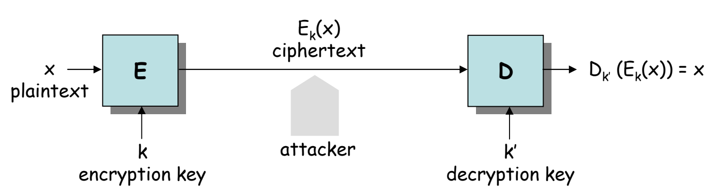
"The obvious mathematical breakthrough would be development of an easy way to factor large prime numbers."
-- Bill Gates, The Road Ahead, page 265

Public-key encryption

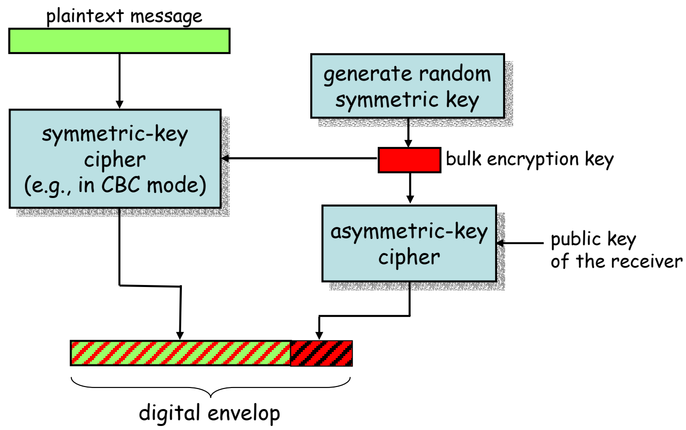
- general principles
- RSA cryptosystem
  - operation
  - properties of the textbook RSA
  - PKCS#1
- ElGamal cryptosystem

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



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

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, $0 \le x \le p-2$, 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 mod n = m^{ed} mod n = $m^{k\,\phi(n)+1}$ mod n = m $m^{k(p-1)(q-1)}$ mod n
- since m < n, it is enough to prove that m $m^{k(p-1)(q-1)} \equiv$ m (mod n)
- Fermat theorem
  - if r is a prime and gcd(a, r) = 1, then $a^{r-1} \equiv 1$ (mod r)
- if gcd(m, p) = 1
  - $m^{p-1} \equiv 1$ (mod p)
  - m $m^{k(p-1)(q-1)} \equiv$ m (mod p)
- if gcd(m, p) = p
  - p | m
  - m $m^{k(p-1)(q-1)} \equiv m \equiv 0$ (mod p)
- for all m, m $m^{k(p-1)(q-1)} \equiv$ m (mod p)
- similarly, for all m, m $m^{k(p-1)(q-1)} \equiv$ m (mod q)
- p, q | m $m^{k(p-1)(q-1)}$ - m
- m $m^{k(p-1)(q-1)} \equiv$ m (mod pq)

Public-key encryption / RSA

Implementing RSA – Computing d

- d can be computed using the extended Euclidean algorithm
- complexity:
  - let k be the length of n in bits (k = [$\log_2$n] + 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)

Public-key encryption / RSA

Implementing RSA – Modular exponentiation

- naïve approach:
  - $m^x \bmod n = m \cdot m \cdot m \cdot \ldots \cdot m \bmod n$
  - 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
  - $x = b_{k-1}2^{k-1} + b_{k-2}2^{k-2} + \ldots + b_1 2 + b_0$
  - $m^x = m^{b_0}(m^{x_1})^2$ where $x_1 = (x-b_0)/2 = b_{k-1}2^{k-2} + b_{k-2}2^{k-3} + \ldots + b_1$
  - $m^{x_1} = m^{b_1}(m^{x_2})^2$ where $x_2 = (x_1-b_1)/2 = b_{k-1}2^{k-3} + b_{k-2}2^{k-4} + \ldots + b_2$
  - …
  - $m^{x_{k-3}} = m^{b_{k-3}}(m^{x_{k-2}})^2$ 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 $O(k \cdot k^2) = O(k^3)$

RSA toy example

- key generation
  - let p = 73, q = 151
  - n = 73*151 = 11023
  - $\phi(n)$ = 72*150 = 10800
  - let e = 11
  - compute d with the extended Euclidean algorithm as follows:
    - 10800 = 981 x 11 + 9 t_2 = 0 – 981x1 mod 10800 = 9819
    - 11 = 1x9 + 2 t_3 = 1 – 1x9819 mod 10800 = 982
    - 9 = 4x2 + 1 t_4 = 9819 – 4x982 = 5891 → 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 → c = c^2m mod n = 17
    - b_2 = 0 → c = c^2 mod n = 289
    - b_1 = 1 → c = c^2m mod n = 1419857 mod 11023 = 8913
    - b_0 = 1 → c = c^2m mod n = … = 1782
    - output c = 17^{11} mod 11023 = 1782
- decryption
  - d = 5891 = 1011100000011 (in binary)
  - we compute m = c^d mod n with the "square and multiply" algorithm as above

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?

Public-key encryption / RSA

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

Public-key encryption / RSA

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

Problems - 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

Problems - 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, …)
- 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 $n_1, n_2, \ldots, n_k$ are pairwise relatively primes, then such a system has a unique solution (mod $n_1 \cdot n_2 \cdot \ldots \cdot n_k$)
  - 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 then $n^{1/e}$
- 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

Problems - Homomorphic property

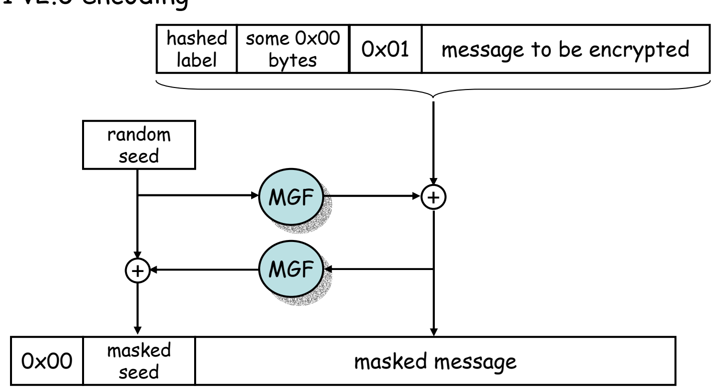
- if m_1 and m_2 are two plaintext messages and c_1 and c_2 are the corresponding ciphertexts, then the encryption of m_1m_2 mod n is c_1c_2 mod n
  - $(m_1m_2)^e \equiv m_1^e\, m_2^e \equiv c_1c_2 \pmod n$
- this leads to an adaptive chosen-ciphertext attack on RSA
  - assume that the attacker wants 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

0x00	0x02	at least 8 non-zero random bytes	0x00	message to be encrypted

- PKCS1 v2.0 encoding



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 ~2^{20} oracle call only

ElGamal cryptosystem

- key generation
  - generate a large random prime p and choose generator g of the multiplicative group $Z_p^* = \{1, 2, ..., p-1\}$
  - 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