Part V

Public-key cryptosystems, I. Key exchange, knapsack, RSA

PROLOGUE

SECURE ENCRYPTION - a PRACTICAL POINT OF VIEW

From practical point of view encryptions by a cryptosystem can be considered as secure if they cannot be broken by

many (thousands) supercomputers with exaflop performance working for some years.

MOST POWERFUL SUPERCOMPUTERS NOWADAYS

1. Tianhe-2, China, 33.8 petaflops, 3,120.000 cores
2. Titan, Cray XK7, OAK Ridge, 17.6 petaflops, 560,640 processors
3. Sequoia, IBM BlueGene, 16.32 petaflops, 1,472,864 cores
4. K, Fujitsu, 11 petaflops, 705,024 cores
5. Mira, IBM BlueGene/Q Argone National Lab., 10 petaflops, 786,432 cores

In April 2013 (June 2014) [June 2015] there were 26 (37) [68] computer systems with more than one petaflop performance.

Performance of the computer on 100 position increased in six months from 172 to 241 Teraflops

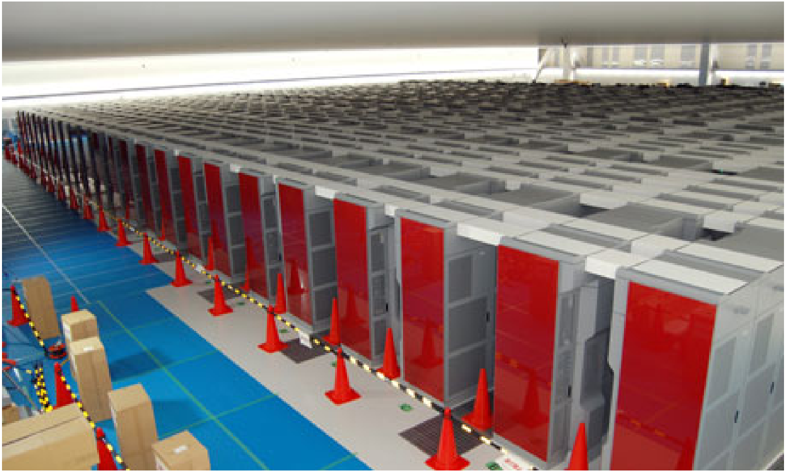
Out of 500 most powerful computer systems in June 2014, 233 was in US, 123 in Asia, 105 in Europe, 76 in China, 30 in UK, 30 in Japan, 27 in France, 11 in India...

Exaflops computers (10^{18}) are expected in 2019

Combined performance of 500 top supercomputers was 361 petaflops in June 2015, and 274 petaflops a year ago - 31% increase in one year.

Supercomputer Salomon in Ostrava, with performance 1.407 petaflops was on 40th place in June 2015; best in India on 79th place.

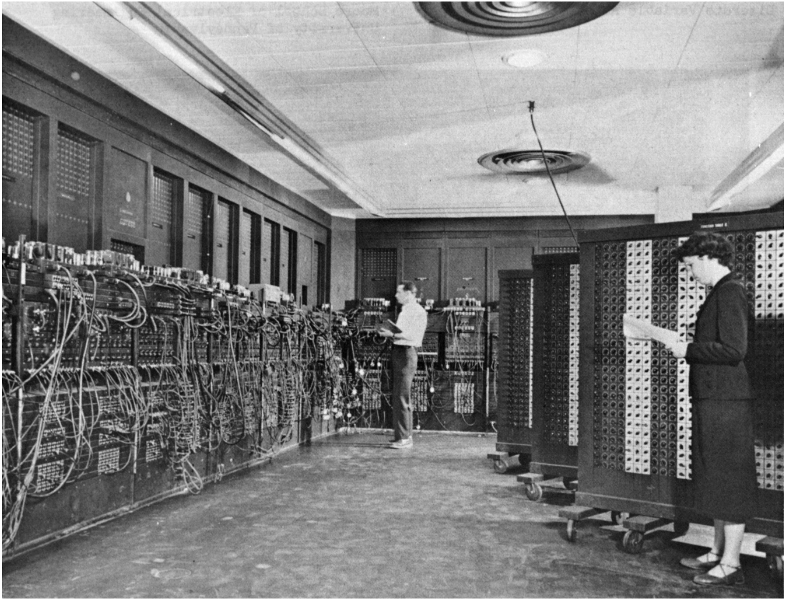
K COMPUTER



K-COMPUTER

TITAN-COMPUTER

ENIAC-COMPUTER



The main problem of secret key (or symmetric) cryptography is that in order to send securely

a message

there is a need to send at first securely

a secret key.

Therefore, **secret key cryptography is not a sufficiently good tool for massive communication capable to protect secrecy, privacy and anonymity.**

PUBLIC KEY CRYPTOGRAPHY

In this chapter we first describe the birth of public key cryptography, that can better manage the key distribution problem, and then three public-key cryptosystems, especially **RSA** cryptosystem.

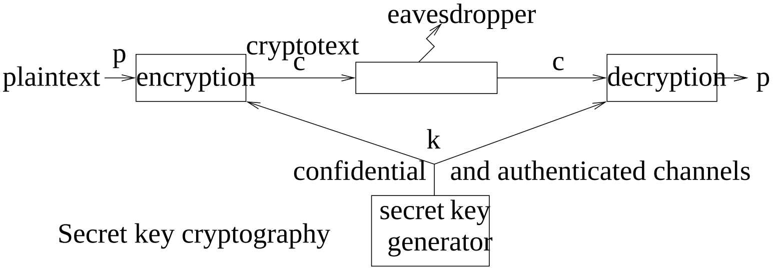
The basic idea of a public key cryptography:

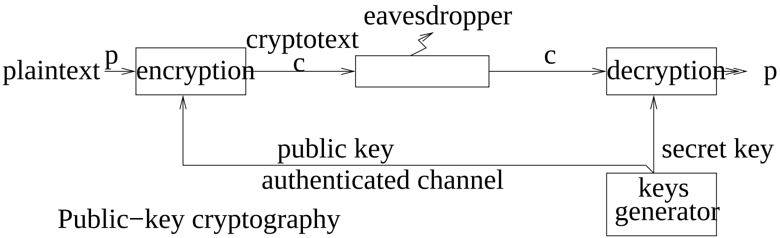
In a public key cryptosystem not only the encryption and decryption algorithms are public, but for each user U also the key e_U for encrypting messages (by anyone) for U is public.

Moreover, each user U gets/creates and keeps secret a specific (decryption) key, d_U, that can be used for decryption of messages that were addressed to him and encrypted with the help of the public encryption key e_U.

Encryption and decryption keys of public key cryptography could (and should) be different - we can therefore say also that pulic-key cryptography is **asymmetric cryptography**. Secret key cryptography, that has the same key for encryption and for decryption is then called also as **symmetric cryptography**.

SYMMETRIC versus ASYMMETRIC CRYPTOSYSTEMS





KEY DISTRIBUTION PROBLEM

KEYS DISTRIBUTION PROBLEM - HISTORY

- **The main problem of secret-key cryptography**: Before two users can exchange secretly (a message) they must already share a secret (the encryption/decryption key).
- Key distribution has been a big problem for 2000 of years, especially during both World Wars.
- **Around 1970** the vision of an internet started to appear (ARPAnet was created in 1969) and it started to be clear that an enormous communication potential that a whole world connecting network could provide, could hardly be fully utilized unless secrecy of communication can be established. Therefore the **key distribution problem started to be seen as the problem of an immense importance**.
- For example around 1970 only US government institutions needed to distribute daily tons of keys (on discs, tapes,...) to users they planned to communicate with.
- Big banks had special employees that used to travel all the time around the world and to deliver keys, in special briefcases, to everyone who had to get a message next week.
- Informatization of society was questioned because if governments had problems with key distribution how smaller companies could handle the key distribution problem without bankrupting?
- At the same time, the **key distribution problem** used to be considered, practically by all, as an **unsolvable problem**.

PADLOCKS

FIRST INGENIOUS IDEA - KEY PLAYERS

Whitfield Diffie (1944), graduated in mathematics in 1965, and started to be obsessed with the key distribution problem - he realized that whoever could find a solution of this problem would go to history as one of the all-time greatest cryptographers.

In 1974 Diffie convinced **Martin Hellman** (1945), a professor in Stanford, to work together on the key distribution problem - Diffie was his graduate student.

In 1975 they got a basic idea that the key distribution may not be needed that can be now illustrated as follows

A padlock protocol

- **If Alice wants to send securely a message to Bob, she puts the message into a box, locks the box with a padlock and sends the box to Bob.**
- **Bob has no key to open the box, so he uses another padlock to double-lock the box and sends this now doubly padlocked box back to Alice.**
- **Alice uses her key to unlock her padlock (but, of course, she cannot unlock Bob's padlock) and sends the box back to Bob.**
- **Bob uses his key to unlock his (now single) padlock and reads the message.**

Great idea was born. The problem then was to find a computational realization of this great idea. The first idea - to model locking of padlocks by doing an encryption.

FIRST ATTEMPT to DIGITALIZE THE PADLOCK PROTOCOL

Let us try to replace locking of padlocks by substitution encryptions.

Alice's encryption substitution.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
H	F	S	U	G	T	A	K	V	D	E	O	Y	J	B	P	N	X	W	C	Q	R	I	M	Z	L

Bob's encryption substitution.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
C	P	M	G	A	T	N	O	J	E	F	W	I	Q	B	U	R	Y	H	X	S	D	Z	K	L	V

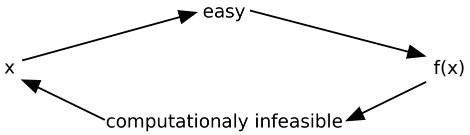
Message	m	e	e	t	m	e	a	t	n	o	o	n
Alice's encrypt.	Y	G	G	C	Y	G	H	C	J	B	B	J
Bob's encrypt.	L	N	N	M	L	N	O	M	E	P	P	E
Alice's decrypt.	Z	Q	Q	X	Z	Q	L	X	K	P	P	K
Bob's decrypt.	w	n	n	t	w	n	y	t	x	b	b	x

Observation The first idea does not work. Why?

A way out: **One-way functions(encryption substitutions)** are needed

ONE-WAY FUNCTIONS

Informally, a function $F : N \to N$ is said to be a **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.

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 $|x| = n > n_c$**

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

Candidates:

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

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

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

APPLICATION - COMPUTER PASSWORDS SECRECY PROBLEM

A **naive solution** to the password secrecy problem is to keep in computer a file with entries as

login CLINTON password BUSH,

that is a list of login names and corresponding passwords. This is obviously not **safe** enough.

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

login CLINTON password BUSH one-way function f_c

where BUSH is a "public" password and CLINTON is the only one that knows a "secret" password, say MADONNA, such that

$$f_c(\text{MADONNA}) = \text{BUSH}$$

PUBLIC ESTABLISHMENT of SECRET KEYS

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

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

Diffie-Hellman Protocol: If two parties, Alice and Bob, want to create a common secret key, then they first agree, somehow, on large primes p and a $q < p$ of large order in Z_p^* and then they perform, using 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 same (**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.

MERKLE JOINING DIFFIE-HELLMAN

After Diffie and Hellman announced their solution to the key generation problem, Ralph Merkle claimed, and could prove, that he had a similar idea some years ago.

That is the way why some people talk about Merkle-Diffie-Hellman key exchange.

MAN-IN-THE-MIDDLE ATTACKS

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

1. Eve chooses an integer (exponent) z.
2. Eve intercepts q^x and q^y – when they are sent from Alice to Bob and from Bob to Alice.
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.

GENERALISATION - BLOOM's KEY PRE-DISTRIBUTION PROTOCOL that

allows a trusted authority (Trent - TA) to distribute secret keys to $\frac{n(n-1)}{2}$ pairs of n users.

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

1. Each user U in the network is assigned, by Trent, a unique public number $r_U < p$.
2. Trent chooses three secret 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 agreed on secret-key cryptosystem.

SECURE COMMUNICATION with SECRET-KEY CRYPTOSYSTEMS

and without any need for secret key distribution

The idea contained in the above mention padlock protocol has been materialized by Shamir as follows:

(**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 too large number).

Advantage: It is a perfect protocol for distribution of secret keys.

BIRTH of PUBLIC KEY CRYPTOGRAPHY I

Diffie and Hellman demonstrated their discovery of the hey establishment protocol at the National Computer Conference in June 1976 and astonished the audience.

Next year they applied for a US-patent.

However, the solution of the key distribution problem through Diffie-Hellman protocol could still be seen as not good enough. Why?

The protocol required still too much communication and a cooperation of both parties for quite a time.

BIRTH of PUBLIC KEY CRYPTOGRAPHY II

Already in 1975 Diffie got the an idea for key distribution that seemed to be better: To design asymmetric cryptosystems - public key cryptosystems.

The basic idea was that in a public key cryptosystem not only the encryption and decryption algorithms would be public, but for each user U also the key e_U for encrypting messages (by anyone) for U would be public, and each user U would keep secret another key, d_U, that could be used for decryption of messages that were addressed to him and encrypted with the help of public encryption key e_U.

The realization that a cryptosystem does not need to be symmetric can be seen nowadays as the single most important breakthrough in modern cryptography.

Diffie published his idea in the summer of 1975 in spite of the fact that he had no idea how to design such a system.

To turn asymmetric cryptosystems from a great idea into a practical invention, somebody had to discover an appropriate mathematical function.

Mathematically, the problem was to find a simple enough one-way trapdoor function.

A search (hunt) for such a function started.

TRAPDOOR ONE-WAY FUNCTIONS

The key concept for design of public-key cryptosystems stsrted to be that of trapdoor one-way functions.

A function $f : X \rightarrow Y$ is a 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.**
- However, the inverse of f can be computed efficiently if some special, "trapdoor", knowledge is available.

New basic question: How to find such a (trapdoor one-way) function?

New basic idea: To make a clever use of outcomes of the computational complexity.

CRYPTOGRAPHY and COMPUTATIONAL COMPLEXITY

Modern cryptography uses such encryption methods that no "enemy" can have enough computational power and time to do decryption (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 an integer $n(=pq)$, 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" 232 digits integer RSA-768 (2009)

Primes recognition: Is a given n a prime? – fast randomized algorithms exist (1977). The existence of polynomial deterministic algorithms for primes recognition has been shown only in 2002

COMPUTATIONALLY INFEASIBLE PROBLEMS

Discrete logarithm problem: Given integers x, y, n, determine an 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, but easy if factorization of n is known

Knapsack problem: Given a (knapsack - integer) vector $X = (x_1, \ldots, x_n)$ and an (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 $x_i > \sum_{j=1}^{i-1} x_j, 1 < i \leq n$.

BIRTH of PUBLIC-KEY CRYPTOGRAPHY- II.

A **candidate for a one-way trapdoor function:** modular squaring $\sqrt{y} \bmod n$ with a fixed modulus n.

- computation of discrete square roots is unfeasible in general, but quite easy if the decomposition of the modulus **n** 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.

FORMAL VIEW of PUBLIC-KEY CRYPTOSYSTEMS

A public-key cryptosystem consists of three fixed and publically known deterministic algorithms:

- E — encryption algorithm;
- D — decryption alorithm;
- G — key-generation algorithm

In addition: the following binary words will be considered:

- M — message;
- C — cryptotext
- K_e — encryption key
- K_d — decription key
- X — trapdoor

Prior to transformation a message M of length n, the receiver generates X, say randomly, where $|X|$ is polynomial in n, and then computes the pair $(K_e, K_d) = G(X)$.

K_e is made public, but K_d and X are kept secret.

When a sender wants to send a message M of length n to the receiver, he computes $C = E(K_e, M)$ and sends C on a public channel. The receiver reconstructs M by computing $M = D(K_d, C)$.

It is also assumed that, for every X, if $(K_e, K_d) = G(X)$, then $M = D(K_d, E(K_e, M))$.

The realization that a cryptosystem does not need to be symmetric can be seen as the single most important breakthrough in the modern cryptography.

GENERAL, UNFEASIBLE, KNAPSACK PROBLEM

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 possible) such that $XB^T = c$.

However, the Knapsack problem with a superincreasing vector is 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 $i = n \leftarrow$ **downto 2 do**
 if $c \geq 2x_i$ **then** terminate $\{$no solution$\}$
  else if $c \geq x_i$ **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 and MCELIECE CRYPTOSYSTEMS

KNAPSACK ENCRYPTION – BASIC IDEAS

Let a (knapsack) vector (of integers)

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

be given.

Encryption of a (binary) message/plaintext $B = (b_1, b_2, \ldots, b_n)$ by A is done by the vector $\times$ 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 raw vector $X = (x_1, \ldots, x_n)$.
2. Choose integers **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: X' – public key
X, u, m – trapdoor information

Encryption: of a binary raw vector w of length n: $c = X'w^T$

Decryption: compute $c' = u^{-1}c \bmod m$
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 $X'w^T = c$. Then

$$c' \equiv u^{-1}c \equiv u^{-1}X'w^T \equiv u^{-1}uXw^T \equiv Xw^T (\text{mod } m).$$

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

$$(Xw^T) \bmod m = Xw^T$$

and therefore

$$c' = Xw^T.$$

DESIGN of KNAPSACK CRYPTOSYSTEMS – EXAMPLE

Example $\mathbf{X} = (1,2,4,9,18,35,75,151,302,606)$
$\mathbf{m} = 1250$, $u = 41$
$\mathbf{X'} = (41,82,164,369,738,185,575,1191,1132,1096)$

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

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

Encryption:

$$c_{1'} = X'w_1^T = 3061 \qquad c_{2'} = X'w_2^T = 2081 \qquad c_{3'} = X'w_3^T = 2203$$

Cryptotext: (3061,2081,2203)

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

By multiplying with $u^{-1} = 61 \pmod{1250}$ 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)

Therefore, the resulting plaintext is: ZIMBABWE

STORY of KNAPSACK

Invented: 1978 - Ralph C. Merkle, Martin Hellman
Patented: in 10 countries
Broken: 1982: Adi Shamir

New idea: to use iterated knapsack cryptosystem with **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 $Y = X_0, X_1, \ldots, X_k = X'$,

where X_0 is a super-increasing vector, and for $i = 1, \ldots, k$ X_i is obtained from X_{i-1} by a strong modular multiplication.

Iterated knapsack cryptosystem was broken **in 1985 - by E. Brickell**

New idea: to use **knapsack cryptosystems with dense vectors**. Density of a knapsack vector $X = (x_1, \ldots, x_n)$ is defined by $d(x) = \frac{n}{log(max\{x_i | 1 \leq i \leq n\})}$

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

KNAPSACK CRYPTOSYSTEM – COMMENTS

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

By **Knapsack problem** one usually understands the following problem:

Given n items with weights $w_1, w_2, \dots, w_n$, values $v_1, v_2, \dots, v_n$ and a knapsack limit c, the task is to find a bit vector $(b_1, b_2, \dots, b_n)$ 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 problems deployed in our construction of knapsack cryptosystems. It is well-known that the decision version of this problem is NP-complete.

For our version of the **knapsack problem** the term **Merkle-Hellman (Knapsack) Cryptosystem** is often used.

McEliece cryptosystem is based on a similar design principle as the Knapsack cryptosystem. McEliece cryptosystem is formed by transforming an easy to break cryptosystem (based on an easy to decode linear code) into a cryptosystem that is hard to break (because it seems to be based on a linear code 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 often 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: $P = (Z_2)^k$; cryptotexts: $C = (Z_2)^n$, key: $K = (G, S, P, G')$, message: w
G' is made public, G, S, P are kept secret.

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

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 $w = w_1 S^{-1}$

COMMENTS on McELIECE CRYPTOSYSTEM I

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 random generator matrix for a linear code for which the 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, the vector 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 than 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).

COMMENTS on McELIECE CRYPTOSYSTEM II

- Cryptosystem was invented in 1978 by Robert McEliece.
- Cryptosystem is a candidate for post-quantum cryptography - all attempts to break it using quantum computers failed.
- There are nowadays various variant of the cryptosystem that use different easy to decode linear codes. Some are known not to be secure.
- McEliece cryptosystem was the first public key cryptosystem that used randomness - a very innovative step.
- For a standard selection of parameters the public key is more than 521 000 bits long.
- That is why cryptosystem is rarely used in practise in spite of the fact that it has some advantages comparing with RSA cryptosystem discussed next - it has more easy encoding and decoding.

FINAL COMMENTS

1. **Deterministic public-key cryptosystems can never provide absolute 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 c such that $e_A(w) = c$.
2. One-way functions exist if and only if $\mathbf{P} = \mathbf{UP}$, 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.

RSA

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 security of RSA cryptosystems.

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.

HISTORY of RSA

- Diffie published his idea of asymmetric cryptosystem in summer 1975, though he had no example of such a cryptosystem.
- The problem was to find a one-way function with a backdoor.
- Rivest, Shamir and Adleman, from MIT, started to work on this problem in 1976.
- Rivest and Shamir spent a year coming up with new ideas and Adleman spent a year shooting them down.
- In April 1977 they spent a holiday (Pasover) evening drinking quite a bit of wine.
- At night Rivest could not sleep, mediated and all of sudden got an idea. In the morning the paper about a new cryptosystem, called now RSA, was practically written down.

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 randomly two large about s-bit primes p,q, where $s \in [512, 1024]$, 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 exponent)
Trapdoor information: p, q, d (decryption exponent)

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 $\{0, 1, \ldots, 9\}$, 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.

PROOF of the CORRECTNESS of RSA

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

$$n = pq, ed \equiv 1 \pmod{\phi(n)}, \gcd(d, \phi(n)) = 1$$

In such a case

$$w \equiv c^d \bmod n$$

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

Proof Since $ed \equiv 1 \pmod{\phi(n)}$, there exists a $j \in N$ 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 Totient Theorem we get that
  $$c^d = w^{ed} = w^{j\phi(n)+1} \equiv w \pmod{n}$$
- **Case 2. Exactly one of numbers p, q divides w – say p.**
  In such a case $w^{ed} \equiv w \pmod{p}$ and by Fermat's Little theorem $w^{q-1} \equiv 1 \pmod{q}$
  $$\begin{aligned} \Rightarrow w^{q-1} \equiv 1 \pmod{q} &\Rightarrow w^{\phi(n)} \equiv 1 \pmod{q} \\ &\Rightarrow w^{j\phi(n)} \equiv 1 \pmod{q} \\ &\Rightarrow w^{ed} \equiv w \pmod{q} \end{aligned}$$
  Therefore: $w \equiv w^{ed} \equiv c^d \pmod{n}$
- **Case 3.** Both p, q divide w.
  This cannot happen because, by our assumption, $w < n$.

HOW TO DO EFFICIENTLY RSA COMPUTATIONS

How to compute w^e mod n? Use the method of exponentiation by squaring - see the Appendix - and perform all operations modulo n

How to compute d^{-1} mod $\phi(n)$? :

Method 1 Use Extended Euclid algorithm, see the Appendix, that shows how to find, given integers $0 < m < n$ with $GCD(m, n) = 1$, integers x, y such that

$$xm + yn = 1$$

Once this is done, $x = m^{-1}$ mod n

Method 2 It follows from Euler's Totient Theorem that

$$m^{-1} \equiv m^{\phi(n)-1} \bmod \phi(n)$$

if $m < n$ and $GCD(m, n) = 1$

EXPONENTIATION by squaring

Exponentiation (modular) plays the key role in many cryptosystems. If

$$n = \sum_{i=0}^{k-1} b_i 2^i, \quad b_i \in \{0,1\}$$

then

$$e = a^n = a^{\sum_{i=0}^{k-1} b_i 2^i} = \prod_{i=0}^{k-1} a^{b_i 2^i} = \prod_{i=0}^{k-1} (a^{2^i})^{b_i}$$

Algorithm for exponentiation

begin $e \leftarrow 1$; $p \leftarrow a$;
    for $i \leftarrow 0$ **to** $k-1$
        do if $b_i = 1$ **then** $e \leftarrow e \cdot p$;
            $p \leftarrow p \cdot p$
        od
end

Modular exponentiation: $a^n \mod m = ((a \mod m)^n) \mod m$
Modular multiplication: $ab \mod n = ((a \mod n)(b \mod n) \mod n)$
Example $3^{1000} \mod 19 = 16$
$3^{10000} \mod 13 = 3$
$3^{340} \mod 11 = 1$
$3^{100} \mod 79 = 51$

Good values of the encryption exponent e should:

have:

- short bits length;
- small Hamming weight
- $e = 3, \;\; 17, \;\; 65.537 = 2^{16} + 1$

EXAMPLE of ENCRYPTION and DECRYPTION in RSA

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 exponents ($e = 23$ **and** $d = 2087$**).**

Plaintext: KARLSRUHE **First encoding (letters–int.):** 100017111817200704

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

$$100, 017, 111, 817, 200, 704$$

Encryptions:

$$100^{23} \bmod 2501, \quad 17^{23} \bmod 2501, \quad 111^{23} \bmod 2501$$
$$817^{23} \bmod 2501, \quad 200^{23} \bmod 2501, \quad 704^{23} \bmod 2501$$

provide cryptotexts:

$$2306, 1893, 621, 1380, 490, 313$$

Decryptions:

$$2306^{2087} \bmod 2501 = 100, 1893^{2087} \bmod 2501 = 17$$
$$621^{2087} \bmod 2501 = 111, 1380^{2087} \bmod 2501 = 817$$
$$490^{2087} \bmod 2501 = 200, 313^{2087} \bmod 2501 = 704$$

RSA CHALLENGE

The first public description of the RSA cryptosystem was in the paper.

Martin Gardner: Mathematical games, Scientific American, 1977

and in this paper the 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 inventors expected that to do encryption would require millions of years.

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

Abstract of the US RSA patent 4,405,829

The system includes a communication channel coupled to at least one terminal having an encoding device and to at least one terminal having a decoding device.

A message-to-be-transferred is enciphered to ciphertext at the encoding terminal by encoding a message as a number, M, in a predetermined set.

That number is then raised to a first predetermined power (associated with the intended receiver) and finally computed. The remainder of residue, C, is ... computed when the exponentiated number is divided by the product of two predetermined prime numbers (associated with the predetermined receiver).

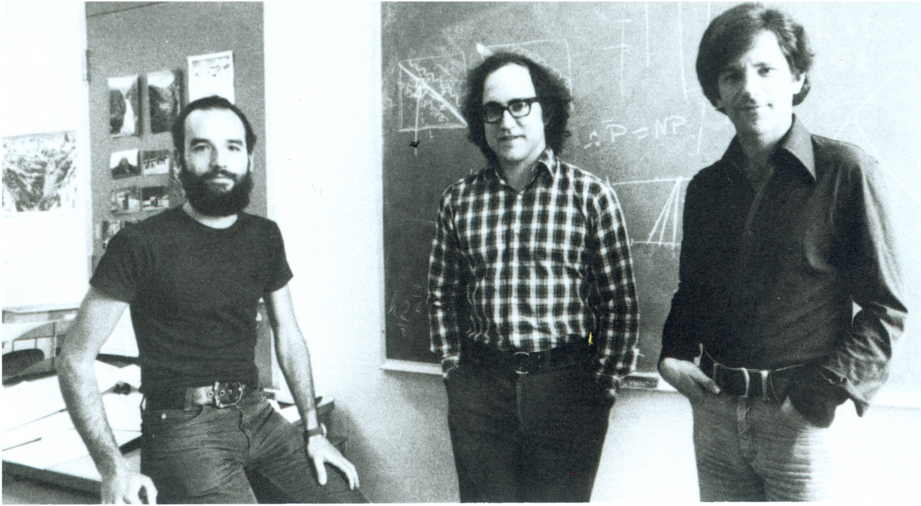
Security of RSA is based on the fact that for the following two problems no classical polynomial time algorithms seem to exist.

- Integer factorization problem.
- RSA problem: Given a public key (n, e) and a cryptotext c find an m such that $c = m^e \pmod n$.

HISTORY of RSA

- Diffie published his idea of asymmetric cryptosystem in summer 1975, though he had no example of such a cryptosystem.
- The problem was to find a one-way function with a backdoor.
- Rivest, Shamir and Adleman, from MIT, started to work on this problem in 1976.
- Rivest and Shamir spent a year coming up with new ideas and Adleman spent a year shooting them down.
- In April 1977 they spent a holiday evening drinking quite a bit of wine.
- At night Rivest could not sleep, mediated and all of sudden got an idea. In the morning the paper about RSA was practically written down.

Ron Rivest, Adi Shamir and Leonard Adleman



Copied from the brochure on LCS

PRIMES - key tools of modern cryptography

- A prime p is an integer with exactly two divisors - 1 and p.
- Primes play very important role in mathematics.
- Already Euclid new that there are infinitely many primes.
- Probability that an n-bit integer is prime is $\frac{1}{2.3n}$. (The accuracy of this estimate is closely related to the *Rieman Hypothesis* considered often as the most important open problem of mathematics.)
- Each integer has a uniquer decomposition as a product of primes.
- Golbach conjecture: says that every even integer n can be written as the sum of two primes (verified for $n \leq 4 \cdot 10^{14}$).
- Vinogradov Theorem: Every odd integer $n > 10^{43000}$ is the sum of three primes.
- There are fast ways to determine whether a given integer is prime or not.
- However, if an integer is not a prime then it is very hard to find its factors.

PRIMES PRIZES

Electronic frontiers foundation offered several prizes for record primes:

- In 1999 \$ 50,000 prize was given for first 1 million digits prime.
- In 2008 \$ 100,000 prize was given for first 10 million digits prime.
- A special prize is offered for first 100 million digits prime.
- Another special prize is offered for first 1 billion digits prime.

HOW to DESIGN REALLY GOOD RSA CRYPTOSYSTEMS?

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 $p+2, p+4, \ldots$ for primality.

   From the Prime Number Theorem it 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 nor too large.
   - 2.2 $\gcd(p-1, q-1)$ should not be large.
   - 2.3 Both $p-1$ and $q-1$ should not contain small 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 $n^{\frac{1}{4}}$. (For $d < n^{\frac{1}{4}}$ a polynomial time algorithm is known to determine d).

If $n = pq$ and $p - q$ is "small", then factorization can be quite easy.

For example, if $p - q < 2n^{0.25}$

(which for even small 1024-bit values of n is about $3 \cdot 10^{77}$)

then factoring of n is quite easy.

PRIMES RECOGNITION and INTEGERS FACTORIZATION

The key problems for the development of RSA cryptosystem are that of primes recognition and integers 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.

LARGEST PRIMES

Largest known prime so far is the Mersenne prime

$$2^{57,885,161} - 1$$

that has $17,425,170$ digits and was discovered on

25.1.2013 at 23.30.26 UTC

The last 15 record primes were also Mersenne primes (of the form $2^p - 1$).

Record was obtained by **Great Internet Mersenne Prime Search (GIMPS)** consortium established in 1997.

RABIN-MILLER's PRIME RECOGNITION

The fastest known sequential deterministic algorithm to decide whether a given integer n is prime has complexity $O\left((\lg n)^{14}\right)$

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

Lemma Let $n \in \mathbf{N}$, $n = 2^s d + 1, d$ is odd. Denote, for $1 \leq x < n$, by $C(x)$ the condition:

$$x^d \not\equiv 1 \pmod{n} \ \ \textit{and}\ x^{2^r d} \not\equiv -1 \pmod{n}\ \textit{for all}\ 1 < r < s$$

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

In other words most of the numbers between 1 and n are witnesses for composability of n. **Rabin-Miller algorithm**

- Choose randomly integers $x_1, \ldots, x_m$ such that $1 \leq x_j < n$;
- For each x_j determine whether $C(x_j)$ holds;
- **if** $C(x_j)$ holds for some x_j;
  then n is not prime
  else n is prime, with probability of error 2^{-m}

FACTORIZATION of 512-BITS and 708-BITS NUMBERS

On August 22, 1999, a team of scientists 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 financial 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-640 was factorized - this took approximately 30 2.2GHz-Opteron-CPU years - over five months of calendar time.

In 2009 RSA-768, a 768-bits number, was factorized by a team from several institutions. Time needed would be 2000 years on a single 2.2 GHz AND Opterons. Cash price obtained - 30 000 $.

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**$-10^{10^{100}}$

FACTORIZATION of very large NUMBERS

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

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

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

In 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.

DESIGN OF GOOD RSA CRYPTOSYSTEMS

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

Indeed, if $|p-q|$ is small, and $p>q$, then $\frac{(p+q)}{2}$ 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

$$p+q=2x, p-q=2y \qquad \text{and therefore } p=x+y, q=x-y.$$

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

Indeed, in the opposite case $s=\mathrm{lcm}(p-1,q-1)$ 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 $w^{ks}\equiv 1 \bmod p$ 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 $U_1, U_2, \ldots$ and also encryption exponents $e_1, e_2, \ldots$ 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.

- Breaking RSA encryptions is known as **RSA problem**..
- **RSA problem** Given cryptotext c, public key n, e find plaintext w such that $c = w^e \pmod{n}$.
- RSA problem is not equivalent to the integer factorization problem.
- Computation of the secret key exponent and factorization of moduli are equivalent problems.

CHOSEN CRYPTOTEXT ATTACK

To encrypt a cryptotext $c = w^e$ the attacker can ask the holder of the decryption exponent d to decrypt an innocently looking message $c' = cr^e$ for some value r chosen by the attacker.

In such a case c' is an encryption of wr. Indeed,

$$c' \equiv cr^e \equiv w^e r^e \equiv (wr)^e \bmod n$$

Hence, if w' is outcome of such a decryption, then

$$w = w' r^{-1} \bmod n$$

This attack is based on the fact that
$w_1^e w_2^e = (w_1 w_2)^e \bmod n$ -

SECURITY of RSA in PRACTICE

None of the numerous attempts to develop attacks on any RSA cryptosystem 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 produced by the RSA cryptosystem.

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

$$\textbf{parity}_{e_k}(c) = \text{the least significant bit of such an } w \text{ that } e_k(w) = c;$$
$$\textbf{half}_{e_k}(c) = 0 \text{ if } 0 \leq w < \tfrac{n}{2} \text{ and } \textbf{half}_{e_k}(c) = 1 \text{ if } \tfrac{n}{2} \leq w \leq n-1$$

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

$$\textbf{half}_{e_k}(c) = \textbf{parity}_{e_k}((c \times e_k(2)) \bmod n$$

$$\textbf{parity}_{e_k}(c) = \textbf{half}_{e_k}((c \times e_k(\frac{1}{2})) \bmod n$$

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

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

BREAKING RSA USING THE ORACLE half

Algorithm:

for $i = 0$ to $\lceil \lg n \rceil$ **do**
  $c_i \leftarrow \mathbf{half}_{e_k}(c)$; $c \leftarrow (c \times e_k(2)) \bmod n$
$u \leftarrow n$
for $i = 0$ to $\lceil \lg n \rceil$ **do**
  $m \leftarrow (i + u)/2$;
  if $c_i = 1$ **then** $i \leftarrow m$ **else** $u \leftarrow m$;
output $\leftarrow [u]$

The algorithm does the job. Indeed, in the first cycle

$$c_i = \mathbf{half}_{e_k}(c \times (e_k(2))^i) = \mathbf{half}_{e_k}(e_k(2^i w)),$$

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

$$\mathbf{half}_{e_k}(e_k(w)) = 0 \equiv w \in [0, \frac{n}{2})$$

$$\mathbf{half}_{e_k}(e_k(2w)) = 0 \equiv w \in [0, \frac{n}{4}) \cup [\frac{n}{2}, \frac{3n}{4})$$

$$\mathbf{half}_{e_k}(e_k(4w)) = 0 \equiv w \in [0, \frac{n}{8}) \cup [\frac{n}{4}, \frac{3n}{8}) \cup [\frac{n}{2}, \frac{5n}{8}) \cup [\frac{3n}{4}, \frac{7n}{8})$$

SECURITY of RSA in PRACTICE II

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 for a cryptotext y designed by RSA with the public key e, n is smaller than $\frac{n}{2}$.

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

Basic idea: algorithm H will 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 $\frac{n}{2}$.

Let us summarize answers all possible outcomes of tests imply:

yes, yes, yes	$0 < x < \frac{n}{8}$	no, yes, yes	$\frac{n}{2} < x < \frac{5n}{8}$
yes, yes, no	$\frac{n}{8} < x < \frac{n}{4}$	no, yes, no	$\frac{5n}{8} < x < \frac{3n}{4}$
yes, no, yes	$\frac{n}{4} < x < \frac{3n}{8}$	no, no, yes	$\frac{3n}{4} < x < \frac{7n}{8}$
yes, no, no	$\frac{3n}{8} < x < \frac{n}{2}$	no, no, no	$\frac{7n}{8} < x < n$

Let a message w be encoded with a modulus n and two encryption exponents e_1 and e_2 such that $\gcd(e_1, e_2) = 1$. Therefore

$$c_1 = w^{e_1} \bmod n, \qquad c_2 = w^{e_2} \bmod n;$$

Then

$$w = c_1^a c_2^b,$$

where, a, b are such that

$$a \cdot e_1 + b \cdot e_2 = 1$$

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. The public-key cryptosystem
- Example RSA and the most spread out secret-key cryptosystems 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.

APPENDIX I

KERBEROS

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

- To communicate with user B the user A asks TA for 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 $T + 1$.

KEY DISTRIBUTION versus KEY 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 to any other party (except possibly the TA).

RSA in PRACTICE

- 660-bits integers were already (factorized) broken in practice.
- 1024-bits integers are currently used as moduli.
- 512-bit integers can be factorized with a device costing 5 K $ in about 10 minutes.
- 1024-bit integers could be factorized in 6 weeks by a device costing 10 millions of dollars.

ATTACKS on RSA

RSA can be seen as well secure. However, this does not mean that under special circumstances some special attacks can not be successful. Two of such attacks are:

- The first attack succeeds in case the decryption exponent is not large enough.
  Theorem (Wiener, 1990) Let $n = pq$, where p and q are primes such that $q < p < 2q$ and let (n, e) be such that $de \equiv 1(\bmod \phi(n))$. If $d < \frac{1}{3}n^{1/4}$. then there is an efficient procedure for computing d.
- **Timing attack** P. Kocher (1995) showed that it is possible to discover the decryption exponent by carefully counting the computation times for a series of decryptions. Basic idea: Suppose that Eve is able to observes times Bob needs to decrypt several cryptotext s. Knowing cryptotext and times needed for their decryption, it is possible to determine decryption exponent.

CASES WHEN RSA IS EASY TO BREAK

- If an user U wants to broadcast a value x to n other users, using for a communication with a user P_i a public key (e, N_i), where e is small, by sending $y_i = x^e \bmod N_i$.
- If $e = 3$ and $2/3$ of the bits of the plaintext are known, then one can decrypt efficiently;
- If 25% of the least significant bits of the decryption exponent d are known, then d can be computed efficiently.
- If two plaintexts differ only in a (known) window of length $1/9$ of the full length and $e = 3$, one can decrypt the two corresponding cryptotext.
- Wiener showed how to get secret key efficiently if $n = pq$, $q < p < 2q$ and $d < \frac{1}{3}n^{0.25}$.

SECURITY POTENTIAL of McELIECE CRYPTOSYSTEM

McEliece cryptosystem is one of those cryptosystems that has not been yet shown to be breakable by quantum computers.

McEliece cryptosystem is not practical, because for the recommended security parameters the public key size is 2^{19} bits; and therefore its security was not much scrutinised.

Big problem of cryptography is to find practical public-key cryptosystem that could not be broken even with quantum computers.

Big question? What comes first, powerful quantum computers or practical public-key cryptosystem secure also against quantum computers.

COMPLEXITY of RSA

Let modulus be product of two s-bit primes.

- Setting cryptosystem requires $O(s^4)$ operations.
  - generation of two s bit primes requires $O(s^4)$;
  - multiplication of primes p and q requires $O(s^2)$;
  - finding exponent e requires one GCD-computation - $O(s^2)$;
  - finding exponent d requires computation of generalized GCD - $O(s^2)$.
- Encryption requires one exponentiation - $O(s^3)$;
- Decryption requires one exponentiation - $O(s^3)$.

IMPLICATIONS of HARD INVERTABILITY of RSA

- Under the assumption that RSA is hard to invert we can design:
  - cryptographically perfect pseudorandom generators;
  - zero-knowledge proofs for any **NP** statement;
  - multiparty protocols for computing securely any multi-variant function.
- The fact that RSA is hard to invert does not imply that RSA is secure cryptosystem.

- Imad Khaled Selah, Abdullah Darwish, Saleh Ogeili: Mathematical attacks on RSA Cryptosystem, Journal of Computer Science 2 (8) 665-671, 2006
- Dan Boneh: Twenty years of attacks on RSA Cryptosystems, crypto.stanford.edu/ Dabo/pubs/papers/RSA-surwey.pdf