The RSA Cryptosystem

PROOF OF RSA

11

RSA consistency: preliminaries [→]

- **Definition** - Euler's Phi Function
  - The number of integers in $\mathbb{Z}_m$ relatively prime to m is denoted by Φ(m).
- **Theorem** - Let m have the following canonical factorization

$$m = {p_1}^{e_1} \cdot {p_2}^{e_2} \cdot \ldots \cdot {p_n}^{e_n}$$

where the p_i are distinct prime numbers and e_i are positive integers, then

$$\Phi(m) = \prod_{i=1}^{n} ({p_i}^{e_i} - {p_i}^{e_i - 1})$$

12 If you consider $\mathbb{Z}_p$, so you consider $[0, p-1]$. What are the numbers coprime with p? p-1. If you apply $\Phi(m) = p-1$
Otherwise, $m = p \cdot q$ in RSA. $\Phi(m) = (p^1 - p^0)(q^1 - q^0)$

RSA consistency: preliminaries

- Theorem - Fermat's Little Theorem
  - Let a be an integer and p be a prime, then $a^p \equiv a \pmod{p}$.
- Theorem - Euler's Theorem
  - Let a and m be integers with $\gcd(a, m) = 1$, then $a^{\Phi(m)} \equiv 1 \pmod{m}$.
- Euler's theorem is a generalization of the Fermat's little theorem

13

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: $y^d \equiv x^{ed} \equiv x^{1+t\cdot\Phi(n)} \equiv x \cdot x^{t\cdot\Phi(n)} \equiv x \cdot (x^{\Phi(n)})^t \bmod n$

14

RSA Consistency: proof

UNIVERSITÀ DI PISA

- Step 2: prove that $x \equiv x \cdot (x^{\Phi(n)})^t \bmod n$ (equivalent)
- **Recall**
  - Euler's Theorem: if gcd(x, n) = 1 then $1 \equiv x^{\Phi(n)} \bmod n$
    - Minor generalization $1 \equiv 1^t \equiv (x^{\Phi(n)})^t \bmod n$
- **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.

$(a \cdot b) \bmod m = [(a \bmod n)(b \bmod n)] \bmod m$

≡

15

RSA Consistency: proof

UNIVERSITÀ DI PISA

- **Case 2**: gcd(x, n) ≠ 1

x cannot be p·q because x < n

  - Since p and q are primes (and x < n) ➔ either x = r·p or x = s·q with r < q and s < p
  - Assume x = r·p ➔ gcd(x, q) = 1 ➔ Euler's Theorem holds in this form $1 \equiv (x^{\Phi(n)})^t \bmod q$ (by construction)
    - 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 ➔ (1 + a multiple of q for some coeff.)
    $x \cdot (x^{\Phi(n)})^t = x + x \cdot u \cdot q = x + (r \cdot p) \cdot u \cdot q = x + r \cdot u \cdot (q \cdot p) = x + r \cdot u \cdot n$
    ➔ $x \cdot (x^{\Phi(n)})^t \equiv x \bmod n$ Q.E.D.

$x(1 + u \cdot q)$, but $x = r \cdot p$

WOW!

PANKO ADMIRES

16

RSA encryption and decryption

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

17

The RSA Cryptosystem

RSA SECURITY

18

Attacks

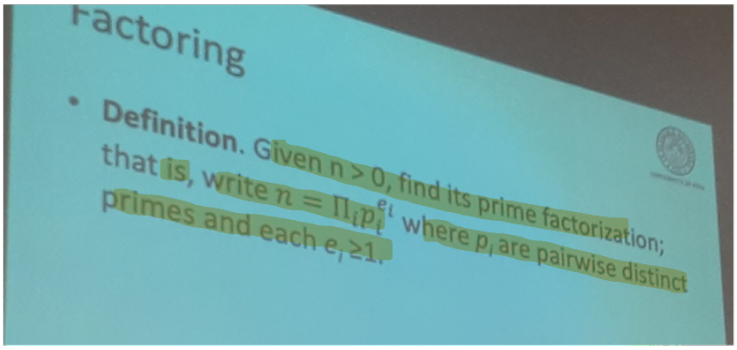
Algorithm can be attacked in several ways

- Protocol attacks
- Mathematical attacks (cryptoanalysis attacks)
- Side-channel attacks (attack the implementations)

19

Protocol attacks

- Based on malleability of RSA: bad news: RSA is malleable
- Avoidable by padding



20

Mathematical attacks

- The RSA Problem (RSAP)
  - Recovering plaintext x from ciphertext y, given the public key (n, e) Adversary can grab all ct. And knows PK.
- RSA VS FACTORING
  - If p and q are known, RSAP can be easily solved
  - RSAP $\leq_p$ FACTORING i.e., 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.

Mar-25 The RSA Cryptosystem 21

21

- For the moment nobody has proven that breaking RSA necessarily requires the ability of factoring n, although this conjecture is considered very plausible

Mathematical Attacks

Theorem: → No proof

- 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

↳ if I use one d value, I can factorize and decrypt their received messages.

Mar-25 The RSA Cryptosystem 22

22

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

23

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 ϕ = (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.

24

Mathematical Attacks

UNIVERSITÀ DI PISA

- 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)

sizeof(d) ≈ sizeof(n)

There are attacks that can exploit discrepancies

Libraries take care of these requirements

Factoring

UNIVERSITÀ DI PISA

- 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

* Primality testing

• I can say in an efficient way if number is prime or not

Factoring: there exist different families of fact. alg.

- Factoring algorithms
  - General purpose algorithms (any number)
    - Running time depends on n
    - Examples
      - Quadratic sieve and general number field sieve
  - 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

Mar-25 The RSA Cryptosystem 27

27

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 (best known general purpose, *
    - For computers, the best algorithm is General Number Field Sieve (GNFS)

* power of a function of the number of bits of key as complexity.
ex: 2^n is exponential, $\sqrt{2^n}$ or $2^{f(n)}$ are sub exponential

Mar-25 The RSA Cryptosystem 28

28

Unless a new algorithm (polinomial) or new cryptoanalytic attacks is found, RSA is secure. But we are migrating to lattice, that cannot be broken with quantum comp.

Factoring

UNIVERSITÀ DI PISA

- 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

29

The RSA Cryptosystem

RSA IN PRACTICE

30

RSA in practice

- Schoolbook/plain RSA is insecure (the one in the original paper)
  - 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

(d must be large to avoid bruteforce. We are talking about e)

e can be small for efficiency but you can be subject to attacks

Mar-25 The RSA Cryptosystem 31

31

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

Mar-25 The RSA Cryptosystem 32

32

RSA Malleability [→]

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

Mar-25 The RSA Cryptosystem 33

33

ex of bad padding: $X||X$. With the attack I get $y' = (X||X) \cdot S \bmod n$

It is very unlikely that $y' = a||a$ for some a.

This solution is ineffective btw.

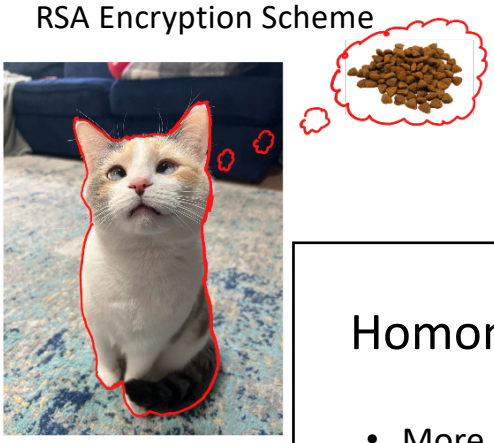
RSA Padding

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

Intuition: put redundancy in PT and you want to find it still after decryption

Mar-25 The RSA Cryptosystem 34

34



Homomorphism

UNIVERSITÀ DI PISA

- 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

Server could perform multiplication without seeing data (homomorphic cryptography).

35

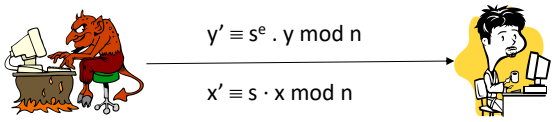
Adaptive chosen-ciphertext attack

UNIVERSITÀ DI PISA

- The problem

Since RSA is homomorphic, you can use that to your advantage

  - Assume that Bob decrypts any ciphertext except a given ciphertext y → We "force" Bob to decrypt y too.
  - The attacker wants to determine the plaintext corresponding to y



36

Adaptive chosen-ciphertext attack

- The attack
  - The adversary selects an integer s, s.t. gcd(s, n) = 1, and sends Bob the quantity $y' \equiv s^e \cdot y \bmod n$
  - 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

37

Common modulus attack

The server uses a common modulus n for all key pairs

A trusted third party

(n, e_1) (n, e_2) (n, e_3) (n, e_4) (n, e_5)

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

↳ Because he knows ϕ and e_5.

38

Modulus MUST NOT be reused

Small message attack

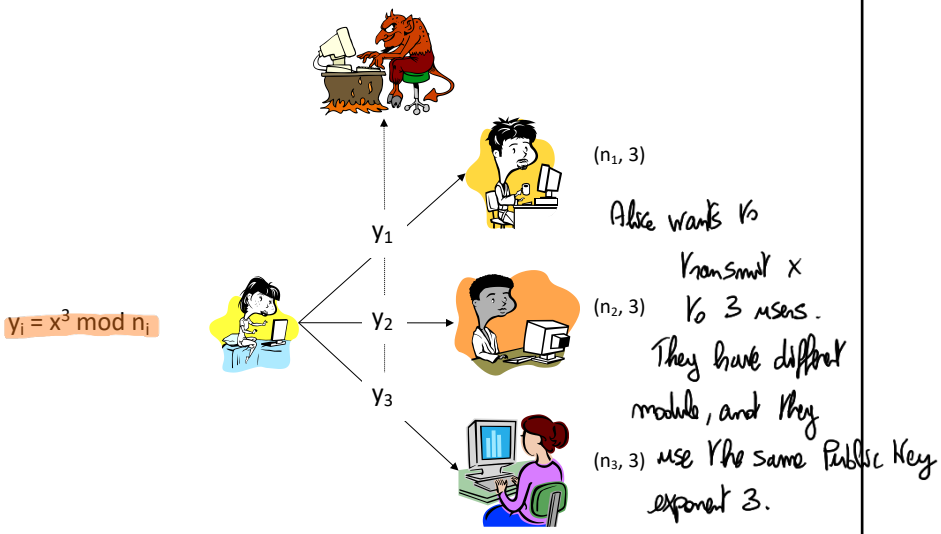
- Let x be a cleartext message, (e, n) a public key, and $y = x^e \bmod n$ a ciphertext message with with x, y ∈ [0, n-1]
- Let x be «small», i.e., $x^e < n$, then $y = x^e$ and thus $x = \sqrt[e]{y}$ which is a "normal" e-th root operation and therefore "easy".

PADDING?

Mar-25 The RSA Cryptosystem 39

39

Low Exponent Attack



Alice wants to transmit x to 3 users. They have different modulo, and they use the same Public Key exponent 3.

Mar-25 The RSA Cryptosystem 40

40

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 $n = n_1 \cdot n_2 \cdots n_k$.

41

Cinese Remainder Theorem [→]

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

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

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.
- A CRT solver can be found here: https://www.dcode.fr/chinese-remainder

42

Let n_1, ..., n_k be integers greater than 1, which are often called *moduli* or *divisors*. Let us denote by N the product of the n_i.

The Chinese remainder theorem asserts that if the n_i are pairwise coprime, and if a_1, ..., a_k are integers such that $0 \le a_i < n_i$ for every i, then there is one and only one integer x, such that $0 \le x < N$ and the remainder of the Euclidean division of x by n_i is a_i for every i.

This may be restated as follows in terms of congruences: If the n_i are pairwise coprime, and if a_1, ..., a_k are any integers, then the system

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ &\vdots \\ x &\equiv a_k \pmod{n_k}, \end{aligned}$$

has a solution, and any two solutions, say x_1 and x_2, are congruent modulo N, that is, $x_1 \equiv x_2 \pmod{N}$.[13]

$X^3 \equiv y_i \bmod a_i$

Attack works because n1 n2 n3 the same

Low Exponent Attack

UNIVERSITÀ DI PISA

gcd(m1, m2) = 1 gcd(m2, m3) = 1 gcd(m3, m1) = 1

- If n_i are pairwise coprime, use CRT to compute $z = x^3 \bmod n_1 n_2 n_3$ that solves
  - $z \equiv y_1 \bmod n_1$ → per costruzione
  - $z \equiv y_2 \bmod n_2$
  - $z \equiv y_3 \bmod n_3$
- 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"

Mar-25 The RSA Cryptosystem 43

43

NOTE: Number of CT must be the same one of the Public Key exponent

Low Exponent Attack

UNIVERSITÀ DI PISA

- COUNTERMEASURES
- Salting
  - A different salt for each receiver: x || $salt_i$
- Use large exponent
  - E.g., $e = 2^{16}+1$ Good performances with more security

Mar-25 The RSA Cryptosystem 44

44

Selecting primes p and q – hints

- Primes p and q should be selected so that factoring n = p·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

45

The RSA Cryptosystem

PERFORMANCE

46

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

47

Computation of *e* and *d* (refined)

1. Select e ∈ (1, φ(n)) — extended euclide algorithm
2. Apply EEA with input parameters n and e (IL PROCESSO CONVERGE)
   - gcd(Φ(n), e) = s·φ(n) + t·e (Diophantine equation)
     - If gcd(e, φ(n)) = 1 then [verify if they are coprime] Φ(n) is even
       - 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)) ≠ 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 (≈ logarithmic)

48

We select e, choose p and q and the execute this algorithm hoping that value is 1. We try t and s to get gcd. If we get 1 good otherwise I try something else.

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] ≈ 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
- True primality test are more computationally intensive than probabilistic ones
  - Apply true tests to a candidate that passes through probabilistic tests

There are polinomial deterministic tests but the exponent of polinomial is large (≈ 12).

Or I can test value by means of encryption and decryption to see if algorythm works.

THE COMP. OF FINDING e and d and the primes is efficient.
What about exponentiation?

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 × 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) Extended euclidean algorythm (logarythmic)
    - Modular exponentiation can be done in $O(k^3)$

↳ Polinomial, which is good. But still K^3.

Fast exponentiation

- How many multiplications to compute 2^{20}?
- Grade-school Algorithm requires
  - 2 x 2 x 2 x … x 2 → 19 multiplications Not very efficient
- Square-and-Multiply Algorithm [1 MUL + 4 SQ]
  - $((2 \times (2^2)^2)^2)^2$ → 1 multiplications + 4 squares → 5 multiplications

↓ Requires less multiplications

More efficient!

Mar-25 The RSA Cryptosystem 53

53

Fast exponentiation

- RSA computes modular exponentiation
  - $a^x \bmod n$, where n is on k bits (i.e., $n \leq 2^k$)* X is equal to e in encryption, otherwise d, that can be very large
- 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

↳ d

- Square-and-multiply Algorithm
  - requires up to 2k multiplications ($2 \times \log_2 x$) We can prove this
  - Overall, can be done in $O(k^3)$

↓ K is number of bits to represent X. This is linear in K.

Mar-25 The RSA Cryptosystem 54

54

* To discourage brute force, d as large as possible
If n is on K bits, d must be as well

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

55

Fast exponentiation

binary rep. of x

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

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

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

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

...

We repeat iteratively until we arrive here

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

must be 1 because otherwise x would'nt be representable on k bits but k-1.

ALGORITHM

```
c ← 1
for (i = k-1; i >= 0; i --) {
    c ← c² mod n;
    if (x_i == 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.

56 (1) Algebraic transformation through property of powers and short collecting

(2) I have as many square as the # of bits and as many mult. as 1s in x.

If x_2 is 0, $a^{x_2} = 1$. So to move to next step I don't actually need to multiply.
The order of square op. is K.

So # of ops to perform is linear in the size of the problem.

Fast exponentiation – exercise

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

57

Fast exponentiation

- Let k = 1024 Modulus is 1024 bits
- #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 (large multiplication! That's why PKE is slower than SKE.

58

→ In reality # of multiplication is larger

Each multip. has 1024 bits operands. Each multiplication is K^2. Size of operands is large

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 and RSA is still secure
  - Values
    - e = 3 — #MUL + #SQ = 2 (commonly used in practice)
    - e = 17 — #MUL + #SQ = 5
    - e = $2^{16}+1$ — #MUL + #SQ = 17 (avoid small exp attack)
  - gcd(e, Φ(n)) = 1 must hold.

only have 2 ones in their binary representation, so only 2 multiplications

1 square and 1 multiplication
1 multiplication

Mar-25 The RSA Cryptosystem 59

59 We optimise e choose to reduce number of multiplications. Why not only 1 but to 1? Either e=1 or 1 is the MSB. 1st case I cannot encrypt, 2nd case gcd(e, ϕ(n)) ≠ 1, because ϕ(n)=(p-1)(q-1), two even numbers. (1)

RSA decryption

encryption

- Assume a 2048-bit modulus and a 32-bit CPU and determine 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

Mar-25 The RSA Cryptosystem 60

60

(1)

Remember, we have to compute $e \cdot d \equiv 1 \bmod \Phi$. Can't I choose this rule for d and get e? So that I speed up decryption and not encryption. This makes bruteforce easier because key size shrinks (n bits ↓ - which 2 of them are 1, the rest 0).

(2) ↑

(2) So how to speed up decryption? Limited speed up possible

↓

RSA ~~decryption~~ encryption

- '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

61

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

62

Fast RSA decryption by CRT [→]

UNIVERSITÀ DI PISA

- 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 $x_q \equiv x \pmod q$

     I can compute x_p and x_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 $d_q \equiv d \bmod (q-1)$

     We won't prove these steps

We move from RSA domain to CRT domain. It is provable that this is possible. I compute y_p and y_q first.

63

Fast RSA decryption by CRT [→]

UNIVERSITÀ DI PISA

- The method (cont.ed)

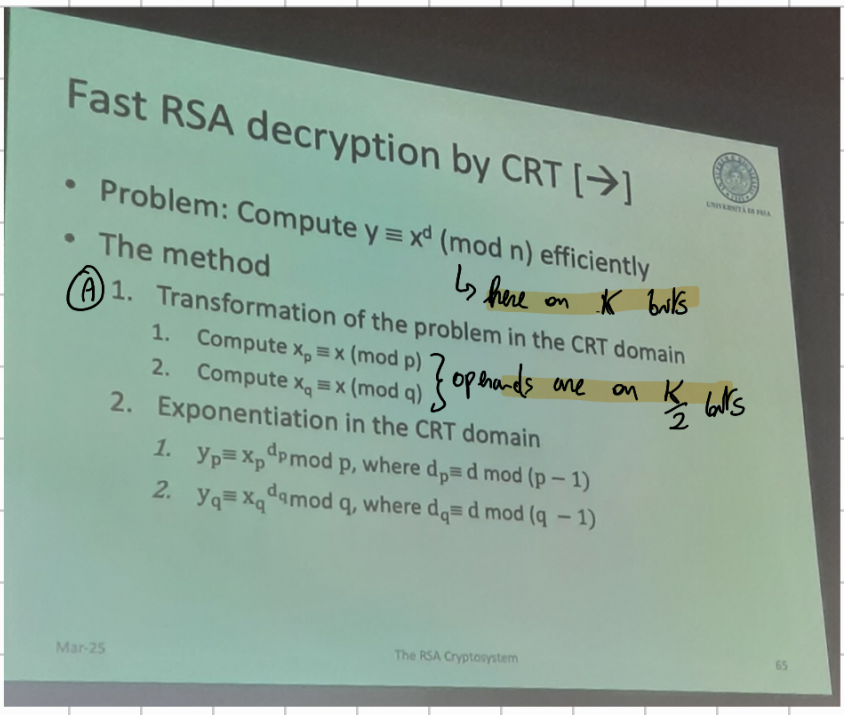
(B) 3. Inverse transformation in the problem domain

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
   - $c_q \equiv p^{-1} \bmod q$

c_p and c_q are fixed and can be precomputed

y_p and y_q depend on current iteration.

64



↳ here on K bits

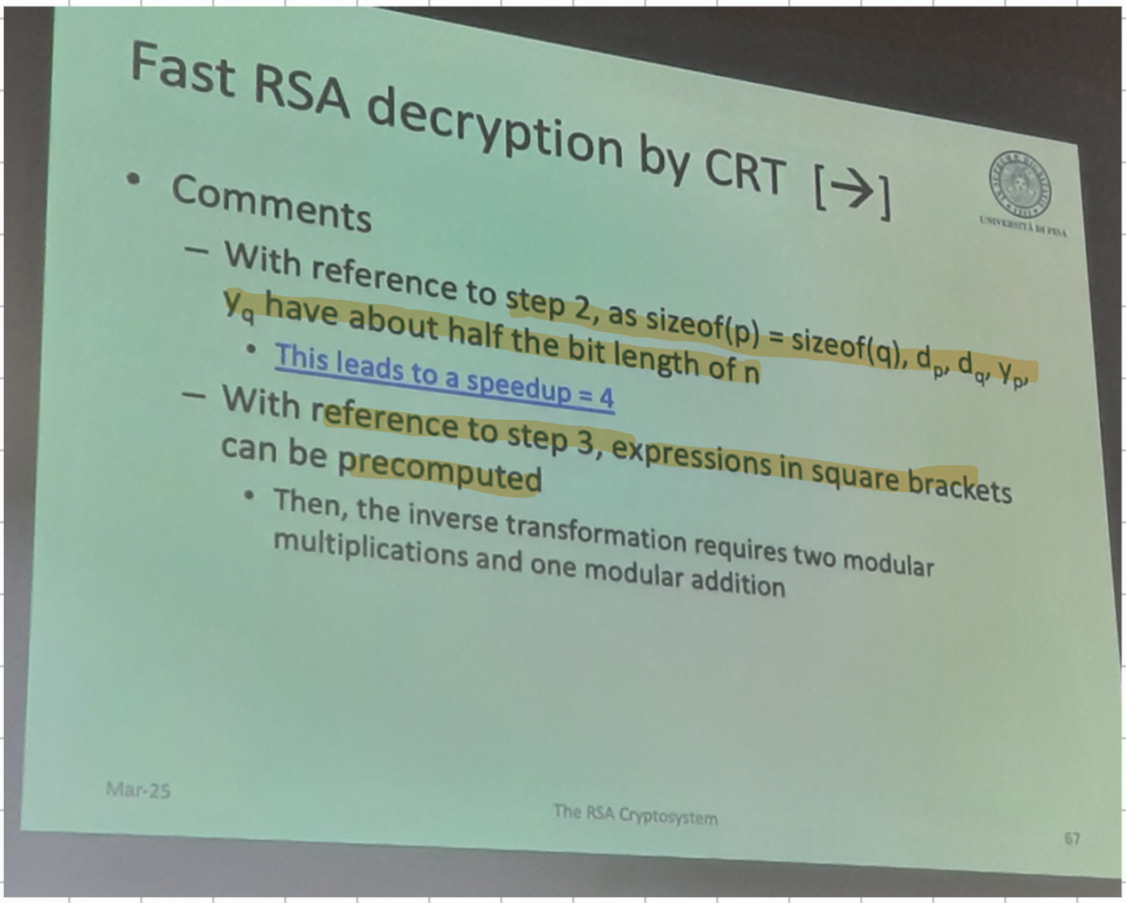
} operands are on $\frac{K}{2}$ bits

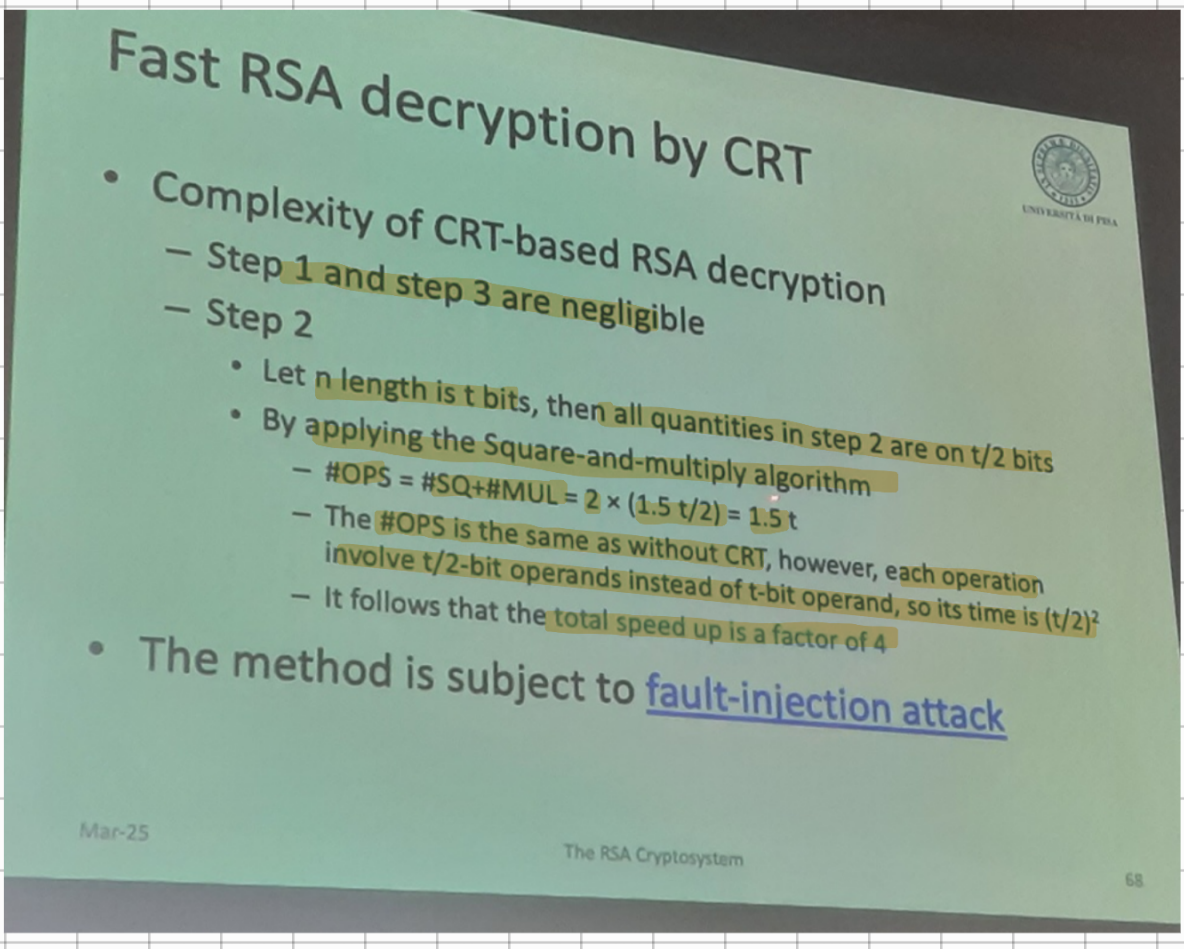
Working on K bits: mod n, but if I work on mod p and mod q I have K/2 bits.

(A) Step 1: negligible

(B) Just like step 3.

Vs. step 2





In terms of # of OPS no difference, but operands are t/2 bits. And each multiplication is $O(k^2)$. So you save time by a factor of 4.

PRICE TO PAY: Maintain p and q around. They cannot be deleted but if adversary finds them cypher is broken.

WEAK TO FAULT INJECTION ATTACK: I attack the implementation. Imagine a smartcard making the computation. I can cause a mistake through electric / magnetic fields to obtain p and n. Optimization. But not free.

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 inverse transformation requires two modular multiplications and one modular addition

65

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 $(t/2)^2$
      - It follows that the total speed up is a factor of 4
- The method is subject to fault-injection attack

66