# Part VI

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

# 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 infeasible 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 discuss in some details such important cryptography primitives as **pseudo-random number generators** and **hash functions**

# 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 $w < n$

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

**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

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

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 shown in 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 $w_1$, $w_2$, $w_3$, $w_4$.

Rabin did not propose this system as a practical cryptosystem.

# 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 $w < n$

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

**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 $w_1$, $w_2$, $w_3$, $w_4$.

Rabin did not propose this system as a practical cryptosystem.

# COMPUTATION of SQUARE ROOTS MODULO PRIMES

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

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

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

and
then to solve 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

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

# DETAILS and CORRECTNESS of DECRYPTION I

Blum primes $p, q$ form a secret key; $n = pq$ is the public key.

Encryption of a plaintext $w < n$:

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

Decryption: Compute

- $r = c^{(p+1)/4} \bmod p$ and $s = c^{(q+1)/4} \bmod q$;
- Find integers $a, b$ such that $ap + bq = 1$ and compute

$$x = (aps + bqr) \bmod n, \quad y = (aps - bqr) \bmod n$$

- Four square roots of $c \bmod n$ then are (all modulo $n$):

$$x, y, -x, -y$$

- In case $w$ is a meaningful English text, it should be easy to determine $w$ from $x, y, -x, -y$.
- However, this is not the case if $w$ is an arbitrary string.

# DETAILS and CORRECTNESS of DECRYPTION II

- Since $c = w^2 \bmod n$ we have $c \equiv w^2 \pmod{p}$ and $c \equiv w^2 \pmod{q}$;
- Since $r \equiv c^{(p+1)/4}$, we have $r^2 \equiv c^{(p+1)/2} \equiv c^{(p-1)/2}c \pmod{p}$, and Fermat theorem then implies that $r^2 \equiv c \pmod{p}$;
- Similarly, since $s \equiv c^{(q+1)/4}$ we receive $s^2 \equiv c \pmod{q}$;
- Since $x^2 \equiv (a^2p^2s^2 + b^2q^2r^2) \pmod{n}$ and $ap + bq = 1$ we have $bq \equiv 1 \pmod{p}$ and therefore $x^2 \equiv r^2 \pmod{p}$;
- Similarly we get $x^2 \equiv s^2 \pmod{q}$ and the Chinese remainder theorem then implies $x^2 \equiv c \pmod{n}$;
- Similarly we get $y^2 \equiv c \pmod{n}$.

# 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 $\omega$ is a nontrivial square root of 1 modulo $n$, then there are four decryptions of $e(x)$:

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

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 randomized 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 $q^{mj} \bmod p, \quad 0 \leq j \leq m-1$.
2. Create list $L_1$ of $m$ pairs $(j, q^{mj} \bmod p)$, sorted by the second item.
3. Compute $yq^{-i} \bmod p, \quad 0 \leq i \leq m-1$.
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

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

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

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

are all distinct, we have that

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

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 $y^{(p-1)/2} \equiv 1 \text{ mod } p$
$L_1(y)$ can therefore be computed as follows:

$$\begin{array}{ll} L_1(y) = 0 & \text{if } y^{(p-1)/2} \equiv 1 \text{ mod } p; \\ L_1(y) = 1 & \text{otherwise} \end{array}$$

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 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**: $\alpha, \beta$
**Trapdoor**: $k$ such that $\alpha^k = \beta$

**Encryption:** of a plaintext $w$ and a random integer $k$

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

**Decryption**: of cryptotext $(y_1, y_2)$:

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

# 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

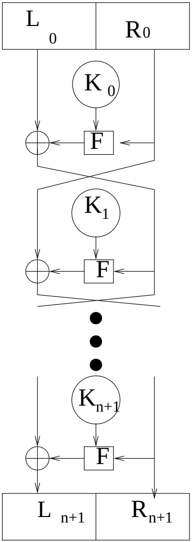
$$L_{i+1} = R_i; \;\; R_{i+1} = L_i \oplus F(R_i, k_i)$$

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: $e_k : P \times R \rightarrow C$

decryption: $d_k : C \rightarrow P$ or $C \rightarrow 2^P$ 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 $d_k$.

# 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:$N \rightarrow R$ 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

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

is a negligible function in $n$.

# 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

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

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 $\mu$ 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.

## PSEUDORANDOM GENERATORS

Psedorandom 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 availaable.

# 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 rally 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".
Repeat the following process:
compute $x^2$ and take sequence of the middle digits of $x^2$ as a new "seed" $x$.

# A SIMPLE PSEUDORANDOM GENERATORS

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?

# SECURE ENCRYPTIONS – PSEUDORANDOM GENERATORS

In cryptography random sequences can 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.

**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 $x^2 \bmod n$ 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 (computationally indistinguishable) pseudorandom generator with stretch function $l(n)$.

# 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.

## PERFECTLY SECURE ENCRYPTION ALGORITHMS - EXAMPLES

# 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 $s_0 \in \{0, 1, \dots, n\}$.
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 $y \oplus \sigma_1\sigma_2 \dots \sigma_m$.

HASH FUNCTIONS

Hash functions have numerous applications in cryptography and in computing in general, in searching and in the design of efficient randomized algorithms.

The concept of hash functions appered in the 1950's, but the search for good and/or secure hash functions continue.

# HASH FUNCTIONS BASICS

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, especially those that minimize impacts of potential **collisions** - the cases two elements are mapped into the same one.

Hash functions have several very important uses in cryptography:

- In integrity of data protection. They can reduce the problem of protection of huge data to the protection of small hashes.
- In making efficient data signatures. They can reduce the problem of creating long signatures of huge date to short signatures of their hashes.
- In making commitments, in authentication of communicating parties and in creating cryptographically perfect (pseudo) randomness

.

# 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) of a message $w$, where $y = sig_k(h(w))$ and she would be able to find a 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 (collision-resistant) property

**Definition 1.** Let w be a message. A hash function h is weakly collision-resistant (collision-free)for w, if it is computationally infeasible to find a $w' \neq w$ 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 (resistant) property

**Definition 2.** A hash function h is strongly collision-resistant (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:

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 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 function 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.

# AN ALMOST GOOD HASH FUNCTION

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

Let p be a large prime such that $q = \frac{(p-1)}{2}$ is also prime and let $\alpha, \beta$ 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 $m = x_0 + x_1 q, 0 \leq x_0, x_1 \leq q - 1$ 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

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

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 $i = 1, \dots, m$, where f is a function that "incorporates" encryption functions $e_k$ of the cryptosystem, for suitable keys $k$, 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 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.

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 brute force attack.

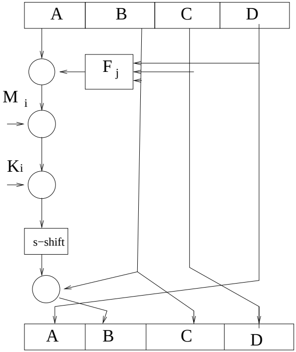
# MD5

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 block, divided into 16 words (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 word that is 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 precessing of a message consists of four rounds, $j$-th round is composed of 16 similar operations using a non-linear function $F_j$ and a left rotation by $s$ places where $s$ varies for each round - see next figure.



## 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 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 that in order no two persons 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 $n = 23$.

# FINDING COLLISIONS USING BIRTHDAY PARADOX

If the hash of a hash function $h$ has 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 first 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 $n$ and a real $\theta$
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 $(y, x)$ in the hash table**
**6.Otherwise search failed**

Theorem If we pick numbers 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%)

# 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).

# APPENDIX

# 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, $\alpha$ can be viewed as a pair $(a, b)$.

$$\alpha_1 + \alpha_2 = (a_1, b_1) + (a_2, b_2) = (a_1 + a_2, b_1 + b_2)$$

$$\alpha_1\alpha_2 = (a_1, b_1) \cdot (a_2, b_2) = (a_1 a_2 + \mathrm{c}\, b_1 b_2, a_1 b_2 + b_1 a_2)$$

The conjugate $\overline{\alpha}$ of $\alpha$ 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 $j \geq i$

$$X_{I+J} = 2X_I X_J + X_{J-1}$$
$$Y_{I+J} = 2Y_I X_J + Y_{J-1}$$

From these and following equations:

$$X_{I+J} = 2X_I X_J + cY_I Y_J$$
$$Y_{I+J} = 2Y_I X_J + X_I Y_J$$

we get the recursive formulas:

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

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 $b$.

# WILLIAMS CRYPTOSYSTEM - BASIC LEMMAS

Congruences on numbers of type $a+b\sqrt{c}$ are defined: $a_1+b_1\sqrt{c}\equiv a_2+b_2\sqrt{c}(\mod n) \Leftrightarrow a_1\equiv a_2(\mod n), b_1\equiv b_2(\mod n)$ Instead of $a^2-cb^2=1$ we will consider congruence $a^2-cb^2\equiv 1(\mod n)$

Basic Lemma: Let $n=p\cdot q$ (both primes) and let $a,b,c$ be such that $a^2-cb^2\equiv 1$ ($\mod n$). Moreover, let the Jacobi-Legendre symbols

$$\varepsilon_p=(c|p), \varepsilon_q=(c|q)$$

satisfy the congruence

$$\varepsilon_i\equiv -i(\mod 4) \text{ for } i\in\{p,q\}.$$

Assume also that $\gcd(cb,n)=1$ and $(2(a+1)|n)=1$.
Denote

$$m=\frac{(p-\varepsilon_p)(q-\varepsilon_q)}{4}$$

and assume that $e$ and $d$ satisfy the congruence

$$ed\equiv\frac{(m+1)}{2}\mod m.$$

Under these assumptions

$$\alpha^{2ed}\equiv\pm\alpha(\mod n),$$

where

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

.

# DESIGN of WILLIAMS CRYPTOSYSTEMS

Choose $p, q$, compute $n = pq$.
Choose $c$ such that Jacobi-Legendre symbols $\varepsilon_p, \varepsilon_q$ satisfy congruences of previous lemma ($c$ can be chosen by a trial).
Choose (by trial) a number $s$ such that

$$(s^2 - c|n) = -1, \quad \gcd(n, s) = 1.$$

Let $m$ be as in Basic lemma and $d$ be such that $\gcd(m, d) = 1$ and let $e$ be such that

$$ed \equiv \frac{(m+1)}{2} \mod m.$$

Public key: $n, e, c, s$
Secret key: $p, q, m, d$
Encryption: A plaintext $0 < w < n$ will first be encoded as a number $\alpha_w$ of the form $a + b\sqrt{c}$.
Denote

$$b_1 = 0, \gamma = w + \sqrt{c} \text{ if } (w^2 - c|n) = 1$$
$$b_1 = 1, \gamma = (w + \sqrt{c})(s + \sqrt{c}) \text{ if } (w^2 - c|n) = -1$$

In both cases:

$$(\gamma\bar{\gamma}|n) = 1.$$

Define: $\alpha = \gamma\bar{\gamma}^{-1} = \frac{\gamma}{\bar{\gamma}}$
(1) if $b_1 = 0$, then $\alpha \equiv \frac{w^2+c}{w^2-c} + \frac{2w}{w^2-c}\sqrt{c} \pmod n$

# DECRYPTION

Decryption: cryptotext: $(E, b_1, b_2)$, where $E = (X_e(\alpha)Y_e(\alpha)^{-1} \mod n)$, $b_2 \in \{0,1\}$, depending whether $a$ is even or odd.
Decryption: Using $E$ the receiver may compute:

$$\begin{aligned}\alpha^{2e} &\equiv \frac{\alpha^{2e}}{(\alpha\bar{\alpha})^e} \equiv \frac{\alpha^e}{\bar{\alpha}^e} = \frac{X_e(\alpha) + Y_e(\alpha)\sqrt{c}}{X_e(\alpha) - Y_e(\alpha)\sqrt{c}} \\ &\equiv \frac{E+\sqrt{c}}{E-\sqrt{c}} = \frac{E^2+c}{E^2-c} + \frac{2E}{E^2-c}\sqrt{c} (\mod n)\end{aligned}$$

(The above computation can perform also a cryptanalyst. Trapdoor is needed for the next computation.)

$$\alpha^{2ed} = X_{2ed}(\alpha) + Y_{2ed}(\alpha)\sqrt{c} = X_d(\alpha^{2e}) + Y_d(\alpha^{2e})\sqrt{c}$$

Now all assumptions of Basic lemma are satisfied and, consequently

$$\alpha^{2ed} \equiv \pm\alpha (\mod n)$$

$b_2$ is then used to determine which of the above signs is correct.
$w$ is now obtained as follows:
Denote:

$$\alpha' = \alpha \text{if } b_1 = 0 \text{and } \frac{s-\sqrt{c}}{s+\sqrt{c}} \text{ if} b_1 = 1$$

Then

$$\alpha' \equiv \frac{w+\sqrt{c}}{\quad} (\mod n)$$

# 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 PRIMES and 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 $k$. Such primes are also called Blum primes.
- 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.

# UNIVERSAL HASHING SCHEMES

A universal hashing scheme is a randomized algorithm that selects a hashing function among a family of hashing functions, in such a way that probability of collision of any two distinct keys is $1/n$, where $n$ is the number of distinct hashes desired – independently of the keys.

Universal hashing ensures - in a probabilistic sense - that the hash function application will behave as if it were using a random function, for any distribution of the input data.

**Theorem** The family of functions $emH = \{h_a \mid a \in \{0, \ldots, m-1\}^{r+1}$, defined by the formula

$$h_a(u) = \sum_{i=0}^{r} a_i u_i \mod m$$

is a universal family of hash functions mapping $\{0, \ldots, m-1\}^{r+1}$ into $\{0, \ldots, m-1$.