5. PUBLIC-KEY CRYPTOGRAPHY

Much increased needs for flexible and secure transmission of information require to use new cryptographic methods.

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

In secret-key (symetric key) cryptography both sender and receiver share the same secret key.

In public-key ryptography there two keys: a public encryption key amd a secret decryption key at the receiver.

BASIC IDEAS and TOY EXAMPLES

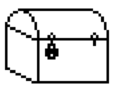
Basic idea: If it is infeasible from the knowledge of an encryption algorithm e_k to construct the corresponding description algorithm d_k, then e_k can be made public.

Toy example (Telephone directory encryption)
Each user U_k makes public a unique telephone directory td$_k$ to encrypt messages for U_k and U_k is the only user to have an inverse telephone directory itd$_k$.

Encryption: Each letter X of a plaintext w is replaced, using the telephone directory td$_k$ of the intended receiver U_k, by the telephone number of a person whose name starts with X.

Decryption: easy for U_k, 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 ones, are freely available.
- Only the legitimate user has the key from his/her padlocks.

1. 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.

Protocol: If two parties, Alice and Bob, want to create a common secret key, then they first agree, somehow, on large prime p and a primitive root q $(\mathrm{mod} p)$ and they perform through 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, 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 knows 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, to have a capability to compute discrete logarithms, or to compute q^{xy} from q^x and q^y, what is believed to be infeasible.

MAN-IN-THE-MIDDLE ATTACK

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

1. Eve chooses an exponenet z.
2. Eve intercepts q^x and q^y.
3. Eve sends q^z to both Alice and Bob. (After that Alice believes she is receiving q^x and Bob believes he is receiving q^y.)
4. Eve computes $K_A = q^{xz} (\mathrm{mod} p)$ and $K_B = q^{yz} (\mathrm{mod} p)$. Alice, not realizing that Eve is in the middle, also computes K_A and Bob computes K_B.
5. When Alice sends a message to Bob, encrypted with K_A, Eve intercepts it, decrypts it, and encrypts it with K_B, and sends it to Bob.
6. Bob decrypts the mesage 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

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

Let large prime $p > n$ be publically known. The protocol has the following steps:

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
$$a_U = (a + br_U) \pmod{p}, \qquad b_U = (b + cr_U) \pmod{p}$$
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 communicate with Bob (B), then Alice computes $K_{AB} = g_A(r_B)$ and Bob computes $K_{BA} = g_B(r_A)$.
6. It is easy to see that $K_{AB} = K_{BA}$ and therefore Alice and Bob can use their keys to communicate using some secret-key cryptosystem.

Secure communication with the secret-key cryptosystem but without any need for 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 are commutative (to form a commutative cryptosystem).

Communication protocol

in which Alice wants to 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: 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 for 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 of the "small" modifications of these problems, surprisingly, simple, fast and good enough (randomized) algorithms do exist.

Integer factorization: Given $n(=pq)$, find p, q – unfeasible.

There is a list of "most wanted to factor integers". Top current success using thousands of computers for months.

($\star$) Factorization of $2^{2^9} + 1$ with 155 digits (1996)
($\star\star$) Factorization of a "typical" 155-digits integer (1999)

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

Discrete logarithm problem: Given x, y, n, compute a such that $y \equiv x^a (\bmod n)$ – unfeasible.

Discrete square root problem: Given y, n, compute x such that $y \equiv x^2 (\bmod n)$ – infeasible in general, easy if n is prime.

Knapsack problem: Given a knapsack vector $X = (x_1, \ldots, x_n)$ and knapsack capacity c, find 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.

$$\begin{tikzcd} x \arrow[rr, bend left, "\text{easy}"] & & f(x) \arrow[ll, bend left, "\text{computation infeasible}"] \end{tikzcd}$$

A more formal approach

Definition A function $f : \{0,1\}^\star \to \{0,1\}^\star$ 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 $\mathcal{A}$, and any constant $c > 0$, there exists an n_c such that for $n > n_c$

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

Candidates: Modular exponentiation: $f(x) = a^x \bmod n$
Modular squaring $f(x) = x^2 \bmod n$, n – a Blum integer
Prime number multiplication $f(p, q) = pq$.

Trapdoor One-way Functions

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

A function $f : X \rightarrow Y$ 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 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.

One way to design a trapdoor one-way function is to transform (using diffusions/confusions) an easy case of a hard (one-way) function to a hard-looking case that can be solved easily for those knowing how the transformation was performed.

Example — Computer passwords

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

login CLINTON password BUSH,

that with logins and corresponding passwords is not sufficiently safe.

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

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 “secret” password, say MADONA, such that

$$f_c(MADONA) = BUSH$$

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

$$XB^\top = c.$$

Knapsack problem with superincreasing vector – easy

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

Determine a binary vector $B = (b_1, \ldots, b_n)$ (if it exists) such that

$$XB^\top = c.$$

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.

An **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

A 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

$$Ab^T =$$

Another view of the knapsack problem

Each **knapsack vector**

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

defines an integer valued
knapsack-function

$$f_A : \{x | 0 \leq x < 2^n\} \rightarrow N$$

specified as follows:

$$f_A(x) = \sum_{\substack{i\text{-th bit in the binary} \\ \text{representation of } x \text{ is } 1}} a_i$$

Example

$$A_0 = (43, 129, 215, 473, 903, 302, 561, 1165, 697, 1523)$$

$$f_{A_0}(364) = f_{A_0}(0101101100) = 129 + 473 + 903 + 561 + 1165 = 3231$$

Unambiguity of knapsack systems

For unambiguity in the decryption of the knapsack cryptosystems with knapsack vector f_A, it is important that

$$f_A(x_1) \neq f_A(x_2) \quad \text{if } x_1 \neq x_2.$$

Example $A = (17, 103, 50, 81, 33)$
Cryptotext:

$(131, 33, 100, 234, 33)$

$SAUNA$ $FAUNA$

$plaintexts$

Design of knapsack cryptosystems

1. Choose a superincreasing vector $X = (x_1, \ldots, x_n)$.
2. Choose m, u such that $m > 2x_n, \gcd(m, u) = 1$.
3. Compute $u^{-1} \bmod m, X' = (x'_1, \ldots, x'_n), x'_i = \underbrace{\underbrace{ux_i}_{diffusion} \bmod m}_{confusion}$.

Cryptosystem: X' – public key
X, u, m – trapdoor information

Encryption: of a binary vector w of length n:

$$c = X'w$$

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

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

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

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

and therefore

$$c' = Xw.$$

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 encode 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: AFRICA results in vectors

$$w_1 = (0000100110) \quad w_2 = (1001001001) \quad w_3 = (0001100001)$$
$$c'_1 = X'w_1 = 3061 \quad c'_2 = X'w_2 = 2081 \quad c'_3 = X'w_3 = 2203$$

Cryptotext: $(3061, 2081, 2203)$

Decryption: cryptotext:

$$(2163, 2116, 1870, 3599)$$

by multiplying with $u^{-1} = 61 \bmod 1250$ we get

$$(693, 326, 320, 789)$$

and in the binary form we have

$$(1101001001, 0110100010, 0000100010, 1011100101)$$

that is the corresponding 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

$$d(x) = \frac{n}{\log(\max\{x_i | 1 \le i \le n\})}$$

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

Breaking knapsack

Basic ideas of Shamir's polynomial time algorithm (in the length of the knapsack vector) to break knapsack cryptosystems.

Assumption: there is a $d > 1$ such that modulus m has $\lceil dn \rceil$ bits and elements $a_i, 1 \leq i \leq n$, of a superincreasing vector, have $\lceil dn \rceil - 1 - n + i$ bits.

(This implies that A is a superincreasing vector and $m > \Sigma_{i=1}^{n} a_i$.)
(Original suggestion: $d = 2, n = 100$.)

Key observation: Given a knapsack vector B, which was obtained from a super-increasing vector A through a strong modular multiplication using m and u, it is not important for successful cryptoanalysis to find original A, m, u. It is enough to find a pair (m', u') such that

(1) the vector A' obtained is superincreasing

(2) $m' > \Sigma\, a_i, t = u^{-1} \bmod m' < m, \gcd(m', t) = 1$

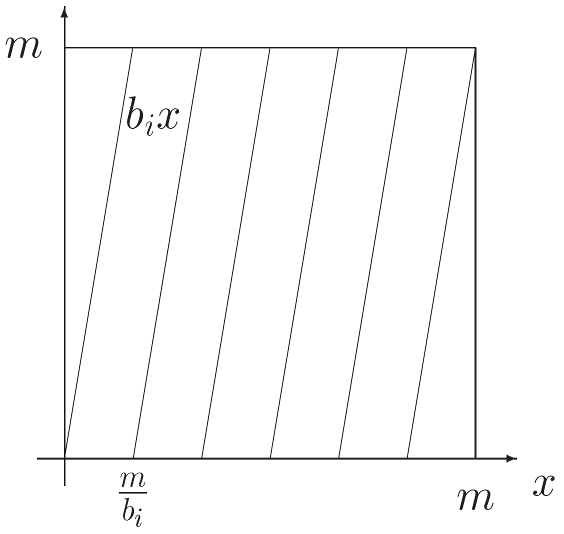
Such a pair is called a trapdoor pair.

To find a trapdoor pair one can proceed as follows:
One consider functions

$$b_i x \bmod m, 1 \leq i \leq n$$

Minimums are in points (discontinuation points)

$$\frac{jm}{b_i}, j \in \mathbb{N}$$



sawtooth curves

We need to find out t and m such that:

$$a_i \equiv b_i t \bmod m$$

and $(a_1, \ldots, a_n)$ is a superincreasing vector.

Since a_1 has to be very small comparing to m, t has to be close to some of the minima of the b_1-graph!

Similarly t has to be close to some minimum of the b_2-graph.

This implies that two minima of the b_1 and b_2-graphs must be close to one another.

Similarly we can consider more b_i-graphs.

The fact that the trapdoor pair value of t is close to a minimum on each b_i-graph implies that all these minima are close to one another.

Thus, instead of trying to find t itself, we try to find out "accumulation points" of the minima of b_i-graphs.

This amounts to constructing a small interval containing a minimum of each b_i-graph, and from this to find a trapdoor value of t.

Experiments show that it suffices to analyze only four b_i-graphs to get a desirable small interval containing t.

The task is now to express the above ideas in terms of inequalities.

The first problem is that also m is unknown. This is easy to deal with.

We reduce the size of figures for b_i-graphs so m becomes 1. This does not change which of the minima are close to another.

The algorithm for finding a trapdoor pair consists of two parts:

1. Candidates are found for an integer p such that the p-th minimum of the b_1-curve is an accumulation point we are looking for.
2. Candidates are tests one by one. One of the candidates has to succeed.

One problem is that the first stage may produce too many candidates.

To deal with this problem an integer r is fixed in advance and if the first stage produces more than r candidates the algorithm terminates and reports failure.

KNAPSACK CRYPTOSYSTEM - COMMENTS

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

By the **Knapsack problem** it is mostly understood the following problem:

Given n items with weights $w_1, w_2, \dots, w_n$ and 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 $\Sigma_{i=1}^{n} b_i w_i \leq c$ and $\Sigma_{i=1}^{n} b_i v_i$ is as large as possible.

The term **subset problem** is usually used for the problem used for 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

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

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 $m = 10$, $t = 50$.)

DESIGN, ENCRYPTION, DECRYPTION

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

Design of McEliece cryptosystems. Let

- G be an $k \times n$ generating matrix for an $[n,k,d]$ Goppa code C;
- S be a random $k \times k$ binary matrix invertible over $\mathbf{Z}_2$;
- P be a random $n \times n$ permutation matrix;
- $G' = SGP$.

Let $\mathbf{K} = \{(G, S, P, G')\}$.

G' is a public key, G, S, P form secret key.

Encryption of a plaintext $w \in \{0,1\}^k$: $c = e_K(w, e) = wG' + e$, where e is a binary random vector of length n and weight t.

Decryption of a cryptotext: $c \in \{0,1\}^n$, where $c = wG' + e$ for some w and e.

1. Compute $c_1 = cP^{-1} = wSGPP^{-1} + eP^{-1} = wSG + eP^{-1}$.
2. Decode c_1 to get w_1 such that $c_1 = w_1 + eP^{-1}$
3. Compute $w_0 \in \{0,1\}^k$ such that $w_0 G = w_1$, $w_0 = wS$.
4. Compute $w = w_0 S^{-1}$.

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 $\mathbf{NP}$-problem.
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 provided unconditional security. This is because an eavesdropper, on observing a cryptotext c sent to an user A can encrypt each posible plaintext by the encryption algorithm e_A until he finds such an c that $e_A(w) = c$.
2. One-way functions exitsts if and only if $\mathbf{P} = \mathbf{UP}$, where $\mathbf{UP}$ is the class of languages accepted by **unambiguous polynomial time bounded nondeterministic Turing machine**.
3. There are actually two types of keys: 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 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 Alice uses it with ONE-TIME PAD.

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