Part VI

Public-key cryptosystems, II. Other cryptosystems, security, PRG, hash functions

CHAPTER 6: OTHER CRYPTOSYSTEMS, PSEUDO-RANDOM NUMBER GENERATORS and HASH FUNCTIONS

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 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 genral infeasible in some groups. Secondly, we discuss pseudo-random number generators and hash functions – other very important concepts of modern cryptography

Finally, 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.

RABIN CRYPTOSYSTEM

Primes p, q of the form $4k+3$, so called Blum primes, are kept secret, $n = pq$ is the public key.
Encryption: of a plaintext $w < n$

$$c = w^2 \bmod n$$

Decryption: 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

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

are two square roots of c modulo p and q. One can now obtain four square roots of c modulo n using the method shown in Appendix.
In case the plaintext w is a meaningful English text, it should be easy to determine w from w_1, w_2, w_3, w_4.

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.
Rabin did not propose this system as a practical cryptosystem.

GENERALIZED RABIN CRYPTOSYSTEM

Public key: n, B $(0 \leq B \leq n-1)$

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* ${x_1}^2 \equiv c \pmod n$
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 Las Vegas algorithm, to factor an integer n.

Algorithm:

1. Choose a random $r, 1 \leq r \leq n-1$;
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

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

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

ElGamal CRYPTOSYSTEM

Design: choose a large prime p – (with at least 150 digits).
choose two random integers $1 \leq q, x < p$ – 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: $a^x \equiv q^{rx} \bmod p$

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

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

SHANKS' ALGORITHM for DISCRETE ALGORITHM

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 second $(i, z) \in L_2$

If such a search is successful, then

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

and as the result

$$\lg_q y \equiv (mj + i) \bmod (p-1).$$

Therefore

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

On the other hand, for any y we can write

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

For some $0 \leq i, j \leq m-1$. Hence the search in the Step 5 of the algorithm has to be successful.

BIT SECURITY of DISCRETE LOGARITHM

Let us consider problem to compute $L_i(y) =$ i-th least significant bit of $\lg_q y$ in Z^*_p.

Result 1 $L_1(y)$ can be computed efficiently.
To show that we use the fact that the set $QR(p)$ has $(p-1)/2$ elements.
Let q be a primitive element of Z^*_p. Clearly, $q^a \in QR(p)$ if a is even. Since the elements

$$q^0 \bmod p, q^2 \bmod p, \ldots, q^{p-3} \bmod p$$

are all distinct, we have that

$$QR(p) = \{q^{2i} \bmod p \,|\, 0 \leq i \leq (p-3)/2\}$$

Consequence: y is a quadratic residue iff $\lg_q y$ is even, that is iff $L_1(y) = 0$.

By Euler's criterion y is a quadratic residue if $y^{(p-1)/2} \equiv 1 \bmod p$
$L_1(y)$ can therefore be computed as follows:

$$L_1(y) = 0 \qquad \text{if } y^{(p-1)/2} \equiv 1 \bmod p;$$
$$L_1(y) = 1 \qquad \text{otherwise}$$

Result 2 Efficient computability of $L_i(y), i > 1$ in Z^*_p would imply efficient computability of the discrete logarithm in Z^*_p.

GROUP VERSION of ElGamal CRYPTOSYSTEM

A group version of discrete logarithm probem

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

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

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 k

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

Decryption: of cryptotext (y_1, y_2):

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

An important special case is that of computation of discrete logarithm in a group of points of an eliptic curve defined over a finite field.

WILLIAMS CRYPTOSYSTEM – BASICS

This cryptosystem is similar to RSA, but with number operations performed in a quadratic field. Complexity of the cryptanalysis of the Williams cryptosystem is equivalent to factoring.

Consider numbers of the form

$$\alpha = a + b\sqrt{c}$$

where a, b, c are integers.

If c is fixed, α can be viewed as a pair (a, b).

$$\alpha_1 + \alpha_2 = (a_1, b_1) + (a_2, b_2) = (a_1 + a_2, b_1 + b_2)$$
$$\alpha_1\alpha_2 = (a_1, b_1) \cdot (a_2, b_2) = (a_1a_2 + c\,b_1b_2, a_1b_2 + b_1a_2)$$

The conjugate $\overline{\alpha}$ of α of a is defined by

$$\overline{\alpha} = a - b\sqrt{c}$$

Auxiliary functions:

$$X_i(\alpha) = \frac{\alpha^i + \alpha^{-i}}{2}$$
$$Y_i(\alpha) = \frac{b(\alpha^i - \alpha^{-i})}{(\alpha - \overline{\alpha})} \left(= \frac{\alpha - \overline{\alpha}^i}{2\sqrt{c}} \right)$$

Hence

$$\alpha^i = X_i(\alpha) + Y_i(\alpha)\sqrt{c}$$
$$\overline{\alpha}^i = X_i(\alpha) - Y_i(\alpha)\sqrt{c}$$

WILLIAMS CRYPTOSYSTEM – EFFICIENT EXPONENTIATION

Assume now

$$a^2 - cb^2 = 1$$

Then $\alpha\overline{\alpha} = 1$ and consequently

$$X_I{}^2 - cY_I{}^2 = 1$$

Moreover, for $j \geq i$

$$X_{I+J} = 2X_IX_J + X_{J-1}$$
$$Y_{I+J} = 2Y_IX_J + Y_{J-1}$$

From these and following equations:

$$X_{I+J} = 2X_IX_J + cY_IY_J$$
$$Y_{I+J} = 2Y_IX_J + X_IY_J$$

we get the recursive formulas:

$$X_{2i} = X_i{}^2 + cY_i{}^2 = 2X_i{}^2 - 1$$
$$Y_{2i} = 2X_iY_i$$
$$X_{2i+1} = 2X_iY_{i+1} - X_1$$
$$Y_{2i+1} = 2X_iY_{i+1} - Y_1$$

Consequences: 1. X_i and Y_i can be, given i, computed fast.
Remark Since $X_0 = 1, X_1 = a, X_i$ does not depend on b.

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.

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?

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:$N \rightarrow R$ 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 ENCRYPTIONS – PSEUDORANDOM GENERATORS

In cryptography random sequences can be usually be well enough replaced by pseudorandom sequences generated by (cryptographically perfect) pseudorandom generators.

Definition - pseudorandom generator. Let $l(n) : N \rightarrow N$ be such that $l(n) > n$ for all n. A (computationally indistinguishable) 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.

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 (computationally indistinguishable) pseudorandom generator with stretch function $l(n)$.

Definition 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.)

It is conjectured that the least significant bit of the modular squaring function x^2 mod n is a hard-core predicate.

CRYPTOGRAPHICALLY STRONG PSEUDO-RANDOM GENERATORS

Fundamental question: when is a pseudo-random generator good enough for cryptographical purposes?

Basic concept: A pseudo-random generator is called cryptographically strong if the sequence of bits it produces, from a short random seed, is so good for using with ONE-TIME PAD cryptosystem, that no polynomial time algorithm allows a cryptanalyst to learn any information about the plaintext from the cryptotext.

A cryptographically strong pseudo-random generator would therefore provide sufficient security in a secret-key cryptosystem if both parties agree on some short seed and never use it twice.
As discussed later: Cryptographically strong pseudo-random generators could provide perfect secrecy also for public-key cryptography.

Problem: Do cryptographically strong pseudo-random generators exist?

Remark: The concept of a cryptographically strong pseudo-random generator is one of the key concepts of the foundations of computing.

Indeed, a cryptographically strong pseudo-random generator exists if and only if a one-way function exists what is equivalent with $P \neq UP$ and what implies $P \neq NP$.

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

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

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.

BBS PSEUDO-RANDOM GENERATOR – ANALYSIS

Choose random x, relatively prime to n, compute $x_0 = x^2$ mod n
Let $x_{i+1} = x_i{}^2$ mod n, and b_i be the least significant bit of x_i
$BBS_{n,i}(x_0) = b_0 \dots b_{i-1}$

Assume that the pseudo-random generator BBS with a Blum integer is not unpredictable to the left.

Let y be a quadratic residue from $Z_n{}^*$.

Compute $BBS_{n,i-1}(y)$ for some $i > 1$.

Let us pretend that last $(i-1)$ bits of $BBS_{n,i}(x)$ are actually the first $(i-1)$ bits of $BBS_{n,i-1}(y)$, where x is the principal square root of y.

Hence, if the *BBS* pseudo-random generator is not unpredictable to the left, then there exists a better method than coin-tossing to determine the least significant bit of x, what is, as mentioned above, impossible.

RANDOMIZED ENCRYPTIONS

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

$$d_k(e_k(p, r)) = p.$$

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

SECURE ENCRYPTION – FIRST DEFINITION

Definition – semantic security of encryption A cryptographic system 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.

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.

SECURE ENCRYPTIONS – SECOND DEFINITION

Definition A randomized-encryption cryptosystem is **polynomial time** secure if, for any c $\in N$ and sufficiently large s $\in N$ (security parameter), any randomized polynomial time algorithms that takes as input s (in unary) and the public key, cannot distinguish between randomized encryptions, by that key, of two given messages of length c, with the probability larger than $\frac{1}{2}+\frac{1}{s^c}$.

Both definitions are equivalent.

Example of a polynomial-time secure randomized (Bloom-Goldwasser) encryption:

p, q - large Blum primes n = p × q - key
Plaintext-space - all binary strings
Random-space – QR_n
Crypto-space - $QR_n \times \{0,1\}^*$

Encryption: Let w be a t-bit plaintext and x_0 a random quadratic residue modulo n. Compute x_t and $BBS_{n,t}(x_0)$ using the recurrence

$$x_{i+1} = x_i^2 \mod n$$

Cryptotext: $(x_t, w \oplus BBS_{n,t}(x_0))$

Decryption: Legal user, knowing p, q, can compute x_0 from x_t, then $BBS_{n,t}(x_0)$, and finally w.

HASH FUNCTIONS

Another very simple, fundamental and important cryptographic concept is that of hash functions.

Hash functions

$$h : \{0,1\}^* \to \{0,1\}^m; \qquad h : \{0,1\}^n \to \{0,1\}^m, \ n >> m$$

map (very) long messages w into short ones, called usually messages digests or hashes or fingerprints of w, in a way that has important cryptographic properties.

Digital signatures are one of important applications of hash functions.

In most of the digital signature schemes, to be discussed in the next chapter, the length of a signature is at least as long as of the message being signed. This is clearly a big disadvantage.

To remedy this situation, signing procedure is applied to a hash of the message, rather than to the message itself. This is OK provided the hash function has good cryptographic properties, discussed next.

PROPERTIES GOOD HASH FUNCTIONS SHOULD HAVE I.

We now derive basic properties cryptographically good hash functions should have – by analysing several possible attacks on their use.

Attack 1 If Eve gets a valid signature (w,y), where $y = sig_k(h(w))$ and she would be able to find w' such that h(w')=h(w), then also (w',y), a forgery, would be a valid signature.

Cryptographically good hash function should therefore have the following weak collision-free property

Definition 1. Let w be a message. A hash function h is weakly collision-free for w, if it is computationally infeasible to find a w' such that h(w)=h(w').

PROPERTIES GOOD HASH FUNCTIONS SHOULD HAVE II.

Attack 2 If Eve finds two w and w' such that h(w')=h(w), she can ask Alice to sign h(w) to get signature s and then Eve can create a forgery (w',s).

Cryptographically good hash function should therefore have the following strong collision-free property

Definition 2. A hash function h is strongly collision-free if it is computationally infeasible to find two elements $w \neq w'$ such that h(w)=h(w').

PROPERTIES HASH FUNCTIONS SHOULD HAVE III.

Attack 3 If Eve can compute signature s of a random z, and then she can find w such that z=h(w), then Eve can create forgery (w,s).

To exclude such an attack, hash functions should have the following one-wayness property.

Definition 3. A hash function h is one-way if it is computationally infeasible to find, given z, an w such that h(w)=z.

One can show that if a hash function has strongly collision-free property, then it has one-wayness property.

HASH FUNCTIONS and INTEGRITY of DATA

An important use of hash functions is to protect integrity of data in the following way:

The problem of protecting data of arbitrary length is reduced, using hash functions, to the problem to protect integrity of the data of fixed (and small) length – of their 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 integers.
This hash functions does not meet any of the three properties mentioned on the last slide.
Example 2 For a vector $a = (a_1, \ldots, a_k)$ of integers let

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

This fuction is one-way, but it is not weakly collision-free.

FINDING COLLISIONS with INVERSION ALGORITHM

Theorem Let $h : X \rightarrow Z$ be a hash function where X and Z are finite and $|X| \geq 2|Z|$. If there is an inversion algorithm **A** for h, then there exists randomized algorithm to find collisions.

Sketch of the proof. One can easily show that the following algorithm

1. Choose a random $x \in X$ and compute z=h(x); Compute $x_1 = \mathbf{A}(z)$;
2. **if** $x_1 \neq x$, then x_1 and x collide (under h – success) **else** failure

has probability of success

$$p(success) = \frac{1}{|X|} \sum_{x \in X} \frac{|[x]| - 1}{|[x]|} \geq \frac{1}{2}$$

where, for $x \in X$, [x] is the set of elements having the same hash as x.

VARIATIONS on BIRTHDAY PARADOX

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

$$\bar{p}(n) = \prod_{i=1}^{n-1}\left(1-\frac{i}{365}\right) = \frac{\prod_{i=0}^{n-1}(365-i)}{365^n} = \frac{365!}{365^n(365-n)!}$$

This equation expresses the fact for no person to share a birthday, the second person cannot have the same birthday as the first one, third person cannot have the same birthday as first two,.....

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 $n = 23$.

HASH FUNCTION DOMAIN LOWER BOUND

Birthday paradox imposes a lower bound on the sizes of message digests (fingerprints)

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

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

AN ALMOST GOOD HASH FUNCTION

We show an example of the hash function (so called Discrete Log Hash Function) that seems to have as the only drawback that it is too slow to be used in practice:

Let p be a large prime such that $q = \frac{(p-1)}{2}$ is also prime and let α, β be two primitive roots modulo p. Denote $a = \log_\alpha \beta$ (that is $\beta = \alpha^a$).

h will map two integers smaller than q to an integer smaller than p, for $m = x_0 + x_1 q, 0 \le x_0, x_1 \le q-1$ as follows,

$$h(x_0, x_1) = h(m) = \alpha^{x_0}\beta^{x_1} \pmod{p}.$$

To show that h is one-way and collision-free the following fact can be used:

FACT: If we know different messages m_1 and m_2 such that $h(m_1) = h(m_2)$, then we can compute $\log_\alpha \beta$.

EXTENDING HASH FUNCTIONS

Let $h : \{0,1\}^m \rightarrow \{0,1\}^t$ be a strongly collision-free hash function, where $m > t+1$.

We design now a strongly collision-free hash function

$$h^* : \sum_{i=m}^{\infty} \{0,1\}^i \rightarrow \{0,1\}^t.$$

Let a bit string x, $|x| = n > m$, have decomposition

$$x = x_1 \| x_2 \dots \| x_k \ ,$$

where $|x_i| = m - t - 1$ if $i < k$ and $|x_k| = m - t - 1 - d$ for some d. (Hence $k = \left\lceil \frac{n}{(m-t-1)} \right\rceil$.)

h^* will be computed as follows:

1. **for** i=1 **to** k-1 **do** $y_i := x_i$;
2. $y_k := x_k \| 0^d$; $y_{k+1} :=$ binary representation of d ;
3. $g_1 := h(0^{t+1} \| y_1)$;
4. **for** i=1 **to** k **do** $g_{i+1} := h(g_i \| 1 \| y_{i+1})$;
5. $h^*(x) := g_{k+1}$.

HASH FUNCTIONS from CRYPTOSYSTEMS

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

If

$$x = x_1 \| x_2 \| \dots \| x_k$$

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

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

for $i = 1, \dots, k$, where f is a function that "incorporates" encryption function e_k of the cryptosystem, then

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

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 are MD4, MD5 (created by Rivest in 1990 and 1991 and producing 128 bit message digest).

NIST even published, as a standard, in 1993, SHA (Secure Hash Algorithm) – producing 160 bit message digest – based on similar ideas as MD4 and MD5.

A hash function is called secure if it is strongly collision-free.

One of the most important cryptographic results of the last years was due to the Chinese Wang who has shown that MD4 is not cryptographically secure.

RANDOMIZED VERSION of RSA-LIKE CRYPTOSYSTEM

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

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

for any pseudorandom generator

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

and any hash function

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

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 ONCE MORE

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

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

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

   and $\sigma_i = lsb(s_i)$.

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

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

- 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 $y \oplus \sigma_1\sigma_2 \ldots \sigma_m$.

APENDIX

GLOBAL GOALS of CRYPTOGRAPHY

Cryptosystems and encryption/decryption techniques are only one part of modern cryptography.

General goal of modern cryptography is construction of schemes which are robust against malicious attempts to make these schemes to deviate from their prescribed functionality.

The fact that an adversary can design its attacks after the cryptographic scheme has been specified, makes design of such cryptographic schemes very difficult – schemes should be secure under all possible attacks.

In the next chapters several of such most important basic functionalities and design of secure systems for them will be considered. For example: digital signatures, user and message authentication,...

Moreover, also such basic primitives as zero-knowledge proofs, needed to deal with general cryptography problems will be presented and discussed.

We will also discuss cryptographic protocols for a variety of important applications. For example for voting, digital cash,...

BLUM INTEGERS

- An integer n is a Blum integer if $n = pq$, where p, q are primes congruent 3 modulo 4, that is primes of the form $4k + 3$ for some integer k.
- If n is a Blum integer, then each $x \in QR(n)$ has 4 square roots and exactly one of them is in $QR(n)$ – so called principal square root of x modulo n.
- Function $f : QR(n) \rightarrow QR(n)$ defined by $f(x) = x^2 \bmod n$ is a permutation.