IV054 CHAPTER 5: Public-key cryptography I. RSA

Rapidly increasing needs for flexible and secure transmission of information require to use new cryptographic methods.

The main disadvantage of the classical (symmetric) cryptography is the need to send a (long) key through a super secure channel before sending the message itself.

In the classical or secret-key (symmetric) cryptography both sender and receiver share the same secret key.

In the public-key (assymetric) cryptography there are two different keys:

a public encryption key (at the sender side)

and

a private (secret) decryption key (at the receiver side).

Basic idea - example

Basic idea: If it is infeasible from the knowledge of an encryption algorithm e_k to construct the corresponding description algorithm d_k, then e_k can be made public.

Toy example: (Telephone directory encryption)

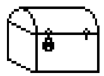
Start: Each user ***U*** makes public a unique telephone directory td_U to encrypt messages for ***U*** and ***U*** is the only user to have an inverse telephone directory itd_U.

Encryption: Each letter ***X*** of a plaintext ***w*** is replaced, using the telephone directory td_U of the intended receiver ***U***, by the telephone number of a person whose name starts with letter ***X***.

Decryption: easy for U_k, with the inverse telephone directory, infeasible for others.

Analogy:

Secret-key cryptography *1*. Put the message into a box, lock it with a padlock and send the box. *2*. Send the key by a secure channel.



Public-key cryptography Open padlocks, for each user different one, are freely available. Only legitimate user has key from his padlocks. *Transmission:* Put the message into the box of the intended receiver, close the padlock and send the box.

Public Establishment of Secret Keys

Main problem of the secret-key cryptography: a need to make a secure distribution (establishment) of secret keys ahead of transmissions.

Diffie+Hellman solved this problem in 1976 by designing a protocol for secure key establishment (distribution) over public channels.

Diffie-Helmann Protocol: If two parties, Alice and Bob, want to create a common secret key, then they first agree, somehow, on a large prime ***p*** and a q<p of large order in Z_p^* and then they perform, through a public channel, the following activities.

- Alice chooses, randomly, a large $1 \leq x < p - 1$ and computes

$$X = q^x \bmod p.$$

- Bob also chooses, again randomly, a large $1 \leq y < p - 1$ and computes

$$Y = q^y \bmod p.$$

- Alice and Bob exchange ***X*** and ***Y***, through a public channel, but keep ***x, y*** secret.
- Alice computes $Y^x \bmod p$ and Bob computes $X^y \bmod p$ and then each of them has the key $K = q^{xy} \bmod p.$

An eavesdropper seems to need, in order to determine ***x*** from ***X, q, p*** and ***y*** from ***Y, q, p***, a capability to compute discrete logarithms, or to compute q^{xy} from q^x and q^y, what is believed to be infeasible.

KEY DISTRIBUTION / AGREEMENT

One should distinguish between key distribution and key agreement.

- Key distribution is a mechanism whereby one party chooses a secret key and then transmits it to another party or parties.

- Key agreement is a protocol whereby two (or more) parties jointly establish a secret key by communication over a public channel.

The objective of key distribution or key agreement protocols is that, at the end of the protocols, the two parties involved both have possession of the same key k, and the value of k is not known (at all) to any other party.

MAN-IN-THE-MIDDLE ATTACK

The following attack, by a man-in-the-middle, is possible against the Diffie-Hellman key establishment protocol.

1. Eve chooses an exponent z.

2. Eve intercepts q^x and q^y.

3. Eve sends q^z to both Alice and Bob. (After that Alice believes she has received q^y and Bob believes he has received q^x.)

4. Eve computes $K_A = q^{xz} \pmod{p}$ and $K_B = q^{yz} \pmod{p}$.
Alice, not realizing that Eve is in the middle, also computes K_A and
Bob, not realizing that Eve is in the middle, also computes K_B.

5. When Alice sends a message to Bob, encrypted with K_A, Eve intercepts it, decrypts it, then encrypts it with K_B and sends it to Bob.

6. Bob decrypts the message with K_B and obtains the message. At this point he has no reason to think that communication was insecure.

7. Meanwhile, Eve enjoys reading Alice's message.

Blom's key pre-distribution protocol

allows to a trusted authority (Trent - TA) to distributed secret keys to $n(n-1)/2$ pairs of n users.

Let a large prime $p > n$ be publiclly known. Steps of the protocol:

1. Each user U in the network is assigned, by Trent, a unique public number $r_U < p$.
2. Trent chooses three random numbers a, b and c, smaller than p.
3. For each user U, Trent calculates two numbers

$$a_U = (a + br_U) \bmod p, \qquad b_U = (b + cr_U) \bmod p$$

and sends them via his secure channel to U.

4. Each user U creates the polynomial

$$g_U(x) = a_U + b_U(x).$$

5. If Alice (A) wants to send a message to Bob (B), then Alice computes her key $K_{AB} = g_A(r_B)$ and Bob computes his key $K_{BA} = g_B(r_A)$.
6. It is easy to see that $K_{AB} = K_{BA}$ and therefore Alice and Bob can now use their (identical) keys to communicate using some secret-key cryptosystem.

Secure communication with secret-key cryptosystems

and without any need for secret key distribution

(Shamir's ``no-key algorithm'')

Basic assumption: Each user ***X*** has its own

secret encryption function e_X

secret decryption function d_X

and all these functions commute (to form a commutative cryptosystem).

Communication protocol

with which Alice can send a message w to Bob.

1. Alice sends $e_A(w)$ to Bob
2. Bob sends $e_B(e_A(w))$ to Alice
3. Alice sends $d_A(e_B(e_A(w))) = e_B(w)$ to Bob
4. Bob performs the decryption to get $d_B(e_B(w)) = w$.

Disadvantage: 3 communications are needed (in such a context 3 is a much too large number) .

Advantage: A perfect protocol for distribution of secret keys.

Cryptography and Computational Complexity

Modern cryptography uses such encryption methods that no ``enemy'' can have enough computational power and time to do encryption (even those capable to use thousands of supercomputers during tens of years for encryption).

Modern cryptography is based on negative and positive results of complexity theory - on the fact that for some algorithm problems no efficient algorithm seem to exists, surprisingly, and for some "small'' modifications of these problems, surprisingly, simple, fast and good (randomized) algorithms do exist. Examples:

Integer factorization: Given ***n* (= *pq*),** it is, in general, unfeasible, to find ***p, q***.

There is a list of "most wanted to factor integers''. Top recent successes, using thousands of computers for months.

(*) Factorization of $2^{2^{9}} + 1$ with 155 digits (1996)

(**) Factorization of a "typical'' 155-digits integer (1999)

Primes recognition: Is a given ***n*** a prime? - fast randomized algorithms exist (1977).

The existence of polynomial deterministic algorithms has been shown only in 2002

Computationaly infeasible problems

Discrete logarithm problem: Given ***x, y, n***, determine integer ***a*** such that $y \equiv x^{a} \pmod{n}$ – infeasible in general.

Discrete square root problem: Given integers ***y, n***, compute an integer ***x*** such that $y \equiv x^{2} \pmod{n}$ - infeasible in general, easy if factorization of ***n*** is known

Knapsack problem: Given a (knapsack - integer) vector $X = (x_1,\ldots,x_n)$ and a (integer capacity) c, find a binary vector $(b_1,\ldots,b_n)$ such that

$$\sum_{i=1}^{n} b_i x_i = c.$$

Problem is *NP*-hard in general, but easy if $x_i > \sum_{j=1}^{i-1} x_j,\ \ 1 < i \le n.$

One-way functions

Informally, a function $F:N \to N$ is said to be one-way function if it is easily computable - in polynomial time - but any computation of its inverse is infeasible.

A one-way permutation is a 1-1 one-way function.

easy

x   *f*(*x*)

computationaly infeasible

A more formal approach

Definition A function $f:\{0,1\}^* \to \{0,1\}^*$ is called a strongly one-way function if the following conditions are satisfied:

1. f can be computed in polynomial time;
2. there are $c, \varepsilon > 0$ such that $|x|^{\varepsilon} \leq |f(x)| \leq |x|^{c}$;
3. for every randomized polynomial time algorithm A, and any constant $c > 0$, there exists an n_c such that for $n > n_c$

$$P_r\left(A(f(x)) \in f^{-1}(f(x))\right) < \frac{1}{n^c}.$$

Candidates: Modular exponentiation: $f(x) = a^x \bmod n$

Modular squaring $f(x) = x^2 \bmod n$, n - a Blum integer

Prime number multiplication $f(p, q) = pq$.

Trapdoor One-way Functions

The key concept for design of public-key cryptosystems is that of trapdoor one-way functions.

A function $f : X \to Y$ is trapdoor one-way function

- if f and its inverse can be computed efficiently,
- yet even the complete knowledge of the algorithm to compute f does not make it feasible to determine a polynomial time algorithm to compute the inverse of f.

A candidate: modular squaring with a fixed modulus.

- computation of discrete square roots is unfeasible in general, but quite easy if the decomposition of the modulus into primes is known.

A way to design a trapdoor one-way function is to transform an easy case of a hard (one-way) function to a hard-looking case of such a function, that can be, however, solved easily by those knowing how the above transformation was performed.

Example - Computer passwords

A naive solution is to keep in computer a file with entries as

login CLINTON **password** BUSH,

that is with logins and their passwords. This is not sufficiently safe.

A more safe method is to keep in the computer a file with entries as

login CLINTON **password** BUSH **one-way function** f_c

The idea is that BUSH is a “public" password and CLINTON is the only one that knows a “secret" password, say MADONA, such that

$$f_c(MADONA) = BUSH$$

LAMPORT's ONE-TIME PASSWORDS

One-way functions can be used to create a sequence of passwords:

- Alice chooses a random *w* and computes, using a one-way function *h*, a sequence of passwords

$$w, h(w), h(h(w)),\ldots,h^n(w)$$

- Alice then transfers securely ``the initial secret'' $w_0=h^n(w)$ to Bob.
- The i-th authentication, $0 < i < n+1$, is performed as follows:

------- Alice sends $w_i=h^{n-i}(w)$ to Bob for I = 1, 2,….,n-1

------- Bob checks whether $w_{i-1}=h(w_i)$.

When the number of identifications reaches *n*, a new *w* has to be chosen.

General knapsack problem - unfeasible

KNAPSACK PROBLEM: Given an integer-vector $X = (x_1,\ldots,x_n)$ and an integer c. Determine a binary vector $B = (b_1,\ldots,b_n)$ (if it exists) such that $XB^T = c$.

Knapsack problem with superincreasing vector – easy

Problem Given a superincreasing integer-vector $X = (x_1,\ldots,x_n)$ (i.e. $x_i > \sum_{j=1}^{i-1} x_j, i > 1$) and an integer c,

determine a binary vector $B = (b_1,\ldots,b_n)$ (if it exists) such that $XB^T = c$.

Algorithm - to solve knapsack problems with superincreasing vectors:

for $i \leftarrow n$ **downto** 2 **do**
    if $c \geq 2x_i$ **then** terminate {no solution}
        else if $c > x_i$ **then** $b_i \leftarrow 1; c \leftarrow c - x_i$;
            else $b_i = 0$;
if $c = x_1$ **then** $b_1 \leftarrow 1$
    else if $c = 0$ **then** $b_1 \leftarrow 0$;
        else terminate {no solution}

Example $X = (1,2,4,8,16,32,64,128,256,512)$ $c = 999$

$X = (1,3,5,10,20,41,94,199)$ $c = 242$

KNAPSACK ENCODING - BASIC IDEAS

Let a (knapsack) vector

$$A = (a_1,\ldots,a_n)$$

be given.

Encoding of a (binary) message $B = (b_1, b_2,\ldots,b_n)$ by A is done by the vector/vector multiplication:

$$AB^T = c$$

and results in the cryptotext c.

Decoding of c requires to solve the knapsack problem for the instant given by the knapsack vector A and the cryptotext c.

The problem is that decoding seems to be infeasible.

Example

If A = (74, 82,94, 83, 39, 99, 56, 49, 73, 99) and B = (1100110101) then

$$AB^T =$$

Design of knapsack cryptosystems

1. Choose a superincreasing vector $\boldsymbol{X} = (x_1,\ldots,x_n)$.
2. Choose m, u such that $m > 2x_n$, $gcd(m, u) = 1$.
3. Compute u^{-1} **mod** m, $X' = (x_1',\ldots,x_n')$, $x_i' = \underbrace{\underbrace{ux_i}_{\text{diffusion}} \bmod m}_{\text{confusion}}$.

Cryptosystem: $\boldsymbol{X'}$ - public key

$\boldsymbol{X, u, m}$ - trapdoor information

Encryption: of a binary vector w of length n: $\boldsymbol{c = X' w}$

Decryption: compute $\boldsymbol{c' = u^{-1}c \bmod m}$

and solve the knapsack problem with X and c'.

Lemma Let X, m, u, X', c, c' be as defined above. Then the knapsack problem instances (X, c') and (X', c) have at most one solution, and if one of them has a solution, then the second one has the same solution.

Proof Let $X'w = c$. Then

$$\boldsymbol{c' \equiv u^{-1}c \equiv u^{-1}X'w \equiv u^{-1}uXw \equiv Xw \pmod{m}}.$$

Since X is superincreasing and $m > 2x_n$ we have

$$\boldsymbol{(X w) \bmod m = X w}$$

and therefore $$\boldsymbol{c' = Xw}.$$

Design of knapsack cryptosystems - example

Example $X = (1,2,4,9,18,35,75,151,302,606)$

$m = 1250$, $u = 41$

$X' = (41,82,164,369,738,185,575,1191,1132,1096)$

In order to encrypt an English plaintext, we first encode its letters by 5-bit numbers _ - 00000, A - 00001, B - 00010,… and then divide the resulting binary strings into blocks of length 10.

Plaintext: Encoding of AFRICA results in vectors

$w_1 = (0000100110)$ $w_2 = (1001001001)$ $w_3 = (0001100001)$

Encryption: $c_{1'} = X'w_1 = 3061$ $c_{2'} = X'w_2 = 2081$ $c_{3'} = X'w_3 = 2203$

Cryptotext: (3061,2081,2203)

Decryption of cryptotexts: (2163, 2116, 1870, 3599)

By multiplying with $u^{-1} = 61$ (mod 1250) we get new cryptotexts (several new c')

(693, 326, 320, 789)

And, in the binary form, solutions B of equations $XB^T=c'$ have the form

(1101001001, 0110100010, 0000100010, 1011100101)

Therefor, the resulting plaintext is:

ZIMBABWE

Story of the Knapsack

Invented: 1978 - Ralp C. Merkle, Martin Hellman

Patented: in 10 countries

Broken: 1982: Adi Shamir

New idea: iterated knapsack cryptosystem using hyper-reachable vectors.

Definition A knapsack vector $X' = (x_{1'},\ldots,x_{n'})$ is obtained from a knapsack vector $X=(x_1,\ldots,x_n)$ by strong modular multiplication if

$$X'_i = ux_i \bmod m,\ i = 1,\ldots,n,$$

where

$$m > 2\sum_{i=1}^{n} x_i$$

and $gcd(u, m) = 1$. A knapsack vector X' is called hyper-reachable, if there is a sequence of knapsack vectors $X = x_0, x_1,\ldots,x_k = X'$,

where x_0 is a super-increasing vector and for $i = 1,\ldots,k\}$ and x_i is obtained from x_{i-1} by a strong modular multiplication.

Iterated knapsack cryptosystem was broken in 1985 - E. Brickell

New ideas: dense knapsack cryptosystems. Density of a knapsack vector: $X=(x_1,\ldots,x_n)$ is defined by $d(x) = \frac{n}{\log(\max\{x_i \mid 1 \le i \le n\})}$

Remark. Density of super-increasing vectors is $\le \frac{n}{n-1}$

KNAPSACK CRYPTOSYSTEM - COMMENTS

The term “knapsack" in the name of the cryptosystem is quite misleading.

By the Knapsack problem one usually understands the following problem:

Given n items with weights $w_1, w_2,\ldots, w_n$ and values $v_1, v_2,\ldots, v_n$ and a knapsack limit c, the task is to find a bit vector $(b_1, b_2,\ldots, b_n)$ such that $\sum_{i=1}^{n} b_i w_i \leq c$ and $\sum_{i=1}^{n} b_i v_i$ is as large as possible.

The term subset problem is usually used for the problem used in our construction of the knapsack cryptosystem. It is well-known that the decision version of this problem is *NP*-complete.

Sometimes, for our main version of the knapsack problem the term Merkle-Hellmman (Knapsack) Cryptosystem is used.

McEliece Cryptosystem

McEliece cryptosystem is based on a similar design principle as the Knapsack cryptosystem. McEliece cryptosystem is formed by transforming an easy to break cryptosystem into a cryptosystem that is hard to break because it seems to be based on a problem that is, in general, *NP*-hard.

The underlying fact is that the decision version of the decryption problem for linear codes is in general *NP*-complete. However, for special types of linear codes polynomial-time decryption algorithms exist. One such a class of linear codes, the so-called Goppa codes, are used to design McEliece cryptosystem.

Goppa codes are $[2^m, n - mt, 2t + 1]$-codes, where $n = 2^m$.

(McEliece suggested to use $m = 10$, $t = 50$.)

McEliece Cryptosystem - DESIGN

Goppa codes are $[2^m, n - mt, 2t + 1]$-codes, where $n = 2^m$.

Design of McEliece cryptosystems. Let

- G be a generating matrix for an $[n, k, d]$ Goppa code C;
- S be a $k \times k$ binary matrix invertible over Z_2;
- P be an $n \times n$ permutation matrix;
- $G' = SGP$.

Plaintexts: $\boldsymbol{P} = (Z_2)^k$; cryptotexts: $\mathbf{C} = (Z_2)^n$, key: $\boldsymbol{K} = (G, S, P, G')$, message: w

G' is made public, G, S, P are kept secret.

Encryption: $e_K(w, e) = wG' + e$, where e is any binary vector of length n & weight t.

Decryption of a cryptotext $c = wG'+e \in (Z_2)^n$.

1. Compute $c_1 = cP^{-1} = wSGPP^{-1} + eP^{-1} = wSG+eP^{-1}$
2. Decode c_1 to get $w_1 = wS$,
3. Compute $w = w_1S^{-1}$

COMMENTS on McELIECE CRYPTOSYSTEM

1. Each irreducible polynomial over Z_2^m of degree t generates a Goppa code with distance at least $2t + 1$.
2. In the design of McEliece cryptosystem the goal of matrices S and C is to modify a generator matrix G for an easy-to-decode Goppa code to get a matrix that looks as a general random matrix for a linear code for which decoding problem is ***NP***-complete.
3. An important novel and unique trick is an introduction, in the encoding process, of a random vector e that represents an introduction of up to t errors - such a number of errors that are correctable using the given Goppa code and this is the basic trick of the decoding process.
4. Since P is a permutation matrix eP^{-1} has the same weight as e.
5. As already mentioned, McEliece suggested to use a Goppa code with m=10 and t=50. This provides a [1024, 524, 101]-code. Each plaintext is then a 524-bit string, each cryptotext is a 1024-bit string. The public key is an 524 × 1024 matrix.
6. Observe that the number of potential matrices S and P is so large that probability of guessing these matrices is smaller that probability of guessing correct plaintext!!!
7. It can be shown that it is not safe to encrypt twice the same plaintext with the same public key (and different error vectors).

FINAL COMMENTS

1. Public-key cryptosystems can never provide unconditional security. This is because an eavesdropper, on observing a cryptotext *c* can encrypt each possible plaintext by the encryption algorithm e_A until he finds an *c* such that $e_A(w) = c$.

2. One-way functions exists if and only if **P = UP**, where UP is the class of languages accepted by **unambiguous polynomial time bounded nondeterministic Turing machine**.

3. There are actually two types of keys in practical use: A session key is used for sending a particular message (or few of them). A master key is usually used to generate several session keys.

4. Session keys are usually generated when actually required and discarded after their use. Session keys are usually keys of a secret-key cryptosystem.

5. Master keys are usually used for longer time and need therefore be carefully stored. Master keys are usually keys of a public-key cryptosystem.

SATELLITE VERSION of ONE-TIME PAD

Suppose a satellite produces and broadcasts several random sequences of bits at a rate fast enough that no computer can store more than a small fraction of the output.

If Alice wants to send a message to Bob they first agree, using a public key cryptography, on a method of sampling bits from the satellite outputs.

Alice and Bob use this method to generate a random key and they use it with ONE-TIME PAD for encryption.

By the time Eve decrypted their public key communications, random streams produced by the satellite and used by Alice and Bob to get the secret key have disappeared, and therefore there is no way for Eve to make decryption.

The point is that satellites produce so large amount of date that Eve cannot store all of them

RSA cryptosystem

The most important public-key cryptosystem is the RSA cryptosystem on which one can also illustrate a variety of important ideas of modern public-key cryptography.

For example, we will discuss various possible attacks on the RSA cryptosystem and problems related to security of RSA.

A special attention will be given in Chapter 7 to the problem of factorization of integers that play such an important role for security of RSA.

In doing that we will illustrate modern distributed techniques to factorize very large integers.

DESIGN and USE of RSA CRYPTOSYSTEM

Invented in 1978 by Rivest, Shamir, Adleman

Basic idea: prime multiplication is very easy, integer factorization seems to be unfeasible.

Design of RSA cryptosystems

1. Choose two large s-bit primes *p,q, s in [512,1024],* and denote

$$n = pq,\ \phi(n) = (p-1)(q-1)$$

2. Choose a large *d* such that

$$\gcd(d, \phi(n)) = 1$$

and compute

$$e = d^{-1} (\bmod\ \phi(n))$$

Public key: *n* (modulus), *e* (encryption algorithm)

Trapdoor information: *p, q, d* (decryption algorithm)

Plaintext *w*

Encryption: cryptotext $c = w^e \bmod n$

Decryption: plaintext $w = c^d \bmod n$

Details: A plaintext is first encoded as a word over the alphabet {0, 1,…,9}, then divided into blocks of length *i* -1, where $10^{i-1} < n < 10^{i}$. Each block is taken as an integer and decrypted using modular exponentiation.

Correctness of RSA

Let $c = w^e \bmod n$ be the cryptotext for a plaintext w, in the cryptosystem with

$$n = pq,\ ed \equiv 1(\bmod\,\phi(n)),\ \gcd(d,\phi(n)) = 1$$

In such a case $$w \equiv c^d \bmod n$$

and, if the decryption is unique, $w = c^d \bmod n$.

Proof Since $ed \equiv 1(\bmod\,\phi(n))$, there exist a j € N such that $ed = j\phi(n)+1$.

- Case 1. Neither p nor q divides w.

In such a case gcd(n, w) = 1 and by the Euler's Totien Theorem we get that

$$c^d = w^{ed} = w^{j\phi(n)+1} \equiv w(\bmod\,n)$$

- Case 2. Exactly one of p,q divides w - say p.

In such a case $w^{ed} \equiv w$ (mod p) and by Fermat's Little theorem $w^{q-1} \equiv 1$ (mod q)

$$\Rightarrow w^{q-1} \equiv 1(\bmod\,q) \Rightarrow w^{\phi(n)} \equiv 1(\bmod\,q)$$

$$\Rightarrow w^{j\phi(n)} \equiv 1(\bmod\,q)$$

$$\Rightarrow w^{ed} \equiv w(\bmod\,q)$$

Therefore: $w \equiv w^{ed} \equiv c^d\ (\bmod\,n)$

- Case 3 Both p,q divide w.

This cannot happen because, by our assumption, $w < n$.

DESIGN and USE of RSA CRYPTOSYSTEM

Example of the design and of the use of RSA cryptosystems.

- By choosing $p = 41, q = 61$ we get $n = 2501$, $\phi(n) = 2400$
- By choosing $d = 2087$ we get $e = 23$
- By choosing $d = 2069$ we get e=29
- By choosing other values of d we would get other values of e.

Let us choose the first pair of encryption/decryption exponents (e=23 and d=2087).

Plaintext: KARLSRUHE

Encoding: 100017111817200704

Since $10^3 < n < 10^4$, the numerical plaintext is divided into blocks of 3 digits $\Rightarrow$ 6 plaintext integers are obtained

Encryption: 100, 017, 111, 817, 200, 704

$$100^{23} \bmod 2501,\quad 17^{23} \bmod 2501,\quad 111^{23} \bmod 2501$$

$$817^{23} \bmod 2501,\ 200^{23} \bmod 2501,\ 704^{23} \bmod 2501$$

provides cryptotexts: 2306, 1893, 621, 1380, 490, 313

Decryption:

$$2306^{2087} \bmod 2501 = 100,\quad 1893^{2087} \bmod 2501 = 17$$

$$621^{2087} \bmod 2501 = 111,\quad 1380^{2087} \bmod 2501 = 817$$

$$490^{2087} \bmod 2501 = 200,\quad 313^{2087} \bmod 2501 = 704$$

RSA challenge

One of the first description of RSA was in the paper.

Martin Gardner: Mathematical games, Scientific American, 1977

and in this paper RSA inventors presented the following challenge.

Decrypt the cryptotext:

9686 9613 7546 2206 1477 1409 2225 4355 8829 0575 9991 1245 7431 9874 6951 2093 0816 2982 2514 5708 3569 3147 6622 8839 8962 8013 3919 9055 1829 9451 5781 5154

Encrypted using the RSA cryptosystem with 129 digit number, called also RSA129

n: 114 381 625 757 888 867 669 235 779 976 146 612 010 218 296 721 242 362 562 561 842 935 706 935 245 733 897 830 597 123 513 958 705 058 989 075 147 599 290 026 879 543 541.

and with e = 9007.

The problem was solved in 1994 by first factorizing n into one 64-bit prime and one 65-bit prime, and then computing the plaintext

THE MAGIC WORDS ARE SQUEMISH OSSIFRAGE

How to design a good RSA cryptosystem

1. How to choose large primes *p,q*?

Choose randomly a large integer *p*, and verify, using a randomized algorithm, whether *p* is prime. If not, check *p* + 2, *p* + 4,…

From the Prime Number Theorem if follows that there are approximately

$$\frac{2^d}{\log 2^d} - \frac{2^{d-1}}{\log 2^{d-1}}$$

d bit primes. (A probability that a 512-bit number is prime is 0.00562.)

2. What kind of relations should be between *p* and *q*?

2.1 Difference |*p*-*q*| should be neither too small not too large.

2.2 gcd(*p*-1, *q*-1) should not be large.

2.3 Both *p*-1 and *q*-1 should contain large prime factors.

2.4 Quite ideal case: *q, p* should be safe primes - such that also (*p*–1)/2 and (*q*-1)/2 are primes. (83,107,10^{100} – 166517 are examples of safe primes).

3. How to choose *e* and *d*?

3.1 Neither *d* nor *e* should be small.

3.2 *d* should not be smaller than $n^{1/4}$. (For $d < n^{1/4}$ a polynomial time algorithm is known to determine *d*).

Prime recognition and factorization

The key problems for the development of RSA cryptosystem are that of prime recognition and integer factorization.

On August 2002, the first polynomial time algorithm was discovered that allows to determine whether a given *m* bit integer is a prime. Algorithm works in time $O(m^{12})$.

Fast randomized algorithms for prime recognition has been known since 1977. One of the simplest one is due to Rabin and will be presented later.

For integer factorization situation is somehow different.

- No polynomial time classical algorithm is known.
- Simple, but not efficient factorization algorithms are known.
- Several sophisticated distributed factorization algorithms are known that allowed to factorize, using enormous computation power, surprisingly large integers.
- Progress in integer factorization, due to progress in algorithms and technology, has been recently enormous.
- Polynomial time quantum algorithms for integer factorization are known since 1994 (P. Shor).

Several simple and some sophisticated factorization algorithms will be presented and illustrated in the following.

Rabin-Miller's prime recognition

Rabin-Miller's Monte Carlo prime recognition algorithm is based on the following result from the number theory.

Lemma Let $n \in N$. Denote, for $1 \leq x \leq n$, by $C(x)$ the condition:

Either $x^{n-1} \neq 1 \pmod{n}$, or there is an $m = \frac{n-1}{2^i}$ for some i, such that $\gcd(n, x^m - 1) \neq 1.$

If $C(x)$ holds for some $1 \leq x \leq n$, then n is not a prime. If n is not a prime, then $C(x)$ holds for at least half of x between 1 and n.

Algorithm:

Choose randomly integers $x_1, x_2, \ldots, x_m$ such that $1 \leq x_i \leq n$.

For each x_i determine whether $C(x_i)$ holds.

Claim: If $C(x_i)$ holds for some i, then n is not a prime for sure. Otherwise n is prime, with probability of error 2^{-m}.

Factorization of 512-bits and 663-bits numbers

On August 22, 1999, a team of scientifists from 6 countries found, after 7 months of computing, using 300 very fast SGI and SUN workstations and Pentium II, factors of the so-called RSA-155 number with 512 bits (about 155 digits).

RSA-155 was a number from a Challenge list issue by the US company RSA Data Security and “represented" 95% of 512-bit numbers used as the key to protect electronic commerce and financinal transmissions on Internet.

Factorization of RSA-155 would require in total 37 years of computing time on a single computer.

When in 1977 Rivest and his colleagues challenged the world to factor RSA-129, they estimated that, using knowledge of that time, factorization of RSA-129 would require 10^{16} years.

In 2005 RSA-200, a 663-bits number, was factorized by a team of German Federal Agency for Information Technology Security, using CPU of 80 AMD Opterons.

LARGE NUMBERS

Hindus named many large numbers - one having 153 digits.

Romans initially had no terms for numbers larger than 10^4.

Greeks had a popular belief that no number is larger than the total count of sand grains needed to fill the universe.

Large numbers with special names:

duotrigintillion=googol - 10^{100} googolplex - $10^{10\wedge 100}$

FACTORIZATION of very large NUMBERS

W. Keller factorized F_{23471} which has 10^{7000} digits.

J. Harley factorized: $10^{10\wedge 1000}+1$.

One factor: 316,912,650,057,350,374,175,801,344,000,001

1992 E. Crandal, Doenias proved, using a computer that F_{22}, which has more than million of digits, is composite (but no factor of F_{22} is known).

Number $10^{10^{10^{34}}}$ was used to develop a theory of the distribution of prime numbers.

IV054 DESIGN OF GOOD RSA CRYPTOSYSTEMS

Claim 1. Difference $|p-q|$ should not be small.

Indeed, if $|p - q|$ is small, and $p > q$, then $(p + q)/2$ is only slightly larger than $\sqrt{n}$ because

$$\frac{(p+q)^2}{4} - n = \frac{(p-q)^2}{4}$$

In addition $\frac{(p+q)^2}{4} - n$ is a square, say y^2.

In order to factor n, it is then enough to test $x > \sqrt{n}$ until x is found such that x^2 - n is a square, say y^2. In such a case

$p + q = 2x$, $p - q = 2y$ and therefore $p = x + y$, $q = x - y$.

Claim 2. gcd(p-1, q-1) should not be large.

Indeed, in the opposite case s = lcm(p-1, q-1) is much smaller than $\phi(n)$. If

$$d'e \equiv 1 \bmod s,$$

then, for some integer k,

$$c^{d'} \equiv w^{ed'} \equiv w^{ks+1} \equiv w \bmod n$$

since $p - 1|s$, $q - 1|s$ and therefore $w^{k1s} \equiv 1 \bmod p$ and $w^{ks+1} \equiv w \bmod q$. Hence, d' can serve as a decryption exponent.

Moreover, in such a case s can be obtained by testing.

Question Is there enough primes (to choose again and again new ones)?

No problem, the number of primes of length 512 bit or less exceeds 10^{150}.

How important is factorization for breaking RSA?

1. If integer factorization is feasible, then RSA is breakable.

2. There is no proof that factorization is indeed needed to break RSA.

3. If a method of breaking RSA would provide an effective way to get a trapdoor information, then factorization could be done effectively.

Theorem Any algorithm to compute $\phi(n)$ can be used to factor integers with the same complexity.

Theorem Any algorithm for computing d can be converted into a break randomized algorithm for factoring integers with the same complexity.

4. There are setups in which RSA can be broken without factoring modulus n.

Example An agency chooses p, q and computes a modulus $n = pq$ that is publicized and common to all users U_1, U_2 and also encryption exponents e_1, e_2,… are publicized. Each user U_i gets his decryption exponent d_i.

In such a setting any user is able to find in deterministic quadratic time another user's decryption exponent.

Security of RSA

None of the numerous attempts to develop attacks on RSA has turned out to be successful.

There are various results showing that it is impossible to obtain even only partial information about the plaintext from the cryptotext produces by the RSA cryptosystem.

We will show that were the following two functions, that are computationally polynomially equivalent, be efficiently computable, then the RSA cryptosystem with the encryption (decryption) algorithm e_k (d_k) would be breakable.

$parity_{ek}(c)$ = the least significant bit of such an w that $e_k(w) = c$;

$$half_{ek}(c) = 0 \text{ if } 0 \leq \text{w} < \frac{\text{n}}{2} \text{ and } half_{ek}(c) = 1 \text{ if } \frac{n}{2} \leq w \leq n-1.$$

We show two important properties of the functions *half* and *parity*.

1. Polynomial time computational equivalence of the functions *half* and *parity* follows from the following identities

$$half_{ek}(c) = parity_{ek}((c \times e_k(2)) \bmod n)$$

$$parity_{ek}(c) = half_{ek}((c \times e_k(\tfrac{1}{2})) \bmod n)$$

and the multiplicative rule $e_k(w_1)e_k(w_2) = e_k(w_1w_2)$.

2. There is an efficient algorithm to determine plaintexts w from the cryptotexts c obtained by RSA-decryption provided efficiently computable function *half* can be used as the oracle:

BREAKING RSA USING AN ORACLE

Algorithm:

for $i = 0$ to $\lfloor \lg n \rfloor$ **do**

$c_i \leftarrow half(c)$; $c \leftarrow (c \times e_k(2)) \bmod n$

$l \leftarrow 0$; $u \leftarrow n$

for $i = 0$ to $[\lg n]$ **do**

$m \leftarrow (i + u) / 2$;

if $c_i = 1$ **then** $i \leftarrow m$ **else** $u \leftarrow m$;

output $\leftarrow [u]$

Indeed, in the first cycle

$$c_i = half\left(c \times (e_k(2))^i\right) = half\left(e_k\left(2^i w\right)\right),$$

is computed for $0 \leq i \leq \lg n$.

In the second part of the algorithm binary search is used to determine interval in which w lies. For example, we have that

$$half(e_k(w)) = 0 \equiv w \in \left[0, \tfrac{n}{2}\right)$$

$$half(e_k(2w)) = 0 \equiv w \in \left[0, \tfrac{n}{4}\right) \cup \left[\tfrac{n}{2}, \tfrac{3n}{4}\right)$$

$$half(e_k(4w)) = 0 \equiv w \in$$

Security of RSA

There are many results for RSA showing that certain parts are as hard as whole. For example any feasible algorithm to determine the last bit of the plaintext can be converted into a feasible algorithm to determine the whole plaintext.

Example Assume that we have an algorithm ***H*** to determine whether a plaintext x designed in RSA with public key e, n is smaller than $n / 2$ if the cryptotext y is given.

We construct an algorithm ***A*** to determine in which of the intervals $(jn/8, (j+1)n/8)$, $0 \leq j \leq 7$ the plaintext lies.

Basic idea H can be used to decide whether the plaintexts for cryptotexts $x^e \bmod n$, $2^e x^e \bmod n$, $4^e x^e \bmod n$ are smaller than $n / 2$.

Answers

yes, yes, yes	$0 < x < n/8$	no, yes, yes	$n/2 < x < 5n/8$
yes, yes, no	$n/8 < x < n/4$	no, yes, no	$5n/8 < x < 3n/4$
yes, no, yes	$n/4 < x < 3n/8$	no, no, yes	$3n/4 < x < 7n/8$
yes, no, no	$3n/8 < x < n/2$	no, no, no	$7n/8 < x < n$

RSA with a composite "to be a prime"

Let us explore what happens if some integer p used, as a prime, to design a RSA is actually not a prime.

Let $n = pq$ where q be a prime, but $p = p_1p_2$, where p_1, p_2 are primes. In such a case

$$\phi(n) = (p_1 - 1)(p_2 - 1)(q - 1)$$

but assume that the RSA-designer works with $\phi_1(n) = (p-1)(q-1)$

Let $u = \mathrm{lcm}(p_1 - 1, p_2 - 1, q - 1)$ and let $\gcd(w, n) = 1$. In such a case

$$w^{p_1-1} \equiv 1 (\bmod p_1),\ w^{p_2-1} \equiv 1 (\bmod p_2),\ w^{q-1} \equiv 1 (\bmod q)$$

and as a consequence

$$w^u \equiv 1 (\bmod n)$$

In such a case u divides $\phi(n)$ and let us assume that also u divides $\phi_1(n)$.

Then

$$w^{\phi_1(n)+1} \equiv w (\bmod n).$$

So if $e_d \equiv 1 \bmod \phi_1(n)$, then encryption and decryption work as if p were prime.

Example $p = 91 = 7 \cdot 13$, $q = 41$, $n = 3731$, $\phi_1(n) = 3600$, $\phi(n) = 2880$, $\mathrm{lcm}(6, 12, 40) = 120$, $120 | \phi_1(n)$.

If $\gcd(d, \phi_1(n)) = 1$, then $\gcd(d, \phi(n)) = 1 \Rightarrow$ one can compute e using $\phi_1(n)$.
However, if u does not divide $\phi_1(n)$, then the cryptosystem does not work properly.

Two users should not use the same modulus

Otherwise, users, say *A* and *B*, would be able to decrypt messages of each other using the following method.

Decryption: *B* computes

$$f = \gcd(e_B d_B - 1, e_A), m = \frac{e_B d_B - 1}{f}$$

Since

$$e_B d_B - 1 = k\phi(\mathrm{n}) \text{ for some } k$$

it holds:

$$\gcd(e_A, \phi(n)) = 1 \Rightarrow \gcd(f, \phi(n)) = 1$$

and therefore

$$m \text{ is a multiple of } \phi(n).$$

m and e_A have no common divisor and therefore there exist integers u, v such that

$$um + ve_A = 1$$

Since m is a multiple of $\phi(n)$ we have

$$ve_A = 1 - um \equiv 1 \bmod \phi(n)$$

and since $e_A d_A \equiv 1 \bmod \phi(n)$ we have

$$(v - d_A)e_A \equiv 0 \bmod \phi(n)$$

and therefore

$$v \equiv d_A \bmod \phi(n)$$

is a decryption exponent of *A*. Indeed, for a cryptotext *c*:

$$c^v \equiv w^{e_A v} \equiv w^{e_A d_A + c\phi(n)} \equiv w \bmod n.$$

Private-key versus public-key cryptography

- The prime advantage of public-key cryptography is increased security - the private keys do not ever need to be transmitted or revealed to anyone.
- Public key cryptography is not meant to replace secret-key cryptography, but rather to supplement it, to make it more secure.
- Example RSA and DES (AES) are usually combined as follows
  1. The message is encrypted with a random DES key
  2. DES-key is encrypted with RSA
  3. DES-encrypted message and RSA-encrypted DES-key are sent.

This protocol is called RSA digital envelope.

- In software (hardware) DES is generally about 100 (1000) times faster than RSA.

If n users communicate with secrete-key cryptography, they need $n(n - 1) / 2$ keys.
If n users communicate with public-key cryptography $2n$ keys are sufficient.

Public-key cryptography allows spontaneous communication.

KERBEROS

We describe a very popular key distribution protocol with trusted authority TA with which each user *A* shares a secrete key K_A.

- To communicate with user *B* the user *A* asks TA a session key (K)
- TA chooses a random session key *K*, a time-stamp *T*, and a lifetime limit *L*.
- TA computes

$$m_1 = e_{K_A}(K, ID(B), T, L); \quad m_2 = e_{K_B}(K, ID(B), T, L);$$

and sends m_1, m_2 to *A*.

- *A* decrypts m_1, recovers *K, T, L, ID(B),* computes $m_3 = e_K(ID(B), T)$ and sends m_2 and m_3 to *B*.
- *B* decrypts m_2 and m_3, checks whether two values of *T* and of *ID(B)* are the same. If so, *B* computes $m_4 = e_K(T+1)$ and sends it to *A*.
- *A* decrypts m_4 and verifies that she got $T+1$.