

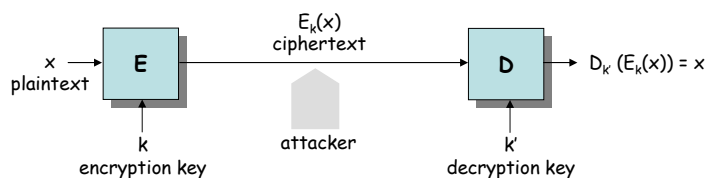
"The obvious mathematical breakthrough would be development of an easy way to factor large prime numbers."

-- Bill Gates, The Road Ahead, page 265

Public-key encryption

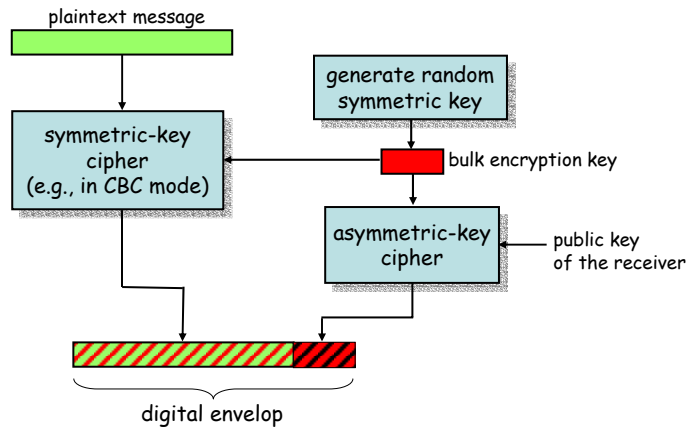
- general principles
- RSA cryptosystem
 - operation
 - relation to factoring
 - properties of the textbook RSA
 - PKCS#1
- ElGamal cryptosystem

Reminder



- asymmetric-key encryption
 - it is hard (computationally infeasible) to compute k' from k
 - k can be made public (public-key cryptography)
- public-keys are not confidential but they must be authentic !
- most popular public-key encryption methods are several orders of magnitude slower than the best known symmetric key schemes

Digital enveloping



Brief reminder on Complexity Theory

- class of all problems can be divided into two basic subclasses:
 - undecidable problems (e.g., Hilbert's tenth problem)
 - decidable problems
 - there exist algorithms that solve them
- algorithms can be classified based on their complexity
 - various complexity measures exist
 - number of basic operations performed (machine independent → commonly used)
 - execution time
 - amount of memory used
 - amount of hardware needed (e.g., number of gates)
 - complexity is usually expressed as a function of the input size
 - e.g., the complexity of multiplying two $n \times n$ matrices is n^3
 - often, what we are interested in is the asymptotic behavior of the complexity as $n \rightarrow \infty$

Brief reminder on Complexity Theory

- average case vs. worst case behavior of an algorithm
 - let D_n be the set of all input instances of length n
 - let $I \in D_n$ and let $P(I)$ be the probability that I occurs
 - let $C(I)$ be the complexity of the algorithm on input instance I
 - average case complexity:

$$\sum_{\text{for all } I \in D_n} P(I) C(I)$$
 - worst case complexity:

$$\max_{I \in D_n} C(I)$$
- complexity of a problem
 - true complexity of a problem is the complexity of the most efficient algorithm that solves the problem
 - true complexity of many problems is not known
 - believed complexity of a problem is the complexity of the best known algorithm that solves the problem

Brief reminder on Complexity Theory

- two important complexity classes:
 - class P:
 - problems solvable with an algorithm that is deterministic and p-time bounded
 - asymptotic worst case complexity is a polynomial function of the input length n
 - class NP:
 - problems solvable with an algorithm that is non-deterministic and run in p-time on a non-deterministic machine
 - problems in NP have no known deterministic p-time algorithms
 - asymptotic worst case complexity of the most efficient algorithms known is often an exponential function of the input length n
 - however, a solution to an NP problem can be verified in p-time on a deterministic machine
- it is conjectured that $P \neq NP$, but it has not been proven yet

Brief reminder on Complexity Theory

- NP-complete problems
 - a subset of NP problems such that all problems in NP reduces to them
 - if there is a p-time deterministic algorithm for an NP-complete problem, then there is a p-time deterministic algorithm for all NP problems (i.e., $P = NP$)
 - the hardest problems in NP

Examples

- factoring problem
 - given a positive integer n , find its prime factors
 - true complexity is unknown
 - it is believed that it does not belong to P
- discrete logarithm problem
 - given a prime p , a generator g of Z_p^* , and an element y in Z_p^* , find the integer x , $0 \leq x \leq p-2$, such that $g^x \bmod p = y$
 - true complexity is unknown
 - it is believed that it does not belong to P
- Diffie-Hellman problem
 - given a prime p , a generator g of Z_p^* , and elements $g^x \bmod p$ and $g^y \bmod p$, find $g^{xy} \bmod p$
 - true complexity is unknown
 - it is believed that it does not belong to P

RSA (Rivest-Shamir-Adleman) cryptosystem

- key generation
 - select p, q large primes (about 500 bits each)
 - $n = pq, \phi(n) = (p-1)(q-1)$
 - select e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$
 - compute d such that $ed \bmod \phi(n) = 1$ (this is easy if $\phi(n)$ is known)
 - the public key is (e, n)
 - the private key is d
- encryption
 - represent the message as an integer m in $[0, n-1]$
 - compute $c = m^e \bmod n$
- decryption
 - compute $m = c^d \bmod n$

Proof of RSA decryption

- $c^d \bmod n = m^{ed} \bmod n = m^{k \phi(n) + 1} \bmod n = m m^{k(p-1)(q-1)} \bmod n$
- since $m < n$, it is enough to prove that $m m^{k(p-1)(q-1)} \equiv m \pmod{n}$
- Fermat theorem
 - if r is a prime and $\gcd(a, r) = 1$, then $a^{r-1} \equiv 1 \pmod{r}$
- if $\gcd(m, p) = 1$
 - $m^{p-1} \equiv 1 \pmod{p}$
 - $m m^{k(p-1)(q-1)} \equiv m \pmod{p}$
- if $\gcd(m, p) = p$
 - $p \mid m$
 - $m m^{k(p-1)(q-1)} \equiv m \equiv 0 \pmod{p}$
- for all m , $m m^{k(p-1)(q-1)} \equiv m \pmod{p}$
- similarly, for all m , $m m^{k(p-1)(q-1)} \equiv m \pmod{q}$
- $p, q \mid m m^{k(p-1)(q-1)} - m$
- $m m^{k(p-1)(q-1)} \equiv m \pmod{pq}$

Euclidean algorithm

- given two integers a and b ($a > b$), we want to compute their gcd
- perform the following sequence of (modular) divisions:

$$a = q_1b + r_2 \quad (0 < r_2 < b)$$

$$b = q_2r_2 + r_3 \quad (0 < r_3 < r_2)$$

$$r_2 = q_3r_3 + r_4 \quad (0 < r_4 < r_3)$$

$$\dots$$

$$r_{k-2} = q_{k-1}r_{k-1} + r_k \quad (0 < r_k < r_{k-1})$$

$$r_{k-1} = q_k r_k$$
- then we have

$$\gcd(a, b) = \gcd(b, r_2) = \gcd(r_2, r_3) = \dots = \gcd(r_{k-1}, r_k) = r_k$$
- example: $\gcd(76, 28) = ?$

$$76 = 2 \times 28 + 20$$

$$28 = 1 \times 20 + 8$$

$$20 = 2 \times 8 + 4$$

$$8 = 2 \times 4 \rightarrow \gcd(76, 28) = 4$$

Extended Euclidean algorithm

- the Euclidean algorithm can be used to determine if b has an inverse mod a ($\gcd(a, b) = 1$?)
- but it does not give us the inverse of b
- extend the algorithm as follows:

$$\begin{array}{ll}
 a = q_1b + r_2 & t_0 = 0, t_1 = 1 \\
 b = q_2r_2 + r_3 & t_2 = t_0 - q_1t_1 \pmod{a} \\
 r_2 = q_3r_3 + r_4 & t_3 = t_1 - q_2t_2 \pmod{a} \\
 \dots & t_4 = t_2 - q_3t_3 \pmod{a} \\
 r_{k-2} = q_{k-1}r_{k-1} + r_k & \dots \\
 r_{k-1} = q_k r_k & t_k = t_{k-2} - q_{k-1}t_{k-1} \pmod{a}
 \end{array}$$

Extended Euclidean algorithm

Theorem: $r_j \equiv t_j b \pmod{a}$

Proof:

- convention: $r_0 = a, r_1 = b$
- $a = r_0 \equiv t_0 b = 0 \pmod{a}$
- $b = r_1 \equiv t_1 b = b \pmod{a}$
- let's assume that $r_{j-1} \equiv t_{j-1} b \pmod{a}$ and $r_{j-2} \equiv t_{j-2} b \pmod{a}$
- $r_j = r_{j-2} - q_{j-1} r_{j-1} \equiv$
 $t_{j-2} b - q_{j-1} t_{j-1} b \equiv$
 $(t_{j-2} - q_{j-1} t_{j-1}) b \equiv$
 $t_j b \pmod{a}$

Corollary: if $\gcd(a, b) = r_k = 1$, then $t_k b \equiv 1 \pmod{a}$, and therefore t_k is the inverse of $b \pmod{a}$.

Extended Euclidean Algorithm

- example: $28^{-1} = ? \pmod{75}$

$$75 = 2 \times 28 + 19$$

$$28 = 1 \times 19 + 9$$

$$19 = 2 \times 9 + 1$$

$$9 = 9 \times 1$$

$$t_2 = 0 - 2 \times 1 \pmod{75} = 73$$

$$t_3 = 1 - 1 \times 73 \pmod{75} = 3$$

$$t_4 = 73 - 2 \times 3 \pmod{75} = 67$$

$$\rightarrow \gcd(75, 28) = 1$$

$$\rightarrow 28^{-1} \pmod{75} = 67$$

Implementing RSA - Computing d

- d can be computed using the extended Euclidean algorithm
- complexity:
 - let k be the length of n in bits ($k = \lceil \log_2 n \rceil + 1$)
 - adding two k-bit integers: $O(k)$
 - multiplication of two k-bit integers: $O(k^2)$
 - reduction modulo n of a 2k-bit integer: $O(k^2)$
 - modular multiplication of two k-bit integers: $O(k^2)$
 - complexity of each step of the Euclidean algorithm: $O(k^2)$
 - number of iterations in the Euclidean algorithm: $O(k)$
 - complexity of computing d: $O(k^3)$

Implementing RSA - Modular exponentiation

- naïve approach:
 - $m^x \bmod n = m \cdot m \cdot m \dots m \bmod n$
 - complexity of x-1 modular multiplication is $O(xk^2)$
 - unfortunately x can be as big as $\phi(n)-1$, hence $x \sim O(n) = O(2^k)$
 - complexity of the naïve approach is $O(2^k)$

Implementing RSA - Modular exponentiation

- there's a better method for modular exponentiation
 - $x = b_{k-1}2^{k-1} + b_{k-2}2^{k-2} + \dots + b_12 + b_0$
 - $m^x = m^{b_0}(m^{x_1})^2$ where $x_1 = (x-b_0)/2 = b_{k-1}2^{k-2} + b_{k-2}2^{k-3} + \dots + b_1$
 - $m^{x_1} = m^{b_1}(m^{x_2})^2$ where $x_2 = (x_1-b_1)/2 = b_{k-1}2^{k-3} + b_{k-2}2^{k-4} + \dots + b_2$
 - ...
 - $m^{x_{k-3}} = m^{b_{k-3}}(m^{x_{k-2}})^2$ where $x_{k-2} = (x_{k-3}-b_{k-3})/2 = b_{k-1}2 + b_{k-2}$
 - $m^{x_{k-2}} = m^{b_{k-2}}(m^{x_{k-1}})^2$ where $x_{k-1} = (x_{k-2}-b_{k-2})/2 = b_{k-1}$
 - $m^{x_{k-1}} = m^{b_{k-1}}$
- "square and multiply" algorithm


```

c = 1
for i = k-1 to 0 do
    c = c2 mod n
    if bi = 1 then c = c·m mod n
end for
output c = mx mod n
            
```
- complexity:
 - k modular squaring (multiplication)
 - at most k modular multiplication
 - complexity of the clever approach is $O(k \cdot k^2) = O(k^3)$

© Levente Buttyán

17

RSA toy example

- key generation
 - let $p = 73, q = 151$
 - $n = 73 \cdot 151 = 11023$
 - $\phi(n) = 72 \cdot 150 = 10800$
 - let $e = 11$
 - compute d with the extended Euclidean algorithm as follows:

$10800 = 981 \times 11 + 9$	$t_2 = 0 - 981 \times 1 \bmod 10800 = 9819$	
$11 = 1 \times 9 + 2$	$t_3 = 1 - 1 \times 9819 \bmod 10800 = 982$	
$9 = 4 \times 2 + 1$	$t_4 = 9819 - 4 \times 982 = 5891$	$\rightarrow d = 5891$
 - public key is $(11, 11023)$, private key is 5891
- encryption
 - let $m = 17$
 - we compute c with the "square and multiply" algorithm as follows:

$e = 11 = 1011$ (in binary)	
$c = 1$	
$b_3 = 1 \rightarrow c = c^2 m \bmod n = 17$	
$b_2 = 0 \rightarrow c = c^2 \bmod n = 289$	
$b_1 = 1 \rightarrow c = c^2 m \bmod n = 1419857 \bmod 11023 = 8913$	
$b_0 = 1 \rightarrow c = c^2 m \bmod n = \dots = 1782$	
output $c = 17^{11} \bmod 11023 = 1782$	
- decryption
 - $d = 5891 = 1011100000011$ (in binary)
 - we compute $m = c^d \bmod n$ with the "square and multiply" algorithm as above

© Levente Buttyán

18

Implementing RSA - Primality testing

- what is the probability of the event that a randomly selected large integer is prime?
 - prime number theorem:
number of primes smaller than n is approximately $\Pi(n) \sim n/\ln(n)$
 - corollary:
probability that a randomly selected k -bit long integer is prime is

$$\frac{\Pi(2^k) - \Pi(2^{k-1})}{2^k - 2^{k-1}} \sim \frac{1}{(k-1)\ln(2)}$$
 - example:
 $k = 512$, probability is $1/354 = 0.0028$
if we consider only randomly selected odd integers, then the probability is $1/177$
- how can we know if a given integer is prime or not?
 - PRIME is in P (there is a polynomial time deterministic decision algorithm)
 - in practice, people use probabilistic primality testing algorithms

Implementing RSA - Fermat-test

- Fermat theorem:
if p prime and $\gcd(b, p) = 1$, then $b^{p-1} \equiv 1 \pmod{p}$
- a composite number n is pseudo-prime for a base b if

$$b^{n-1} \equiv 1 \pmod{n}$$
 where $1 < b < n$ and $\gcd(b, n) = 1$
- testing approach
 - choose a random base b , and check if $b^{n-1} \equiv 1 \pmod{n}$ holds
 - if not, then n is composite
 - if yes, then n may be prime and we need to test it further with other bases
 - if n passes the test for many bases, then we accept it as a prime
 - this is a Monte Carlo algorithm
 - the algorithm always gives an answer
 - the answer may be wrong with some probability ϵ
- what is the probability of a false answer?

Implementing RSA - Fermat-test

- bad news:
 - there exist composite numbers that always pass the Fermat-test (for every possible base)
 - these are called Carmichael-numbers, and they are quite rare
 - example: 561
- good news:
 - if n is composite and not a Carmichael number, then n passes the test for at most half of the possible bases
 - if we run T tests, and n passes all of them, then the probability of error is upper bounded by 2^{-T}
 - error probability can be made arbitrarily low

Implementing RSA - Fermat-test

- if n passes the test for base b , then it passes the test for base b^{-1} :

$$(b^{-1})^{n-1} = (b^{n-1})^{-1} = 1^{-1} = 1 \pmod{n}$$
- if n passes the test for bases b_1 and b_2 , then it passes it for $b_1 b_2$ too:

$$(b_1 b_2)^{n-1} = b_1^{n-1} b_2^{n-1} = 1 \cdot 1 = 1 \pmod{n}$$
- let $B = \{b_1, b_2, \dots, b_s\}$ be the set of bases for which n passes the test
- let b' be a base for which n doesn't pass the test (such b' exists because n is not Carmichael number)
- consider $b'B = \{b'b_1 \pmod{n}, b'b_2 \pmod{n}, \dots, b'b_s \pmod{n}\}$
 - n cannot pass the test for $b'b_i \pmod{n}$, since otherwise it would pass it for $b'b_i b_i^{-1} \pmod{n} = b'$
 - all $b'b_i \pmod{n}$ are different, since otherwise
 - if $b'b_i \pmod{n} = b'b_j \pmod{n}$, then $n \mid b'(b_i - b_j)$
 - $\gcd(b', n) = 1$, thus, $n \mid (b_i - b_j)$
 - this is possible only if $b_i = b_j$, since $b_i < n$ and $b_j < n$
- n does not pass the test for at least as many bases as it passes

Relation to factoring

- the problem of computing d from (e, n) is computationally equivalent to the problem of factoring n
 - if one can factor n , then he can easily compute d
 - if one can compute d , then he can efficiently factor n
- the problem of computing m from c and (e, n) (RSA problem) is believed to be computationally equivalent to factoring
 - if one can factor n , then he can easily compute m from c and (e, n)
 - there's no formal proof for the other direction
- given the latest progress in developing algorithms for factoring, the size of the modulus should at least be 1024 bits

Chinese remainder theorem

- let m_1, m_2, \dots, m_r be pairwise relatively prime positive integers
- consider the following set of congruences:

$$x \equiv a_1 \pmod{n_1}$$

$$x \equiv a_2 \pmod{n_2}$$

$$\dots$$

$$x \equiv a_r \pmod{n_r}$$
- there's a unique solution for x modulo $N = n_1 n_2 \dots n_r$:

$$x = a_1 N_1 y_1 + a_2 N_2 y_2 + \dots + a_r N_r y_r \pmod{N}$$
 where $N_i = N/n_i$ and $y_i = N_i^{-1} \pmod{n_i}$
- it is easy to verify that $a_1 N_1 y_1 + a_2 N_2 y_2 + \dots + a_r N_r y_r \equiv a_j \pmod{n_j}$
 - if $i \neq j$, then $n_j \mid a_i N_i y_i = a_i n_1 \dots n_j \dots n_r y_i$
 - if $i = j$, then $a_i N_i y_i = a_i N_i N_i^{-1} \pmod{n_i} = a_i \pmod{n_i}$
- uniqueness (mod N):
 - assume that there are two solutions x and x'
 - $n_1, n_2, \dots, n_r \mid x - x' \rightarrow N \mid x - x'$
 - since $-N < x - x' < N$, it follows that $x = x'$

Factoring n

if one can compute d from (e, n) , then he can efficiently factor n

- approach
 - let A be the algorithm that computes d from (e, n)
 - we construct another algorithm B that uses A as a subroutine, and factors n
 - B will be a Las Vegas algorithm
 - the algorithm may fail to give an answer (factor n) with probability ε
 - however, if it gives an answer then the answer is correct
 - such an algorithm should be run several times until it finds an answer
 - the probability that the algorithm fails m consecutive times is ε^m , and thus, can be arbitrarily small as m grows
 - the average number of times it needs to be run to find an answer is $1/(1-\varepsilon)$

Square roots of 1 modulo $n=pq$

- $x^2 \equiv 1 \pmod{p}$ has two solutions $x \equiv \pm 1 \pmod{p}$
- $x^2 \equiv 1 \pmod{pq}$ if and only if $x^2 \equiv 1 \pmod{p}$ and $x^2 \equiv 1 \pmod{q}$
- this means that $x \equiv \pm 1 \pmod{p}$ and $x \equiv \pm 1 \pmod{q}$
- there are four square roots of 1 (mod pq) and they can be found with the Chinese remainder theorem (if p and q are known)
 - for instance solving
 - $x \equiv 1 \pmod{p}$
 - $x \equiv 1 \pmod{q}$
 gives one of the square roots
 - two out of the four square roots are trivial: $x = 1$ and $x = -1$
 - the other two are non-trivial
 - example:
 - $n = 13 \times 31 = 403$
 - square roots of 1 (mod 403) are 1, 92, 311 = -92, 402 = -1
- if x is a non-trivial square root, then $pq \mid x^2 - 1 = (x-1)(x+1)$, but pq does not divide $(x-1)$ and $(x+1)$
- this is only possible if $p \mid x-1$ and $q \mid x+1$, or vice versa
- thus, $\gcd(x+1, pq) = q$ (or p)
- given a non-trivial square root of 1 (mod pq), one can use the Euclidean algorithm to find p and q !!!

Factoring algorithm B

1. choose w at random ($0 < w < n$)
2. compute $x = \text{gcd}(w, n)$
3. **if** $x > 1$ **then stop** (success: $x = p$ or $x = q$)
4. compute $d = A(e, n)$
5. write $ed - 1 = 2^s r$, where r is odd
6. compute $v = w^r \bmod n$
7. **if** $v \equiv 1 \pmod{n}$ **then stop** (failure)
8. **while** $v \not\equiv 1 \pmod{n}$ **do**
9. $t = v$
10. $v = v^2 \bmod n$
11. **end while**
12. **if** $t \equiv -1 \pmod{n}$ **then stop** (failure: t is a trivial root)
13. **else**
14. compute $x = \text{gcd}(t+1, n)$
15. **stop** (success: $x = p$ or $x = q$)

Analysis of algorithm B

- choose a random w ($w < n$) [step 1]
- if you are lucky, then w divides n , and thus, it is equal to p or q [steps 2 and 3]
- otherwise, the algorithm computes $w^r, w^{2r}, w^{4r}, \dots$ [step 10 within the while loop]
- the computation stops, when $w^{2^z r} \equiv 1 \pmod{n}$ for some z [condition in step 8]
 - since $w^{2^s r} = w^{ed-1} = w^{k\phi(n)} \equiv 1 \pmod{n}$, the while loop ends after at most s iterations
- after the while loop, $t^2 \equiv 1 \pmod{n}$ and we know that $t \not\equiv 1 \pmod{n}$, since otherwise the while loop would have been ended in the previous round (and we wouldn't have computed t^2)
- if $t \equiv -1$ then t is a trivial square root of $1 \pmod{n}$ [step 12]
- otherwise t is a non-trivial square root of $1 \pmod{n}$ and we can factor n with the Euclidean algorithm [step 14]
- it can be proven that the failure probability of the algorithm is at most $\frac{1}{2}$

Unconcealed messages

- a message is unconcealed if it encrypts to itself (i.e., if $m^e \bmod n = m$)
- trivial examples for unconcealed messages are $m = 0$, $m = 1$, and $m = n-1$
- the exact number of unconcealed messages is $(1 + \gcd(e-1, p-1))(1 + \gcd(e-1, q-1))$
 - if p , q , and e are selected at random (or e is small such as $e = 3$), then the number of unconcealed messages is negligibly small

Small encryption exponent e

- to improve efficiency of encryption, it is desirable to select a small exponent e (e.g., $e = 3$ is typical)
- a group of entities may use the same exponent, but different moduli (e.g., $e = 3$, and n_1, n_2, \dots)
- in this case, an attacker may find a plaintext m efficiently, if m is sent to several (at least 3) recipients:
 - assume that the attacker observes $c_i = m^3 \bmod n_i$ ($i = 1, 2, 3$)
 - let $x = m^3$
 - the attacker must solve for x the following system of congruences:
 - $x \equiv c_1 \pmod{n_1}$
 - $x \equiv c_2 \pmod{n_2}$
 - $x \equiv c_3 \pmod{n_3}$
 - Chinese remainder theorem: if n_1, n_2, \dots, n_k are pairwise relatively primes, then such a system has a unique solution $(\bmod n_1 \cdot n_2 \cdot \dots \cdot n_k)$
 - since $m^3 < n_1 \cdot n_2 \cdot n_3$ the solution found must be m^3
 - the attacker then computes the cube root of m^3 to get m

Salting

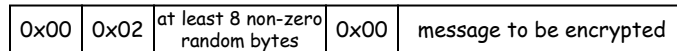
- appending a (pseudo) random bit string to the plaintext prior to encryption
- salting is a solution to the small exponent problem
 - even if the same message m has to be sent to many recipients, the actual plaintext that is encrypted will be different for everyone due to salting
- another problem of small exponents where salting helps
 - if $m < n^{1/e}$, then $m^e < n$, and hence $c = m^e$
 - m can be computed from c by taking the e^{th} root of c
 - salting helps, because it increases the plaintext so that it becomes larger than $n^{1/e}$
- it is also good for preventing forward search attacks
 - if the message space is small and predictable, then an attacker can pre-compute a dictionary by encrypting all possible plaintexts
 - salting increases the number of possible plaintexts and makes pre-computing a dictionary harder

Homomorphic property

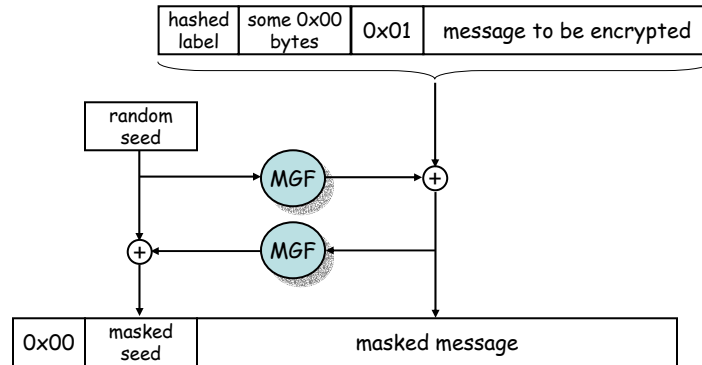
- if m_1 and m_2 are two plaintext messages and c_1 and c_2 are the corresponding ciphertexts, then the encryption of $m_1 m_2 \bmod n$ is $c_1 c_2 \bmod n$
 - $(m_1 m_2)^e \equiv m_1^e m_2^e \equiv c_1 c_2 \pmod{n}$
- this leads to an adaptive chosen-ciphertext attack on RSA
 - assume that the attacker wants to decrypt $c = m^e \bmod n$ intended for Alice
 - assume that Alice will decrypt arbitrary ciphertext for the attacker, except c
 - the attacker can select a random number r and submit $c \cdot r^e \bmod n$ to Alice for decryption
 - since $(c \cdot r^e)^d \equiv c^d \cdot r^{ed} \equiv m \cdot r \pmod{n}$, the attacker will obtain $m \cdot r \bmod n$
 - he then computes m by multiplication with $r^{-1} \pmod{n}$
- this attack can be circumvented by imposing some structural constraints on plaintext messages
 - e.g., a plaintext must start with a well-known constant bit string
 - since r is random, $m \cdot r \pmod{n}$ will not have the right structure with very high probability, and Alice can refuse to respond

RSA encryption in practice: PKCS #1

- PKCS1 v1.5 encoding



- PKCS1 v2.0 encoding



© Levente Buttyán

33

Bleichenbacher's attack on PKCS1 v1.5

- adaptive chosen ciphertext attack
- the goal is to decrypt a message with the help of an oracle that
 - inputs an arbitrary message
 - decrypts it
 - verifies PKCS formatting
 - responds with 1 if the obtained plaintext is PKCS conform, and 0 otherwise
- the attack needs $\sim 2^{20}$ oracle call only
- details can be found in the handwritten notes

© Levente Buttyán

34

ElGamal cryptosystem

- key generation
 - generate a large random prime p and choose generator g of the multiplicative group $Z_p^* = \{1, 2, \dots, p-1\}$
 - select a random integer a , $1 \leq a \leq p-2$, and compute $A = g^a \bmod p$
 - the public key is (p, g, A)
 - the private key is a
- encryption
 - represent the message as an integer m in $[0, p-1]$
 - select a random integer r , $1 \leq r \leq p-2$, and compute $R = g^r \bmod p$
 - compute $C = m \cdot A^r \bmod p$
 - the ciphertext is the pair (R, C)
- decryption
 - compute $m = C \cdot R^{p-1-a} \bmod p$
- proof of decryption

$$C \cdot R^{p-1-a} \equiv m \cdot A^r \cdot R^{p-1-a} \equiv m \cdot g^{ar} \cdot g^{r(p-1-a)} \equiv m \cdot (g^{p-1})^r \equiv m \pmod{p}$$

Relation to hard problems

- security of the ElGamal scheme is said to be based on the discrete logarithm problem in Z_p^* , although equivalence has not been proven yet
- recovering m given p, g, A, R , and C is equivalent to solving the Diffie-Hellman problem
- given the latest progress on the discrete logarithm problem, the size of the modulus p should at least be 1024 bits

Notes on the ElGamal scheme

- encryption requires two modular exponentiations, whereas decryption requires only one
- encrypted message is twice as long as the plaintext (message expansion)
- all entities in a system may choose to use the same prime p and generator g
 - size of the public key is reduced
 - encryption can be speed up by pre-computation

Exercise

- Show that in case of the ElGamal cryptosystem, it is crucial that different random integers r be used to encrypt different messages.