmod q$
    - $(x^{\Phi(n)})^t = 1 + u\cdot q$, for some integer u
    - $x \cdot(x^{\Phi(n)})^t = x + x\cdot u\cdot q = x + (r\cdot p)\cdot u \cdot q = x + r\cdot u\cdot(q\cdot p) = x + r\cdot u\cdot n$
    - $x\cdot(x^{\Phi(n)})^t \equiv x \bmod n$ Q.E.D.

RSA encryption and decryption

- Comments
  - RSA proof is based on Euler's theorem
  - The proof becomes simpler by using the Chinese Remainder Theorem

The RSA Cryptosystem

PERFORMANCE

RSA

- RSA algorithms for key generation, encryption and decryption are “easy”
- They involve the following operations
  - Discrete exponentiation
  - Generation of large primes
  - Solving diophantine equations

Apr-23 The RSA Cryptosystem 15

15

Computation of e and d (refined)

- Select e $\in$ (1, φ(n))
- Apply EEA with input parameters n and e and obtain the relationship
  - gcd(Φ(n), e) = s$\cdot\varphi$(n) + t$\cdot$e (Diophantine equation)
    - If gcd(e, φ(n)) = 1 then
      - Parameter e is a valid public key
      - Unknown t = e^{-1} mod Φ(n), i.e., t = d mod Φ(n)
    - If gcd(e, Φ(n)) $\neq$ 1 then
      - Select another value for e and repeat the process
  - Efficiency
    - Number of steps is close to the number of digit of the input parameter ($\approx$ logarithmic)

Apr-23 The RSA Cryptosystem 16

16

Finding large primes

- Algorithm

```
repeat
    p ← RNG(x);       // secure random generator
until isPrime(p);     // primality test
```

- Comment
  - RNG must be secure, i.e., unpredictable
- Problems
  - How many random numbers we must test before we have a prime?
  - How fast can we check whether a random integer is prime?
  - It turns out that both steps are reasonably fast

How common are primes?

- Let Pi(x) be the number of prime less than x
- Prime Numbers Theorem
  - For a very large x, Pi(x) tends to x/ln(x)
  - Furthermore, primes are distributed approximately uniformly over [2, x]
- Probability to find a prime in [0, x] $\approx$ 2/(ln x)
  - As we test only odd numbers

$$P = (x/\ln x)/(x/2) = 2/\ln x$$

  - Expected number of trials to find a prime in [0, x] is (ln x)/2

Primality tests

- Primality tests are computationally much easier than factorization
- Practical primality tests are probabilistic
  - At the question: "is p* prime?" they answer
    - p* is composed which is always a true statement
    - p* is prime, which is only true with a high probability
- Primality test
  - Fermat test
  - Miller-Rabin test

Modular ops - complexity

- Bit complexity of basic operations in $\mathbb{Z}_n$
  - Let n be on k bits ($n < 2^k$)
  - Let a and b be two integers in $\mathbb{Z}_n$ (on k-bits)
    - Addition $a + b$ can be done in time $O(k)$
    - Subtraction $a – b$ can be done in time $O(k)$
    - Multiplication $a \times b$ can be done in $O(k^2)$
    - Division $b \times a^{-1}$ can be done in time $O(k^2)$
    - Inverse a^{-1} can be done in $O(k)$
    - Modular exponentiation can be done in $O(k^3)$

Fast exponentiation

- How many multiplications to compute 2^{20}?
- Grade-school Algorithm requires
  - $2 \times 2 \times 2 \times \ldots \times 2$ => 19 multiplications
- Square-and-Multiply Algorithm
  - $((2 \times (2^2)^2)^2)^2$ => 1 multiplications + 4 squares => 5 multiplications

Apr-23 The RSA Cryptosystem 21

21

Fast exponentiation

- RSA computes modular exponentiation
  - $a^x \bmod n$, where n is on k bits (i.e., $n \leq 2^k$)
- Grade-school Algorithm
  - requires $(x - 1)$ modular multiplications
    - If *x* is as large as n, which is exponentially large in k, the Grade-school Algorithm is inefficient
- Square-and-multiply Algorithm
  - requires up to 2k multiplications ($2 \times \log_2 x$)
  - Overall, can be done in $O(k^3)$

Apr-23 The RSA Cryptosystem 22

22

Fast exponentiation

- Square and multiply
  - Exponentiation by repeated squaring and multiplication
  - The exponentiation a^x mod n requires at most
    - $\log_2(x)$ multiplications and
    - $\log_2(x)$ squares
  - Proof
    - See next slide

Apr-23 The RSA Cryptosystem 23

23

Fast exponentiation

$$a^x \bmod n = a^{\left(x_{k-1}2^{k-1}+x_{k-2}2^{k-2}+\cdots+x_2 2^2+x_1 2+x_0\right)} \bmod n \equiv$$

$$a^{x_{k-1}2^{k-1}} a^{x_{k-2}2^{k-2}} \cdots a^{x_2 2^2} a^{x_1 2} a^{x_0} \bmod n \equiv$$

$$\left(a^{x_{k-1}2^{k-2}} a^{x_{k-2}2^{k-3}} \cdots a^{x_2 2} a^{x_1}\right)^2 a^{x_0} \bmod n \equiv$$

$$\left(\left(a^{x_{k-1}2^{k-3}} a^{x_{k-2}2^{k-4}} \cdots a^{x_2}\right)^2 a^{x_1}\right)^2 a^{x_0} \bmod n \equiv$$

...

$$\left(\left(\left(\left(a^{x_{k-1}}\right)^2 a^{x_{k-2}}\right)^2 \cdots a^{x_2}\right)^2 a^{x_1}\right)^2 a^{x_0} \bmod n$$

ALGORITHM

```
c ← 1
for (i = k-1; i >= 0; i --) {
    c ← c² mod n;
    if (xi == 1)
        c ← c × a mod n;
}
```

COMMENT

- always k square operations
- at most k multiplications
  - *equal to the number of 1 in the binary representation of x*
- Modulo reduction is performed at each round in order to keep the intermediate results small.

Apr-23 The RSA Cryptosystem 24

24

Fast exponentiation – exercise

- Compute r = a^{20}
  - x = 20 = 10100_2
  - Step 0
    - $r_0 = a^1$
  - Step 1
    - $r_1 = (a^1)^2 = a^2 = a^{[10]_2}$
  - Step 2
    - $r_2 = (r_1)^2 = a^4 = a^{[100]_2}$
    - $r_2 = r_2 \cdot a = x^5 = a^{[101]_2}$
  - Step 3
    - $r3 = (r_2)^2 = a^{10} = a^{[1010]_2}$
  - Step 4
    - $r_4 = (r_3)^2 = a^{20} = a^{[10100]_2}$

Apr-23 The RSA Cryptosystem 25

25

Fast exponentiation

- Let k = 1024
- #MUL in the Grade-School Algorithm
  - #MUL = 2^{1024} multiplications
- #Ops in the Square-and-Multiply Algorithm
  - #SQ = k
  - #MUL = #(1's in the binary representation)
    - On average #MUL = 0.5K
  - #Ops = 1.5k = 1536 multiplications
  - Each multiplication is on 1024 bits

Apr-23 The RSA Cryptosystem 26

26

RSA fast encryption with short public exponent

- RSA ops with public exponent e can be speeded-up
  - Encryption
  - Digital signature verification
- The public key e can be chosen to be a very small value
  - e = 3 #MUL + #SQ = 2
  - e = 17 #MUL + #SQ = 5
  - $e = 2^{16}+1$ #MUL + #SQ = 17
  - RSA is still secure

RSA decryption

- Assume a 2048-bit modulus and a 32-bit CPU
- Decryption computing overhead
  - On average #MUL+#SQ = 1.5 × 2048 = 3072 long multiplications each of which involves 2018-bit operands
  - Single long-number multiplication
    - Each operand requires 2048/32 = 64 registers
    - Each long-number multiplication requires 64^2 = 4096 integer multiplications
    - Modulo reduction requires 64^2 = 4096 integer multiplications
    - In total 4096 + 4096 = 8192 integer multiplications for a single long multiplication
  - In total, 3072 × 8192 = 25.165.824 integer multiplications

RSA decryption

- '70s-'80s: only hardware implementation
- Today, an RSA decryption takes ≈100 μs on high-speed hw
- End '80s, software implementation becomes possible
- Today, 2048-bit RSA takes ≈10 ms on a 2 GHz CPU
  - Throughput = 2048 × 100 = 204.800 bit/s
  - ≈ 3 orders of magnitude slower than symmetric encryption

Apr-23 The RSA Cryptosystem 29

29

RSA Fast decryption

- There is no easy way to accelerate RSA when the private exponent d is involved
  - sizeof(d) = sizeof(n) to discourage brute force attack
    - It can be shown that sizeof(d) ≥ 0.3 sizeof(n)
- One possible approach is based on the Chinese Remainder Theorem (CRT)
  - We do not prove the theorem
  - We just apply it

Apr-23 The RSA Cryptosystem 30

30

Fast RSA decryption by CRT

- Problem
  - Compute $y \equiv x^d \pmod{n}$ efficiently
- The method
  1. Transformation of the problem in the CRT domain
     1. Compute $x_p \equiv x \pmod{p}$
     2. Compute $x_q \equiv x \pmod{q}$
  2. Exponentiation in the CRT domain
     1. $y_p \equiv x_p^{d_p} \bmod p$, where $d_p \equiv d \bmod (p-1)$
     2. $y_q \equiv x_q^{d_q} \bmod q$, where $d_q \equiv d \bmod (q-1)$

Apr-23 The RSA Cryptosystem 31

31

Fast RSA decryption by CRT

- The method (cont.ed)
  3. Inverse transformation in the problem domain
     1. $y \equiv [q \cdot c_p]y_p + [p \cdot c_q]y_q \bmod n$ where
        - $c_p \equiv q^{-1} \bmod p$ and
        - $c_q \equiv p^{-1} \bmod q$

Apr-23 The RSA Cryptosystem 32

32

Fast RSA decryption by CRT

- Comments
  - With reference to step 2, as sizeof(p) = sizeof(q), d_p, d_q, y_p, y_q have about half the bit length of n
    - This leads to a speedup = 4
  - With reference to step 3, expressions in square brackets can be precomputed
    - Then, the reverse transformation requires two modular multiplications and one modular addition

Apr-23 The RSA Cryptosystem 33

33

Fast RSA decryption by CRT

- Complexity of CRT-based RSA decryption
  - Step 1 and step 3 are negligible
  - Step 2
    - Let n length is t bits, then all quantities in step 2 are on t/2 bits
    - By applying the Square-and-multiply algorithm
      - #OPS = #SQ+#MUL = 2 × (1.5 t/2) = 1.5 t
      - The #OPS is the same as without CRT, however, each operation involve t/2-bit operands instead of t-bit operand, so its time is $(t/2)^2$
      - As multiplication complexity is quadratic, **the total speed up is a factor of 4**
- The method is subject to fault-injection attack

Apr-23 The RSA Cryptosystem 34

34

The RSA Cryptosystem

RSA IN PRACTICE

RSA in practice

- Schoolbook/plain RSA is insecure
  - RSA is deterministic
    - A given pt is always mapped into a specific ct
  - PT values 0 and 1 produce CT equal to 0 and 1
  - Small exponent and small pt might be subject to attacks
  - RSA is malleable
- Padding is a solution to all these problems
  - Never use plain RSA

RSA malleability

- Malleability
  - A crypto scheme is said to be malleable if the attacker is capable of transforming the ciphertext into another ciphertext which leads to a known transformation of the plaintext
    - The attacker does not decrypt the ciphertext, but (s)he is able to manipulate the plaintext in a predictable manner

RSA Malleability

- The sender
  - Transmits $y = x^e \bmod n$
- The adversary
  - Intercepts y
  - Chooses s s.t. $\gcd(s, n) = 1$
  - Computes and forwards $y' = s^e \cdot y \bmod n$
- The receiver
  - Decrypts y', $x' = y'^d = (s^e \cdot y)^d = s^{ed} \cdot y^d = s \cdot x \bmod n$
    - The attacker manages to multiply the ct x by a factor s

RSA Padding

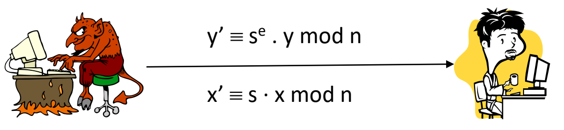
- Padding intuition
  - It embeds a random structure into the plaintext before encryption
- Padding in RSA
  - Optimal Asymmetric Encryption Padding (OAEP)
    - Specified and standardized in PKCS#1 (Public Key Cryptography Standard #1)

RSA malleability

- More in general, RSA malleability descends from the homomorphic property
  - Let x_1 and x_2 two plaintext messages
  - Let y_1 and y_2 their respective encryptions
  - Then, $y \equiv (x_1 \cdot x_2)^e \equiv {x_1}^e {x_2}^e \equiv y_1 \cdot y_2 \bmod n$
  - That is, the CT of the product is the product of the CTs

Adaptive chosen-ciphertext attack

- The problem
  - Assume that Bob decrypts any ciphertext except a given ciphertext y
  - The attacker wants to determine the plaintext corresponding to y



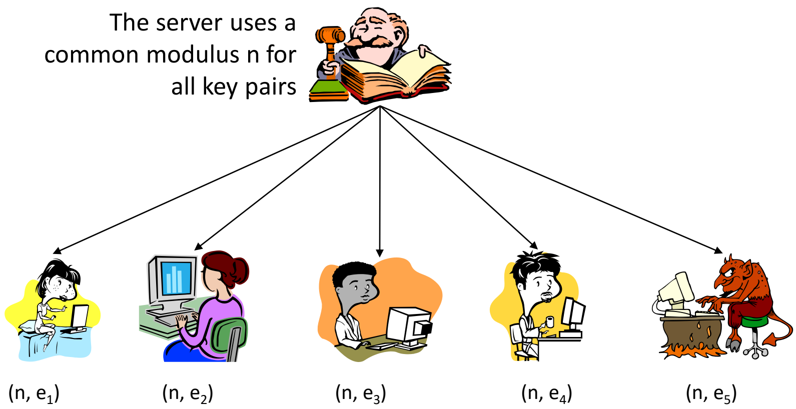
41

Adaptive chosen-ciphertext attack

- The attack
  - The adversary selects an integer s, s.t. gcd(s, n) = 1, and sends Bob the quantity $y' \equiv s^e \,.\, y \bmod n$
  - Upon receiving y', as $y' \neq y$, Bob decrypts y', producing $x' \equiv s \cdot x \bmod n$, and returns x' to the adversary
  - The adversary determines x, by computing $x \equiv x' \cdot s^{-1} \bmod n$
- Countermeasure
  - The attack can be contrasted by using padding
  - Bob returns x' iff it has a structure coherent with padding

42

Common modulus attack

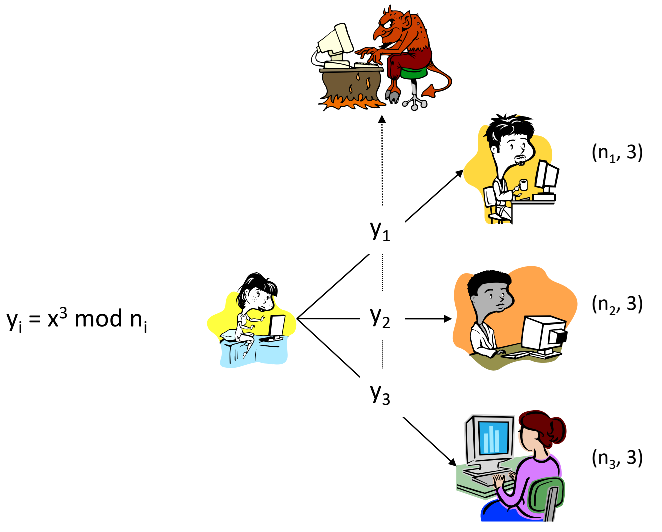


Mr Lou Cipher can efficiently factor n from d_5 (FACT 1) and then compute all d's

Small message attack

- Let x be a cleartext message, (e, n) a public key, and $y = x^e \bmod n$ a ciphertext message with with x, y $\in$ [0, n-1]
- Let x be «small» i.e. $x^e < n$. Then, $y = x^e$ and thus $x = \sqrt[e]{y}$ which is a "normal" e-th root operation that is "easy".

Low Exponent Attack



Cinese Remainder Theorem

- CHINESE REMAINDER THEOREM. If the integers n_1, $n_2, \ldots, n_k$ are pairwise relatively prime, then the system of simultaneous congruences
  - $x \equiv a_1 \pmod{n_1}$
  - $x \equiv a_2 \pmod{n_2}$
  - ...
  - $x \equiv a_k \pmod{n_k}$

  has a unique solution modulo $n = n_1 n_2 \cdots n_k$.

Cinese Remainder Theorem

- GAUSS'S ALGORITHM. The solution x to the simultaneous congruences in the Chinese remainder theorem may be computed as

$$x = \sum_{i=1}^{k} a_i N_i M_i \bmod \mathrm{n}$$

where $N_i = n/n_i \bmod n_i$ and $M_i = {N_i}^{-1} \bmod n$

- These computations can be performed in $O((\lg n)^2)$ bit operations.

Low Exponent Attack

- If n_i are pairwise coprime, use CRT to compute $z = x^3 \bmod n_1 n_2 n_3$ that solves

$$\begin{cases} z \equiv y_1 \bmod n_1 \\ z \equiv y_2 \bmod n_2 \\ z \equiv y_3 \bmod n_3 \end{cases}$$

- According to RSA encryption definition $x < n_i$ then $x^3 < n_1 n_2 n_3$ and thus $z = x^3$ ➔ x is the integer cube root of z, $x = \sqrt[3]{z}$
  - This is not a modular root ➔ it is "easy"

Low Exponent Attack

- COUNTERMEASURES
- Salting
  - A different salt for each receiver: $x \mid\mid salt_i$
- Use large exponent
  - E.g., $e = 2^{16}+1$

Apr-23 The RSA Cryptosystem 49

49

Selecting primes p and q – hints

- Primes p and q should be selected so that factoring $n = p \cdot q$ is computationally infeasible, therefore
- p and q should be sufficiently large and about the same bit length (to avoid the elliptic curve factoring algorithm)
- $p - q$ should be not too small
- $(p - 1)/2$ and $(q - 1)/2$ should be relatively prime

Apr-23 The RSA Cryptosystem 50

50

The RSA Cryptosystem

RSA SECURITY

Attacks

- Protocol attacks
- Mathematical attacks
- Side-channel attacks

Protocol attacks

- Based on malleability of RSA
- Avoidable by padding

Apr-23 The RSA Cryptosystem 53

53

Mathematical attacks

- The RSA Problem (RSAP)
  – Recovering plaintext x from ciphertext y, given the public key (n, e)
- RSA VS FACTORING
  – If p and q are known, RSAP can be easily solved
  – RSAP $\leq_P$ FACTORING
    - FACTORING is at least as difficult as RSAP or, equivalently, RSAP is not harder than FACTORING
      – It is widely believed that RSAP and Factoring are computationally equivalent, although no proof of this is known.

Apr-23 The RSA Cryptosystem 54

54

Mathematical Attacks

- THM (FACT 1) Computing the decryption exponent d from the public key (n, e) is computationally equivalent to factoring n
  - Proof
    - If factorization of n is known, then it is possible to compute the private key d efficiently
    - (It can be proven that) if d known, then it is possible to factor n efficiently

Apr-23 The RSA Cryptosystem 55

55

Mathematical Attacks

- RSAP vs e-th root
  - A possible way to decrypt y = x^e mod n is to compute the modular e-th root of y, i.e., $x = \sqrt[e]{y} \bmod n$
- THM (FACT 2) Computing the e-th root is a computationally easy problem iff n is prime
- THM (FACT 3) If n is composite the problem of computing the e-th root is equivalent to factoring

Apr-23 The RSA Cryptosystem 56

56

Mathematical Attacks

- THM - Knowing ϕ is computationally equivalent to factoring
  - PROOF.
    - Given p and q, s.t. n =pq
      - Computing ϕ is immediate.
    - Given ϕ
      - From $\phi = (p-1)(q-1) = n - (p+q) + 1$, determine $x1 = (p+q)$.
      - From $(p - q)^2 = (p + q)^2 - 4n = x_1^2 - 4n$, determine $x2 = (p - q)$.
      - Finally, $p = (x1 + x2)/2$ and $q = (x1 - x2)/2$.

Apr-23 The RSA Cryptosystem 57

57

Mathematical Attacks

- Exhaustive Private Key Search
  - This attack must be more difficult than factoring n
  - The bit length of private exponent d must be the same as the bit length of n
    - sizeof(p) ≈ sizeof(q)
    - sizeof(d) >> sizeof(p) AND sizeof(d) >> sizeof(q)

Apr-23 The RSA Cryptosystem 58

58

Factoring

- Primality testing vs. factoring
  - FACT 5 – To decide whether an integer is composite or prime seems to be, in general, much easier than the factoring problem

Factoring

- Factoring algorithms
  - Special purpose algorithms
    - Tailored to perform better when the integer n being factored is of special form
      - Running time depends on certain properties of factors of n
    - Examples
      - Trial division, Pollard's rho alg., Pollard's p – 1 alg., elliptic curve alg., and special number sieve
  - General purpose algorithms
    - Running time depends on n
    - Examples
      - Quadratic sieve and general number field sieve

Factoring

- Factoring algorithms
  - No algorithm can factor all integers in polynomial time
    - Neither the existence nor non-existence of such algorithms has been proven, but it is generally suspected that they do not exist
    - Peter Shor discovered a quantum algorithm that is polynomial (1994)
  - There are sub-exponential algorithms
    - For computers, the best algorithm is General Number Field Sieve (GNFS)

Factoring

- Length of the modulus
  - RSA sparked much interest in the old problem of integer factorization
    - Factoring methods improved considerably during ’80s and ‘90s
  - Advisable modulus length
    - Until recently, 1024-bit was a default
      - Nowadays factorization within 10-15 years or even earlier
    - Modulus in the range 2048-4096 bit for long term security

Apr-23 The RSA Cryptosystem 63

63