Part I

Public-key cryptosystems II. Other cryptosystems and cryptographic primitives

CHAPTER 6: OTHER CRYPTOSYSTEMS and BASIC CRYPTOGRAPHY PRIMITIVES

A large number of interesting and important cryptosystems have already been designed. **In this chapter we present several other of them in order to illustrate other principles and techniques that can be used to design cryptosystems.**

At first, we present several cryptosystems security of which is based on the fact that computation of square roots and discrete logarithms is in general unfeasible in some groups.

Secondly, we discuss one of the fundamental questions of modern cryptography: when can a cryptosystem be considered as (computationally) perfectly secure?

In order to do that we will:

- discuss the role randomness play in the cryptography;
- introduce the very fundamental definitions of perfect security of cryptosystem;
- present some examples of perfectly secure cryptosystems.

Finally, we will discuss, in some details, such very important cryptography primitives as **pseudo-random number generators** and **hash functions** .

FROM THE APPENDIX

STORY of SQUARE ROOTS

and

QUADRATIC RESIDUES

MODULAR SQUARE ROOTS PROBLEM

The problem is to determine, given integers y and n, such an integer x that

$$y = x^2 \bmod n.$$

Therefore the problem is to find square roots of y modulo n

Examples

$\{x \mid x^2 = 1 \pmod{15}\} = \{1, 4, 11, 14\}$
$\{x \mid x^2 = 2 \pmod{15}\} = \emptyset$
$\{x \mid x^2 = 3 \pmod{15}\} = \emptyset$
$\{x \mid x^2 = 4 \pmod{15}\} = \{2, 7, 8, 13\}$
$\{x \mid x^2 = 9 \pmod{15}\} = \{3, 12\}$

No polynomial time algorithm is known to solve the modular square root problem for arbitrary modulus n.

However, in case n is a prime or a product of two odd primes, such a polynomial squaring algorithm exists to solve modular square root problem..

QUADRATIC RESIDUES

Let $+_n, \times_n$ denote addition and multiplication modulo n

$$a +_n b = (a + b) \mod n, \quad a \times_n b = (ab) \mod n$$

$Z_n = \{0, 1, \dots, n-1\}$ is a group under the operation $+_n$
$\mathbf{Z}_n^\star = \{x \,|\, 1 \leq x \leq n, \gcd(x, n) = 1\}$ is a group under the operation $\times_n$

$Z_n^\star$ is a field under the operations $+_n, \times_n$, if n is a prime.

Theorem: For any n, the multiplicative inverse of any $z \in Z_n^\star$ and exponentiation in $Z_n^\star$ can be computed in polynomial time.
Definition: An integer $x \in Z_n^\star$ is called a **quadratic residue** modulo n if

$$x \equiv y^2 \ (mod n)$$

for some $y \in Z_n^\star$, otherwise x is called a **quadratic nonresidue**.

Notation: QR(n) – the set of all quadratic residues modulo n. $QR(n)$ is therefore subgroup of squares in $\mathbf{Z}_n^\star$.
QNR(n) – the set of all quadratic nonresidues modulo n.
For any prime p the set QR(p) has $\frac{p-1}{2}$ elements.

So called **Euler criterion** says that c is a quadratic residue modulo prime p iff

$$c^{(p-1)/2} \equiv 1 \ (\text{mod } p).$$

EXAMPLES of $Z_N^\star$ SETS and THEIR MULTIPLICATION TABLES

Z_9^*={1,2,4,5,7,8}

*	1	2	4	5	7	8
1	1	2	4	5	7	8
2	2	4	8	1	5	7
4	4	8	7	2	1	5
5	5	1	2	7	8	4
7	7	5	1	8	4	2

Z_{11}^*={,1,2,3,4,5,6,7,8,9,10}

*	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	1	3	5	7	9
3	3	6	9	1	4	7	10	2	5	8
4	4	8	1	5	9	2	6	10	3	7
5	5	10	4	9	3	8	2	7	1	6
6	6	1	7	2	8	3	9	4	10	5
7	7	3	10	6	2	9	5	1	8	4
8	8	5	2	10	7	4	1	9	6	3
9	9	7	5	3	1	10	8	6	4	2
10	10	9	8	7	6	5	4	3	2	1

EXAMPLE of $Z_N^\star$ SETS and THEIR QUADRATIC RESIDUES

To get all quadratic residues $QR(n)$ of $Z_N^\star$ we need to compute squares of all elements in $Z_n^\star$.

If $n = 8$ **then** $Z_8^\star = \{1, 3, 5, 7\}$
$1^2 \equiv 1(\mod 8)$, $3^2 \equiv 1(\mod 8)$,
$5^2 \equiv 1(\mod 8)$, $7^2 \equiv 1(\mod 8)$

$$QR(8) = \{1\}$$

If $n = 9$ **then** $Z_9^\star = \{1, 2, 4, 5, 7, 8\}$
$1^2 \equiv 1(\mod 9)$, $2^2 \equiv 4(\mod 9)$, $4^2 \equiv 7(\mod 9)$,
$5^2 \equiv 7(\mod 9)$, $7^2 \equiv 4(\mod 9)$, $8^2 \equiv 1(\mod 9)$

$$QR(9) = \{1, 4, 7\}$$

If $n = 15$ **then** $Z_{15}^\star = \{1, 2, 4, 7, 8, 11, 13, 14\}$
$1^2 \equiv 1(\mod 15)$, $2^2 \equiv 4(\mod 15)$, $4^2 \equiv 1(\mod 15)$,
$7^2 \equiv 4(\mod 15)$, $8^2 \equiv 4 \mod 15$,
$11^2 \equiv 1(\mod 15)$, $13^2 \equiv 4(\mod 15)$, $14^2 \equiv 1(\mod 15)$

$$QR(15) = \{1, 4\}$$

BLUM PRIMES and INTEGERS

If p, q are primes such that $p \equiv 3 \pmod 4$, $q \equiv 3 \pmod 4$ then they are called **Blum primes** and the integer $n = pq$ is called **Blum integer**

Blum integers n have the following important properties.

- If $x \in QR(n)$, then x has exactly four square roots and exactly one of them is in QR(n) – this square root is called **primitive square root** of x modulo n.
- Function $f : QR(n) \to QR(n)$ defined by $f(x) = x^2$ is a permutation on $QR(n)$.
- The inverse function is $f^{-1}(x) = x^{((p-1)(q-1)+4)/8} \mod n$

EXAMPLE

For $n = 21 = 3 \times 7$

$$Z_{21}^{*} = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$$

$$QR(21) = \{1, 4, 16\}$$

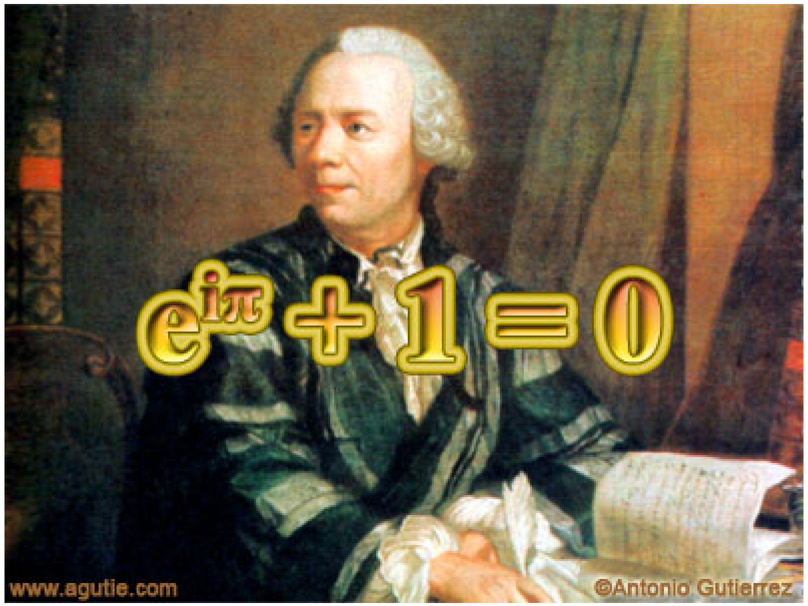
and

$$1^2 = 1 \bmod 21 \quad 4^2 = 16 \bmod 21 \quad 16^2 = 4 \bmod 21$$

DISCRETE SQUARE ROOTS CRYPTOSYSTEMS

DISCRETE SQUARE ROOTS CRYPTOSYSTEMS

Leonhard EULER - picture



Leonhard EULER

- Russian, one of the greatest mathematicians of the world, who worked in many areas of mathematics especially with applications to physics.
- Euler lectured also on astronomy, philosophy and religion.
- He worked also on problems of insurance, design of channels and waterworks.
- Euler was extremely productive: his papers fill 75 large volumes
- He used to produce about 800 pages of papers per year and it took 50 years after his death to publish all his papers.
- Since 1735 he could see only on one eye and the last 12 years he was totally blind and as such wrote 400 papers by dictating them to his children.
- Euler had phenomenal memory.
- He took great care of his 13 children.

DISCRETE SQUARE ROOTS CRYPTOSYSTEMS

DISCRETE SQUARE ROOTS CRYPTOSYSTEMS

RABIN CRYPTOSYSTEM

Let Blum primes p, q are kept secret, and let the Blum integer $n = pq$ be the public key.
Encryption: of a plaintext $w < n$

$$c = w^2 \ (mod \ n)$$

Decryption: -briefly

It is easy to verify (using Euler's criterion which says that if c is a quadratic residue modulo p, then $c^{(p-1)/2} \equiv 1 \pmod p$,) that

$$\pm c^{(p+1)/4} mod \ p \quad \text{and} \quad \pm c^{(q+1)/4} mod \ q$$

are two square roots of c modulo p and q. (Indeed, $\frac{p+1}{2} = \frac{p-1}{2} + 1$) One can now obtain four square roots of c modulo n using the method of Chinese remainder shown in the Appendix.

In case the plaintext w is a meaningful English text, it should be easy to determine w from the four square roots w_1, w_2, w_3, w_4 presented above.

However, if w is a random string (say, for a key exchange) it is impossible to determine w from w_1, w_2, w_3, w_4.

That is, likely, why Rabin did not propose this system as a practical cryptosystem.

COMPUTATION of SQUARE ROOTS MODULO PRIMES

In case of Blum primes p and q and Blum integer $n = pq$, in order to solve the equation $x^2 \equiv a (\text{mod} \ n)$, one needs to compute squares of a modulo p and modulo q and then to use the Chinese remainder theorem to solve the equation $x^2 = a \pmod{pq}$.

Example To solve modular equation $x^2 \equiv 71 \pmod{77}$,one needs to solve modular equation

$$x^2 \equiv 71 \equiv 1 \pmod 7 \text{ to get } x \equiv \pm 1 (\text{mod } 7)$$

and
to solve also modular equation

$$x^2 \equiv 71 \equiv 5 \pmod{11} \text{ to get } x \equiv \pm 4 \pmod{11}.$$

Using the Chinese Remainder Theorem we then get

$$x \equiv \pm 15, \pm 29 \pmod{77}.$$

CHINESE REMAINDER THEOREM

Theorem Let $m_1, \ldots, m_t$ be integers, $\gcd(m_i, m_j) = 1$ if $i \neq j$, and $a_1, \ldots, a_t$ be integers such that $0 < a_i < m_i, 1 \leq i \leq t$.
Then the system of congruences

$$x \equiv a_i \ (mod \ m_i), 1 \leq i \leq t$$

has the solution

$$x = \sum_{i=1}^{t} a_i M_i N_i \qquad (\star)$$

where

$$M = \prod_{i=1}^{t} m_i, M_i = \frac{M}{m_i}, N_i = M_i^{-1} \quad \text{mod } m_i$$

and the solution $(\star)$ is unique up to the congruence modulo M.

Application Each integer $0 < x < M$ is uniquely represented by t-tuple:

$$x \ (mod \ m_1), \ldots, x \ (mod \ m_t).$$

Example If $m_1 = 2, m_2 = 3, m_3 = 5$, then $(1, 0, 2)$ represents integer 27.
Advantage: With such a modular representation addition, subtraction and multiplication can be done component-wise and therefore in parallel time.

GENERALIZED RABIN CRYPTOSYSTEM

Public key: n, B $(0 \le B < n)$

Trapdoor: Blum primes p, q $(n = pq)$

Encryption: $e(x) = x(x + B) \bmod n$

Decryption: $d(y) = \left(\sqrt{\frac{B^2}{4} + y} - \frac{B}{2}\right) \bmod n$

It is easy to verify that if ω is a nontrivial square root of 1 modulo n, then there are four decryptions of $e(x)$:

$$x, \quad -x, \quad \omega\left(x + \tfrac{B}{2}\right) - \tfrac{B}{2}, \quad -\omega\left(x + \tfrac{B}{2}\right) - \tfrac{B}{2}$$

Example

$e\left(\omega\left(x + \frac{B}{2}\right) - \frac{B}{2}\right) = \left(\omega\left(x + \frac{B}{2}\right) - \frac{B}{2}\right)\left(\omega\left(x + \frac{B}{2}\right) + \frac{B}{2}\right) = \omega^2\left(x + \frac{B}{2}\right)^2 - \left(\frac{B}{2}\right)^2 = x^2 + Bx = e(x)$

Decryption of the generalized Rabin cryptosystem can be reduced to the decryption of the original Rabin cryptosystem.

Indeed, the equation $x^2 + Bx \equiv y \pmod n$ can be transformed,
by the substitution $x = x_1 - B/2$, into $x_1^2 \equiv B^2/4 + y \pmod n$
and, by defining $c = B^2/4 + y$, *into* $x_1{}^2 \equiv c \pmod n$
Therefore decryption can be done by factoring n and solving congruences

$$x_1{}^2 \equiv c \pmod p \qquad x_1{}^2 \equiv c \pmod q$$

SECURITY of RABIN CRYPTOSYSTEM

We show that any hypothetical decryption algorithm **A** for Rabin cryptosystem, can be used, as an oracle, in the following randomized algorithm, to factor an integer n.

Algorithm:

1. Choose a random $r, 1 \le r < n$;
2. Compute $y = (r^2 - B^2/4) \bmod n$; $\{y = e_k(r - B/2)\}$.
3. Call $A(y)$, to obtain a decryption $x = \left(\sqrt{\frac{B^2}{4} + y} - \frac{B}{2}\right) \bmod n$;
4. Compute $x_1 = x + B/2$; $\{x_1^2 \equiv r^2 \bmod n\}$
5. **if** $x_1 = \pm r$ **then quit** (failure)
   else $\gcd(x_1 + r, n) = p$ or q

Indeed, after Step 4, either $x_1 = \pm r \bmod n$ or $x_1 = \pm\omega r \bmod n$.
In the second case we have

$$n \mid (x_1 - r)(x_1 + r),$$

but n does not divide any of the factors $x_1 - r$ or $x_1 + r$.
Therefore computation of $\gcd(x_1 + r, n)$ or $\gcd(x_1 - r, n)$ must yield factors of n.

DISCRETE LOGARITHM CRYPTOSYSTEMS

DISCRETE LOGARITHM CRYPTOSYSTEMS

ElGamal CRYPTOSYSTEM

Design: choose a large prime p – (with at least 150 digits).
choose two random integers $1 \le q, x < p$ – where q is a primitive element of Z^*_p
calculate $y = q^x \bmod p$.

Public key: p, q, y; trapdoor: x

Encryption of a plaintext w: choose a random r and compute

$$a = q^r \bmod p, \qquad b = y^r w \bmod p$$

Cryptotext: $c = (a, b)$
(Cryptotext contains indirectly r and the plaintext is "masked" by multiplying with y^r (and taking modulo p))

Decryption: $w = \frac{b}{a^x} \bmod p = ba^{-x} \bmod p$.

Proof of correctness: $a^x \equiv q^{rx} \bmod p$

$$\frac{b}{a^x} \equiv \frac{y^r w}{a^x} \equiv \frac{q^{rx} w}{q^{rx}} \equiv w \pmod p$$

Note: Security of the ElGamal cryptosystem is based on infeasibility of the discrete logarithm computation.

SHANKS' ALGORITHM for DISCRETE LOGARITHM

Let $m = \lceil \sqrt{p-1} \rceil$. The following algorithm computes $\lg_q y$ in $Z^*{}_p$.

1. Compute $q^{mj} \bmod p$, $0 \leq j \leq m-1$.
2. Create list L_1 of m pairs $(j, q^{mj} \bmod p)$, sorted by the second item.
3. Compute $yq^{-i} \bmod p$, $0 \leq i \leq m-1$.
4. Create list L_2 of pairs $(i, yq^{-i} \bmod p)$ sorted by the second item.
5. Find two pairs, one $(j,z) \in L_1$ and $(i,z) \in L_2$ with identical second element

If such a search is successful, then

$$q^{mj} \bmod p = z = yq^{-i} \bmod p$$

and as the result

$$q^{mj+i} \equiv y \pmod p$$

On the other hand, for any y we can write

$$\lg_q y = mj + i,$$

for some $0 \leq i, j < m$. Hence the search in the Step 5 of the algorithm has to be successful.

GROUP VERSION of ElGamal CRYPTOSYSTEM

A group version of discrete logarithm problem

Given a group $(G, \circ)$, $\alpha \in G$, $\beta \in \{\alpha^i \mid i \geq 0\}$. Find

$$\log_\alpha \beta = k \text{ such that } \alpha^k = \beta \text{ that is } k = \log_\alpha \beta$$

GROUP VERSION of ElGamal CRYPTOSYSTEM

ElGamal cryptosystem can be implemented in any group in which discrete logarithm problem is infeasible.

Cryptosystem for $(G, \circ)$
Public key: α, β
Trapdoor: k such that $\alpha^k = \beta$

Encryption: of a plaintext w and a random integer r

$$e(w,k) = (y_1, y_2) \quad \text{where} \quad y_1 = \alpha^r, y_2 = w \circ \beta^r$$

Decryption: of cryptotext (y_1, y_2):

$$d(y_1, y_2) = y_2 \circ y_1^{-k}$$

FEISTEL ENCRYPTION/DECRYPTION SCHEME

This is a general scheme for design of cryptosystems that was used at the design of several important cryptosystems, such as Lucifer and DES.
Its main advantage is that encryption and decryption are very similar, and even identical in some cases, and then the same hardware can be used for both encryption and decryption.
Let F a be a so-called round function and $K_0, K_1, \ldots, K_n$ be sub-keys for rounds $0, 1, 2, \ldots, n$.
Encryption is as follows:

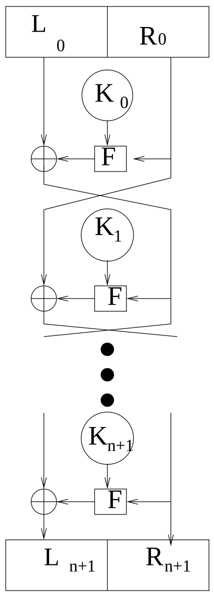
- Split the plaintext into two equal size parts L_0, R_0.
- For rounds $i \in \{0, 1, \ldots, n\}$ compute

$$L_{i+1} = R_i; \; R_{i+1} = L_i \oplus F(R_i, K_i)$$

then the ciphertext is (R_{n+1}, L_{n+1})
Decryption of (R_{n+1}, L_{n+1}) is done by computing, for $i = n, n-1, \ldots, 0$

$$R_i = L_{i+1}, L_i = R_{i+1} \oplus F(L_{i+1}, K_i)$$

and (L_0, R_0) is the plaintext



WHEN ARE ENCRYPTIONS PERFECTLY SECURE?

WHEN ARE ENCRYPTIONS PERFECTLY SECURE?

RANDOMIZED ENCRYPTIONS

From security point of view, public-key cryptography with deterministic encryptions has the following serious drawback:

A cryptanalyst who knows the public encryption function e_k and a cryptotext c can try to guess a plaintext w, compute $e_k(w)$ and compare it with c.

The purpose of randomized encryptions is to encrypt messages, using randomized algorithms, in such a way that one can prove that no feasible computation on the cryptotext can provide any information whatsoever about the corresponding plaintext (except with a negligible probability).

Formal setting: Given:

	plaintext-space	P
	cryptotext	C
	key-space	K
	random-space	R

encryption: $e_k : P \times R \to C$
decryption: $d_k : C \to P$ or $C \to 2^P$ such that for any p, r:

$$p = d_k(e_k(p,r)) \quad \text{or} \quad p \in d_k(e_k(p,r))$$

- d_k and e_k should be easy to compute.
- Given e_k, it should be unfeasible to determine d_k.

WHEN is a CRYPTOSYSTEM (perfectly) SECURE?

First question: Is it enough for perfect security of a cryptosystem that one cannot get a plaintext from a cryptotext?

NO, NO, NO
WHY

For many applications it is crucial that no information about the plaintext could be obtained.

- Intuitively, a cryptosystem is (perfectly) secure if one cannot get any (new) information about the corresponding plaintext from any cryptotext.
- It is very nontrivial to define fully precisely when a cryptosystem is (computationally) perfectly secure.
- It has been shown that perfectly secure cryptosystems have to use randomized encryptions.

SECURE ENCRYPTIONS – BASIC CONCEPTS I

We now start to discuss a very nontrivial question: **when is an encryption scheme computationally perfectly SECURE?**

At first, we introduce two very basic technical concepts:

Definition A function f:$N \to R$ is **a negligible function** if for any polynomial $p(n)$ and for almost all n:

$$f(n) \leq \frac{1}{p(n)}$$

Definition – computational distinguishability Let $X = \{X_n\}_{n \in N}$ and $Y = \{Y_n\}_{n \in N}$ be **probability ensembles** such that each X_n and Y_n ranges over strings of length n. We say that X and Y are computationally indistinguishable if for every feasible algorithm A the difference

$$d_A(n) = \mid Pr[A(X_n) = 1] - Pr[A(Y_n) = 1] \mid$$

is a negligible function in n.

SECURE ENCRYPTION – FIRST DEFINITION

Definition – semantic security of encryption A cryptographic system with an encryption function e is **semantically secure** if for every feasible algorithm A, there exists a feasible algorithm B so that for every two functions

$$f, h : \{0,1\}^* \to \{0,1\}^n$$

and all probability ensembles $\{X_n\}_{n \in N}$, where X_n ranges over $\{0,1\}^n$

$$Pr[A(e(X_n), h(X_n)) = f(X_n)] < Pr[B(h(X_n)) = f(X_n)] + \mu(n),$$

where μ is a negligible function.

In other words, a cryptographic system is semantically secure if whatever we can do with the knowledge of cryptotext we can do also without that knowledge.

It can be shown that any semantically secure public-key cryptosystem must use a randomized encryption algorithm.

RSA cryptosystem is not secure in the above sense. However, randomized versions of RSA are semantically secure.

PSEUDORANDOM GENERATORS - PRG

PSEUDORANDOM GENERATORS - PRG

PSEUDORANDOM GENERATORS STORY

Pseudorandom generators are algorithms that generate pseudorandom (almost random) strings or integers.

Pseudorandom generators is an additional key concept of cryptography and of the design of efficient algorithms.

There is a variety of classical algorithms capable to generate pseudorandomness of different quality concerning randomness.

Quantum processes can generate perfect randomness and on this basis quantum (almost perfect) generators of randomness are already commercially available.

STORY of RANDOMNESS

STORY of RANDOMNESS

DOES RANDOMNESS EXIST? - I

One of the fundamental questions (of science) has been, and actually still is, whether randomness really exists or whether term **randomness** is used only to deal with events the laws of which we do not fully understand. Two early views are:

The ***randomness is the unknown*** *and Nature is determined in its fundamentals.*

Democritos (470-404 BC)

By Democritos, the order conquered the world and this order is governed by unambiguous laws. By Leucippus, the teacher of Democritos.

Nothing occurs at random, but everything for a reason and necessity.

By Democritus and Leucippus, the word **random** is used when we have an incomplete knowledge of some phenomena. On the other side:

The randomness is objective, it is the proper nature of some events.

Epikurus (341-270 BC)

By Epikurus, there exists a true randomness that is independent of our knowledge.

Einstein also accepted the notion of randomness only in the relation to incomplete knowledge.

VIEWS on RANDOMNESS in 19th CENTURY

Main arguments, before 20th century, why randomness does not exist:

God-argument: There is no place for randomness in a world created by God.

Science-argument: Success of natural sciences and mechanical engineering in 19th century led to a belief that everything could be discovered and explained by deterministic causalities of a cause and the resulting effect.

Emotional-argument: Randomness used to be identified with uncertainty or unpredictability or even chaos.

There are only two possibilities, either a big chaos conquers the world, or order and law.

Marcus Aurelius

EINSTEIN versus BOHR

God does not roll dice.

Albert Einstein, 1935, a strong opponent of randomness.

The true God does not allow anybody to prescribe what he has to do.

Famous reply by Niels Bohr - one of the fathers of quantum mechanics.

RANDOMNESS in NATURE

Two big scientific discoveries of 20th century changed the view on usefulness of randomness.

1. It has turned out that random mutations of DNA have to be considered as a crucial instrument of evolution.
2. Quantum measurement yields, in principle, random outcomes.

RANDOMNESS

- Randomness as a mathematical topic has been studied since 17th century.
- Attempts to formalize chance by mathematical laws is somehow paradoxical because, a priory, chance (randomness) is the subject of no law.
- There is no proof that perfect randomness exists in the real world.
- More exactly, there is no proof that quantum mechanical phenomena of the microworld can be exploited to provide a perfect source of randomness for the macroworld.

CRYPTOGRAPHICALLY PERFECT PSEUDORANDOM GENERATORS

One of the most basic questions of perfect security of encryptions is whether there are **cryptographically perfect pseudorandom generators** and what such a concept really means.

The concept of pseudorandom generators is quite old. An interesting example is due to John von Neumann:

Take an arbitrary integer x as the "seed"
and repeat the following process:
compute x^2 and take a sequence of the middle digits of x^2 as a new "seed" x.

SIMPLE PSEUDORANDOM GENERATORS

Informally, a **pseudorandom generator** is a deterministic polynomial time algorithm which expands short random sequences (called **seeds**) into longer bit sequences such that the resulting probability distribution is in polynomial time indistinguishable from the uniform probability distribution.

Example. **Linear congruential generator**

One chooses n-bit numbers m, a, b, X_0 and generates an n^2 element sequence

$$X_1 X_2 \dots X_{n^2}$$

of n-bit numbers by the iterative process

$$X_{i+1} = (aX_i + b) \bmod m.$$

CRYPTOGRAPHY and RANDOMNESS

Randomness and cryptography are deeply related.

1. Prime goal of any good encryption method is to transform even a highly nonrandom plaintext into a highly random cryptotext. (Avalanche effect.)

   Example Let e_k be an encryption algorithm, x_0 be a plaintext. And
   $$x_i = e_k(x_{i-1}), i \geq 1.$$
   It is intuitively clear that if encryption e_k is "cryptographically secure", then it is very, very likely that the sequence $x_0\ x_1\ x_2\ x_3$ is (quite) random.

   Perfect encryption should therefore produce (quite) perfect (pseudo)randomness.

2. The other side of the relation is more complex. It is clear that perfect randomness together with ONE-TIME PAD cryptosystem produces perfect secrecy. The price to pay: a key as long as plaintext is needed.

   The way out seems to be to use an encryption algorithm with a pseudo-random generator to generate a long pseudo-random sequence from a short seed and to use the resulting sequence with ONE-TIME PAD.

   Basic question: When is a pseudo-random generator good enough for cryptographical purposes?

CRYPTOGRAPHICALY STRONG PSEUDORANDOM GENERATORS

In cryptography **random sequences** can usually be replaced by **pseudorandom sequences** generated by **(cryptographically perfect/strong) pseudorandom generators**.

Definition. Let $l(n) : N \to N$ be such that $l(n) > n$ for all n. A **(cryptographically strong) pseudorandom generator with a stretch function l**, is an efficient deterministic algorithm which on the input of a random n-bit seed outputs a $l(n)$-bit sequence which is computationally indistinguishable from any random $l(n)$-bit sequence.

Candidate for a cryptographically strong pseudorandom generator:

A very fundamental concept: A predicate b is a hard core predicate of the function f if b is easy to evaluate, but $b(x)$ is hard to predict from f(x). (That is, it is unfeasible, given f(x) where x is uniformly chosen, to predict $b(x)$ substantially better than with the probability 1/2.)

Conjecture: The least significant bit of x^2 **mod** n is a hard-core predicate.

Theorem Let f be a one-way function which is length preserving and efficiently computable, and b be a hard core predicate of f, then

$$G(s) = b(s) \cdot b(f(s)) \cdots b\left(f^{l(|s|)-1}(s)\right)$$

is a (cryptographically strong) pseudorandom generator with stretch function $l(n)$.

THEOREM

Theorem A cryptographically strong (perfect) pseudorandom generator exists if one-way functions exist.

PSEUDORANDOM GENERATORS and ENCRYPTIONS

If two parties share a pseudorandom generator g, and exchange (secretly) a short random string - (**seed**) - s

then they can generate and use long pseudorandom string $g(s)$ as a key k

for one-time pad for encoding and decoding.

CANDIDATES for CRYPTOGRAPHICALLY STRONG PSEUDO-RANDOM GENERATORS

So far there are only candidates for cryptographically strong pseudo-random generators.

For example, **cryptographically strong are all pseudo-random generators that are unpredictable to the left** in the sense that a cryptanalyst that knows the generator and sees the whole generated sequence except its first bit has no better way to find out this first bit than to toss the coin.

It has been shown that if integer factoring is intractable, then the so-called *BBS* pseudo-random generator, discussed below, is unpredictable to the left.

(We make use of the fact that if factoring is unfeasible, then for almost all quadratic residues x mod n, coin-tossing is the best possible way to estimate the least significant bit of x after seeing x^2 mod n.)

Let n be a Blum integer. Choose a random quadratic residue x_0 (modulo n).

For $i \geq 0$ let

$$x_{i+1} = {x_i}^2 \bmod n, \quad b_i = \text{the least significant bit of } x_i$$

For each integer i, let

$$BBS_{\ n,i}(x_0) = b_0 \dots b_{i-1}$$

be the first i bits of the pseudo-random sequence generated from the seed x_0 by the *BBS* pseudo-random generator.

PERFECTLY SECURE CIPHERS - EXAMPLES

PERFECTLY SECURE CIPHERS - EXAMPLES

RANDOMIZED VERSION of RSA-LIKE CRYPTOSYSTEM

The scheme works for **any trapdoor function** (as in case of RSA),

$$f : D \to D, D \subset \{0,1\}^n,$$

for any **pseudorandom generator**

$$G : \{0,1\}^k \to \{0,1\}^l,\ k << l$$

and any **hash function**

$$h : \{0,1\}^l \to \{0,1\}^k,$$

where **n = l + k**. Given a random seed $s \in \{0,1\}^k$ as input, G generates a pseudorandom bit-sequence of length l.

Encryption of a message $m \in \{0,1\}^l$ is done as follows:

1. A random string $r \in \{0,1\}^k$ is chosen.
2. Set $x = (m \oplus G(r)) \| (r \oplus h(m \oplus G(r)))$. (If $x \notin D$ go to step 1.)
3. Compute encryption c = f(x) – length of x and of c is n.

Decryption of a cryptotext c.

- Compute $f^{-1}(c) = a \| b$, $|a| = l$ and $|b| = k$.
- Set $r = h(a) \oplus b$ and get $m = a \oplus G(r)$.

Comment: Operation "$\|$" stands for a concatenation of strings.

BLOOM-GOLDWASSER CRYPTOSYSTEM

Private key: Blum primes p and q.
Public key: n = pq.
Encryption of $x \in \{0,1\}^m$.

1. Randomly choose $s_0 \in \{0,1,\dots,n\}$.
2. For i = 1, 2, . . . , m + 1 compute

$$s_i \leftarrow s_{i-1}^2 \mod n$$

and $\sigma_i = lsb(s_i)$. —{lsb – least significant bit}

The cryptotext is then (s_{m+1}, y), where $y = x \oplus \sigma_1\sigma_2 \dots \sigma_m$.

Decryption: of the cryptotext (r, y):
Let $d = 2^{-m} \mod \phi(n))$.

- Let $s_1 = r^d \mod n$.
- For i = 1, . . . , m, compute $\sigma_i = lsb(s_i)$ and $s_{i+1} \leftarrow s_i^2 \mod n$

The plaintext x can then be computed as $y \oplus \sigma_1\sigma_2 \dots \sigma_m$.

CRITERIA for a CRYPTOSYSTEM to be PRACTICAL

One can neve be sure - in the sence of a rigorous proof - that a public-key cryptosystem cannot feasibly be broken.

The best one can hope for is to havve a large number of empirical evidence that

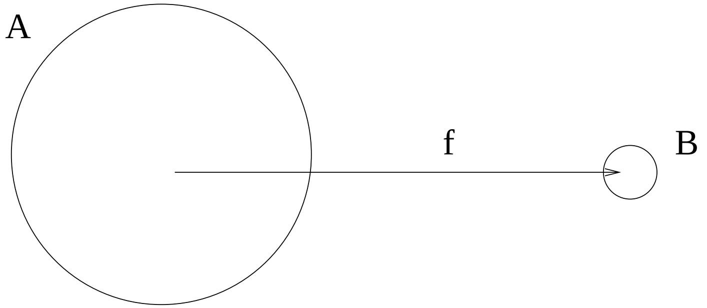
- the cryptosystem cannot be cracked without solving a ceratin mathemarical problem, and
- there is no method known, in spite of many years of many attempts, to show that that problem can be solved in a reasonable length of time.

HASH FUNCTIONS

HASH FUNCTIONS

HASH FUNCTIONS - PICTURE

Hash functions f map huge sets A (randomly and uniformly) into very small sets B in such a way that for many important information processing tasks one can, well enough, replace working with (huge) elements x from A by working with (small) elements $f(x)$ from B.



Cryptographic hash functions are hash functions that satisfy well enough basic cryptographic properties.

HASH FUNCTIONS - BASICS

A **hash function** is any function that maps (uniformly and randomly) digital data of huge (arbitrary) size to digital data of small fixed size, in such a way that slight differences in input data produce big differences in output data.

The values returned by a hash function are called **hash values, hash codes, fingerprints, message digests, digests** or simply **hashes**.

A good hash function should map possible inputs as evenly as possible over its output range.

In other words, if a hash function maps a set A of n elements into a set B of $m << n$ elements, then the probability that an element of B is the value of much more than $\frac{n}{m}$ elements of A should be very small.

Hash function have a variety applications, especially in the design of efficient algorithms and in cryptography.

CRYPTOGRAPHIC HASH FUNCTIONS

A good cryptographic hash function f is such a hash function that withstands all known cryptographic attacks. As a minimum, it must have the following properties:

Pre-image resistance: Given a hash h it should be unfeasible (difficult) to find a (message) m such that $h = f(m)$. In such a case it is also said that f should have **one-wayness property**.

Second pre-image resistance: Given a message m_1 it should be unfeasible (difficult) to find another message m_2 such that $f(m_1) = f(m_2)$. In such a case it is also said that f should be **weakly collision resistant**.

Collision resistance: It should be unfeasible (difficult) to find two messages m_1 and m_2 such that $f(m_1) = f(m_2)$. In such a case it is also said that f should be **strongly collision resistant**.

In cryptographic practice "**difficult**" generally means "**almost certainly beyond the reach of any adversary who must be prevented from breaking the system for as long as the security of the system is considered to be very important**".

SOME APPLICATIONS

- **To verify integrity of messages**: To determine whether a change was made to a message during a transmission, can be done by comparing message digests calculating before, and after, the transmission.
- **Passport verification** The idea is to store only hashes of each password. To authenticate a user, the password presented by the user is hashed and resulting hash is compared with the stored hash.

In 2013 a long-term **Password Hashing Competition** was announced to choose a new, standard algorithm for password hashing.

HASH FUNCTIONS and INTEGRITY of DATA

An important use of hash functions is to protect integrity of data:

The problem of protecting **integrity of data of arbitrary length** is reduced, using hash functions, to the problem to protect **integrity of data of fixed (and small) length hashes** – of the data fingerprints.

In addition, to send reliably a message w through an unreliable (and cheap) channel, one sends also its (small) hash h(w) through a very secure (and therefore expensive) channel.

The receiver, familiar also with the hash function h that is being used, can then verify the integrity of the message w' he receives by computing h(w') and comparing

h(w) and h(w') .

EXAMPLES

Example 1 For a vector $a = (a_1, \ldots, a_k)$ of integers let

$$H(a) = \sum_{i=0}^{k} a_i \mod n$$

where n is a product of two large primes.
This hash functions does not meet any of the three properties mentioned above.

Example 2 For a vector $a = (a_1, \ldots, a_k)$ of integers let

$$H(a) = \sum_{i=0}^{k} a_i^2 \mod n$$

where n is product of two large primes.

This function is one-way, but it is not weakly collision resistant.

HASH FUNCTIONS h from CRYPTOSYSTEMS

Let us have computationally secure cryptosystem with plaintexts, keys and cryptotexts being binary strings of a fixed length n and with encryption functions e_j.

If

$$x = x_1 \| x_2 \| \ldots \| x_m$$

is the decomposition of x into substrings of length n, g_0 is a random string, and

$$g_i = f(x_i, g_{i-1})$$

for $i = 1, \ldots, m$, where f is a function that "incorporates" encryption functions e_j of the cryptosystem, for suitable keys k_j, then

$$h(x) = g_m \ .$$

For example such good properties have these two functions:

$$f(x_i, g_{i-1}) = e_{g_{i-1}}(x_i) \oplus x_i$$
$$f(x_i, g_{i-1}) = e_{g_{i-1}}(x_i) \oplus x_i \oplus g_{i-1}$$

PRACTICALLY USED HASH FUNCTIONS

A variety of hash functions has been constructed. Very often used hash functions were MD4, MD5 (created by Rivest in 1990 and 1991 and producing 128 bit message digest).

NSA published, as standards, starting in 1993, SHA-0, SHA-1 (Secure Hash Algorithm) – producing 160 bit message digest – based on similar ideas as MD4 and MD5.

Some of the most important cryptographic results of the last years were due to the Chinese Wang who has shown that MD4 is not cryptographically perfectly secure and Dr. Kimy who has done that also for MD5.

Observe that every cryptographic hash function is vulnerable to a collision attack using so called birthday attack. Due to the **birthday problem** a hash of n bits can be broken in $\sqrt{2^n}$ evaluations of the hash function - much faster than the brute force attack.

RECENT DEVELOPMENTS CONCERNING HASH FUNCTIONS

- In February 2005, an attack on SHA-1 was reported that would find collision in about 2^{69} hashing operations - rather than the 2^{80} as expected by dictionary attack for a 160-bit hash function.
- In August 2005 another attack on SHA-1 was reported that would find collisions in 2^{63} operations.
- Though no collision for SHA-1 was found, it started to be expected that this will soon happen and so SHA2 was developed.
- Very recently a successful attack on SH1 has been reported.
- In order to ensure long-term robustness of applications that use hash functions a public competition was announced by NIST to replace SHA-2.
- On October 2012 Keccak was selected as the winner and a version of this algorithm is expected to be a new standard (since 2014) under the name SHA-3.

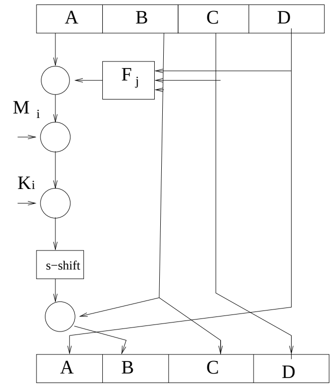
MD5

Often used in practise has been hash function MD5 designed in 1991 by Rivest. It maps any binary message into 128-bit hash.

The input message is broken into 512-bit blocks, divided into 16 words-states (of 32 bits) and padded if needed to have final length divisible by 512. Padding consists of a bit 1 followed by so many 0's as required to have the length up to 64 bits fewer than a multiple of 512. Final 64 bits represent the length of the original message modulo 2^{64}.

The main MD5 algorithm operates on 128-bits words that are divided into four 32-bits words A, B, C, D initialized to some fixed constants. The main algorithm then operates on 512 bit message blocks in turn - each block modifying the state.

The processing of a message consists of four rounds. j-th round is composed of 16 similar operations using non-linear functions F_j and left rotations by s_j places where s_j varies for each round - see next figure. K_i and M_i are 32-bits keys and messages.



BREAKING MD5

- In 2006 Vladimír Klima published an algorithm to find a collision for MD5 within one minute on a notebook.
- In 2010 T. Xie, O. Feng published single-block MD5 collision.

HOW to FIND COLLISIONS of HASH FUNCTIONS

HOW to FIND COLLISIONS of HASH FUNCTIONS

The most basic method is based on so-called birthday paradox related to so-called the birthday problem.

BIRTHDAY PROBLEM and its VARIATIONS

It is well known that if there are 23 (29) [40] {57} $< 100 >$ people in one room, then the probability that two of them have the same birthday is more than 50% (70%)[89%] {99%} $< 99.99997\% >$ — this is called a Birthday paradox.

More generally, if we have n objects and r people, each choosing one object (so that several people can choose the same object), then if $r \approx 1.177\sqrt{n}(r \approx \sqrt{2n\lambda})$, then probability that two people choose the same object is 50% $((1 - e^{-\lambda})\%)$.

Another version of the birthday paradox: Let us have n objects and two groups of r people. If $r \approx \sqrt{\lambda n}$, then probability that someone from one group chooses the same object as someone from the other group is $(1 - e^{-\lambda})$.

BASIC DERIVATIONS related to BIRTHDAY PARADOX

For the probability $\bar{p}(n)$ that all $n < 366$ people in a room have birthday in different days, it holds

$$\bar{p}(n) = \prod_{i=1}^{n-1}\left(\frac{365-i}{365}\right) = \frac{\prod_{i=1}^{n-1}(365-i)}{365^{n-1}} = \frac{365!}{365^n(365-n)!}$$

This equation expresses the following fact: First chosen person has for sure birthday different from any person chosen before. the second person cannot have the same birthday as the first one with probability $\frac{365-1}{365}$, the third person cannot have the same birthday as first two with probbility $\frac{365-2}{365}$,,.....

Probability $p(n)$ that at least two person have the same birthday is therefore

$$p(n) = 1 - \bar{p}(n)$$

This probability is larger than 0.5 first time for $n = 23$.

FINDING COLLISIONS USING BIRTHDAY PARADOX

If the hash of a hash function h has the size n, then to a given x to find x' such that $h(x) = h(x')$ by brute force requires 2^n hash computations in average.

The idea, based on the birthday paradox, is simple. Given x we iteratively pick a random x' until $h(x) = h(x')$. The probability that i-th trial is the first one to succeed is $(1 - 2^{-n})^{i-1}2^{-n}$;

The average complexity, in terms of hash function computations is therefore

$$\sum_{i=1}^{\infty} i(1 - 2^{-n})^{i-1}2^{-n} = 2^n.$$

To find collisions, that is two x_1 and x_2 such that $h(x_1) = h(x_2)$ is easier, thanks to the birthday paradox and can be done by the following algorithm:

ALGORITHM

Input: A hash function h onto a domain of size n, a real θ and an empty hash table.
Output: A pair (x_1, x_2) such that $x_1 \neq x_2$ and $h(x_1) = h(x_2)$

1. for $\theta\sqrt{(n)}$ different x do
2. compute $y = h(x)$
3. if there is a (y, x') pair in the hash table then
4. yield (x, x') and stop
5. add (y, x) to the hash table
6. Otherwise search failed

Theorem If we pick the numbers x with uniform distribution in $\{1, 2, \ldots, n\}$ $\theta\sqrt{n}$ times, then we get at least one number twice with probability converging (for $n \to \infty$) to

$$1 - e^{-\frac{\theta^2}{2}}$$

For $n = 365$ we get triples: $(\theta, \theta\sqrt{n}, \text{probability})$ as follows: (0.79, 15, 25%); (1.31, 25, 57%); (2.09, 40, 89%)

WHY CURRENTLY BROADLY USED HASHES HAVE 160 BITS?

The birthday paradox imposes also a lower bound on the sizes of hashes of the cryptographically good hash functions.

For example, a 40-bit hashes would be insecure because a collision could be found with probability 0.5 with just over 40^{20} random guesses.

Minimum acceptable size of hashes seems to be 128 and therefore 160 are used in such important systems as DSS – Digital Signature Schemes (a standard).