Steam cipher Cryptography

University of Technology
Computer science department
2nd Class-first course
Professor
Dr. Alaa .K adhim

Computers are now fond in every layer society, and information is being communicated and processed automatically on a large scale. Such as medical and financial files, automatic banking, video-phones, pay-tv, facsimiles, tele-shopping, and global computer networks. In all these cases there is a growing need for the protection of information to safeguard economic interests, to prevent fraud and to ensure privacy.

2024-2025

Lec-1-

1. Intoduction of Cryptography

Computers are now fond in every layer society, and information is being communicated and processed automatically on a large scale. Such as medical and financial files, automatic banking, video-phones, pay-tv, facsimiles, tele-shopping, and global computer networks. In all these cases there is a growing need for the protection of information to safeguard economic interests, to prevent fraud and to ensure privacy.

The term Cryptography is originally derived from the two greek words “kryptos" and “graph", meaning hidden and writing. This is an accurate representation of the meaning of the word, as cryptography is the art of ensuring that messages (writing) are kept secure (hidden) from those recipients to whom the messages are not addressed .

Cryptography is the science and study of methods of protecting data in computer and communication systems from unauthorized disclosure and modification.

The Cryptographic systems are classified into two cryptosystems, private-key cryptosystem and public-key cryptosystem. Both are based on complex mathematical algorithms and are controlled by keys.

The advances in cryptography have boosted its use in recent decades, opening up an amazing array of applications. It can be used to authenticate computer users, ensure the integrity and confidentiality of electronic communications, and keep sensitive information safely stored. From a secretive military technology, cryptography has emerged a key technology for all participants in the information society concerned about information securit.

2. History of Cryptography

cryptography arose as a means to enable parties to maintain privacy of the information they send to each other, even in the presence of an adversary with access to the communication channel. While providing privacy there remains a central goal, the field has expanded to encompass many others, including not just other goals of communication security, such as guaranteeing, integrity and authenticity of communications, but many more sophisticated and fascinating goals. Cryptography is a discipline of mathematics and computer science concerned with information security and related issues, particularly encryption, authentication, and such applications as access Control. Cryptography, as an interdisciplinary subject, draws on several fields. Prior to the early 20th century, cryptography was chiefly concerned with Linguistic patterns. Since then, the emphasis has shifted, and Cryptography now makes extensive use of mathematics, including topics from information theory, computational complexity, statistics, combinatory, and especially number theory.

3. Complexity Theory

Modern cryptography introduces a new dimension: the amount of computing power available to an attacker. It seeks to have security as long as attackers don't have "too much" computing time. Schemes are breakable "in principle," but not in practice. Attacks are infeasible, not impossible. This is a radical shift from many points of view. It takes cryptography from the realm of information theory into the realm of computer science, and complexity theory in particular, since that is where its can be studied how hard, problems are to solve as a function of the computational resources invested. In addition, it changes what can efficiently achieve. It will be wanted to make statements like this: Assuming the attacker uses no more than computing cycles .

Notice again the statement is probabilistic. Almost all of our statements will be. Notice another important thing. Nobody said anything about how the attacker operates. What algorithm, or technique, does he use? Anything about that is not known. The statement holds nonetheless. So it is a very strong statement. It should be clear that, in practice, a statement like the one above would be good enough. As the attacker works harder, his chance of breaking the scheme increases, and if the attacker had 2200 computing cycles at his disposal, no security would be had left at all. But nobody has that much of computing power. Now we must ask ourselves how can hope to get protocols with such properties? The legitimate parties must be able to efficiently execute the protocol instructions: their effort should be reasonable. Somehow, the task of the attacker must be harder. The basic ingredient in any cryptosystem

is a difficult computational problem. Now it can be introduced some relevant terminology used in cryptography .

• **Algorithm**. An algorithm is an explicit description of how a particular computation should be performed (or a problem solved). The efficiency of an algorithm can be measured as the number of elementary steps it takes to solve the problem. Usually one has to settle for the asymptotic running time,

which can be expressed using the big -*O* notation: The algorithm is considered as *O(f(n))*, if its worst-case running time divided by *f(n)* is bounded by a fixed (positive) constant as the input size *n* increases.

• **Computational Complexity**. A problem is *polynomial* time or *in* **P** if it can be solved by an algorithm which takes less than *O(nt)* steps, where *t* is some finite number and the variable *n* measures the size of the problem instance. If a guessed solution to a problem can be verified in polynomial time then the problem is said to be *in* **NP** (non-deterministic polynomial time). The set of problems that lie in **NP** is very large; it includes the problem of integer factorization. A problem is **NP**-*hard* if there is no other problem in **NP** that is easier to solve. There is no known polynomial time algorithm for any **NP**-hard problem, and it is believed that such algorithms in fact do not exist. In public-key cryptography, the attacker is interested in solving particular instances of a problem (factoring some given number), rather than providing a general solution (an algorithm to factor any possible number efficiently). This causes some concern for cryptographers, as some instances of a problem that is **NP**-hard in general may be easily solvable.

• **Factoring**. Every integer can be represented uniquely as a product of prime numbers. For example, *10 = 2 * 5* (the notation * is common for multiplication in computer science) and it is unique (except for the order of the factors *2* and *5*). The art of factorization is almost as old as mathematics itself. However, the study of fast algorithms for factoring is only a few decades old. One possible algorithm for factoring an integer is to divide the input by all small prime numbers iteratively until the remaining number is prime. This is efficient only for integers that are saying of size less than 10^{16} as this already require trying all primes up to 10^{8}. In public-key cryptosystems based on the problem of factoring, numbers are of size 10^{300} and this would require trying all primes up to 10^{150} and there are about 10^{147} such prime numbers according to the prime number theorem. This far exceeds the number of atoms in

the universe, and is unlikely to be enumerated by any effort. The easy instance of factoring is the case where the given integer has only small prime factors. For example, *759375* is easy to factor as we can write it as $3^5 * 5^5$. In cryptography we want to use only those integers that have only large prime factors. Preferably, is selecting an integer with two large prime factors, as is done in the RSA cryptosystem. Currently one of the best factoring algorithms is the number field sieve algorithm (NFS) that consists of a sieving phase and a matrix step. The sieving phase can be distributed (and has been

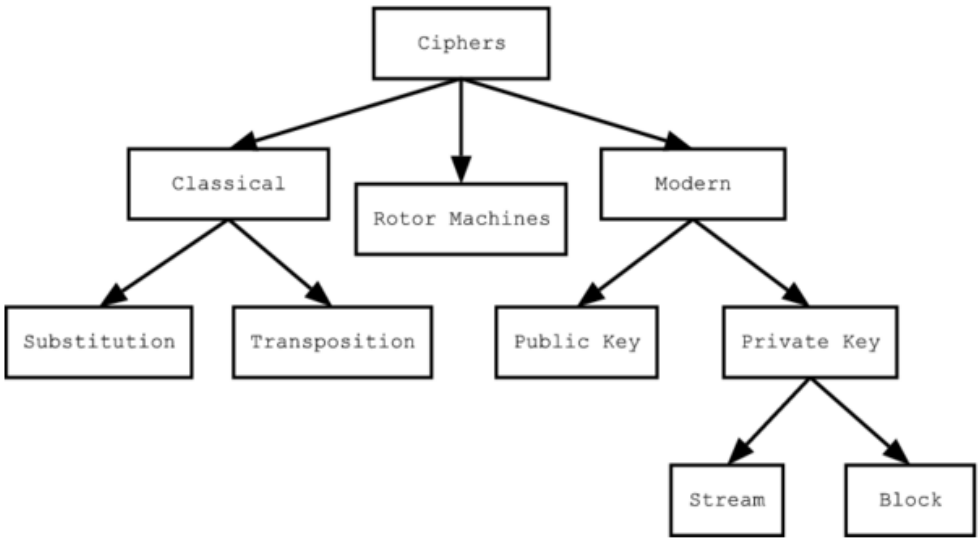
several times) among a large number of participants, but the matrix step needs to be performed on large supercomputers. The effectiveness of the NFS algorithm becomes apparent for very large integers; it can factor any integer of size 10^{150} in a few months time. The NFS algorithm takes sub-exponential time (which is still not very efficient). There is no known proof that integer factorization is an **NP**-hard problem nor that it is not polynomial time solvable. If any **NP**-hard problem were polynomial time solvable, then also factoring would, but there is very little hope that this is the case. It is plausible under current knowledge that factoring is not polynomial time solvable **.**

• **Discrete logarithms**. Another important class of problems is the problem of finding *n* given only some *y* such that $y = g^n$. The problem is easy for integers, but when the work is in a slightly different setting, it becomes very hard. To obscure the nature of *n* in g^n, divide the infinite set of integers into a finite set of *remainder classes*. Intuitively, we take the string of integers and wrap it on a circle (which has circumference of length *m*). The numbers *0, m, 2m, 3m, ...* all cover the same point on the circle, and therefore are said to be in the same equivalence class (we also write "*0 = m = 2m = ... (mod m)*"). Each equivalence class has a least representative in *0 .. m-1*. So you can write any integer *n* as *t + km* for any integer *t*, where *0 <= t < m*. It is a convention to write *n = t (mod m)* in this case. Here *m* is said to be the *modulus*.

It can be shown that you can add, subtract and multiply with these Classes of integers (modulo some *m*). This structure, when *m = p* with *p* a prime number, is often called a prime field or even harder to solve the discrete logarithm problem over elliptic curves than over GF(*p*). This has also the effect that there are some key size benefits for using elliptic-curve-based public-key cryptosystems as opposed to factoring-based cryptosystems.

4. Types

There are a variety of different types of encryption. Algorithms used earlier in the history of cryptography are substantially different from modern methods, and modern ciphers can be classified according to how they operate and whether they use one or two keys.



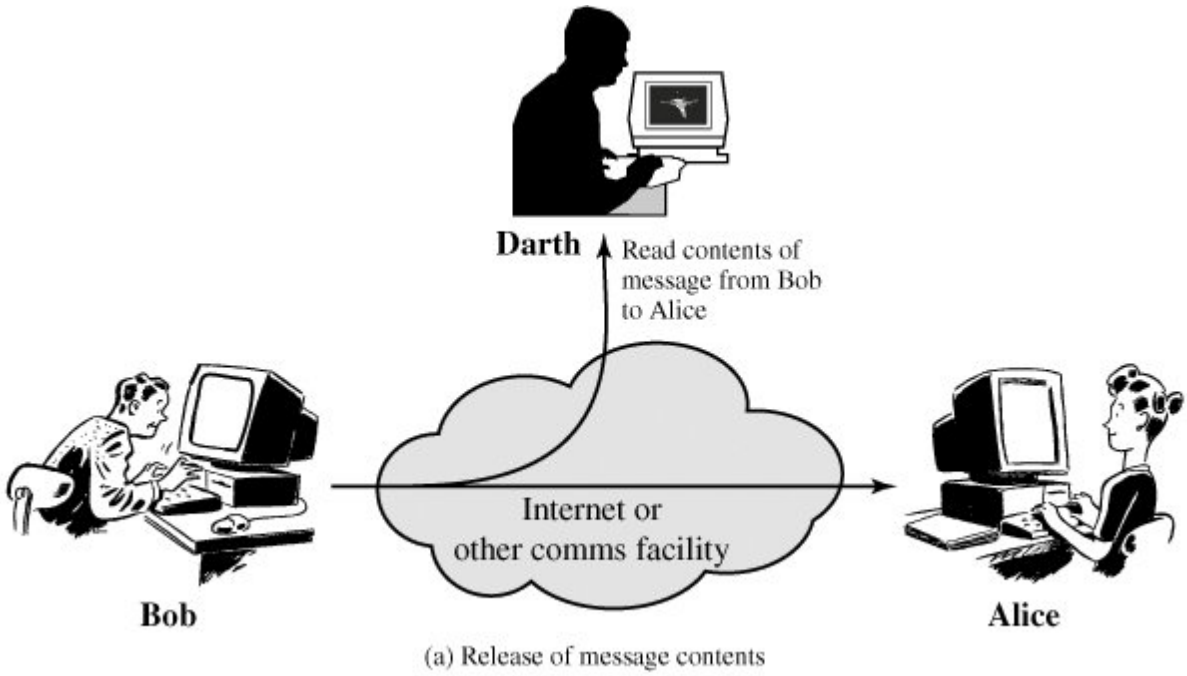
5. Security Attacks

A useful means of classifying security attacks, used both in X.800 and RFC 2828, is in terms of passive attacks and active attacks. A passive attack attempts to learn or make use of information from the system but does not affect system resources. An active attack attempts to alter system resources or affect their operation.

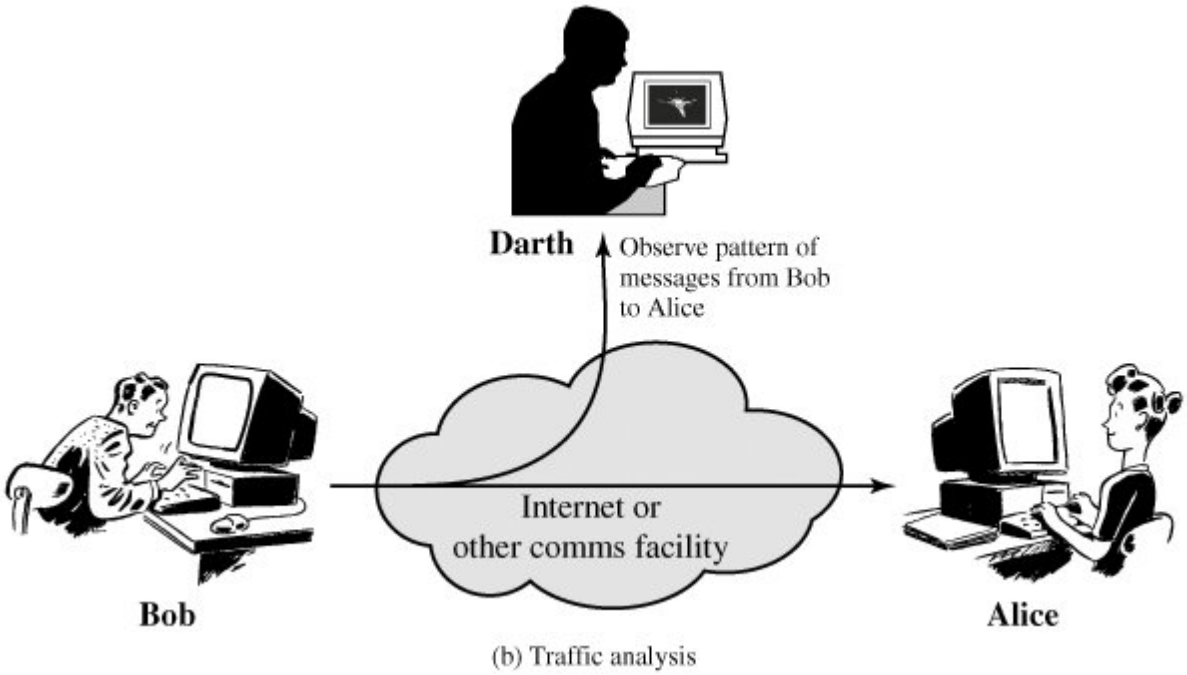
✓ *Passive Attacks*

Passive attacks are in the nature of eavesdropping on, or monitoring of, transmissions. The goal of the opponent is to obtain information that is being transmitted. Two types of passive attacks are **release of message contents and traffic analysis.**

The release of message contents is easily understood (following Figure a). A telephone conversation, an electronic mail message, and a transferred file may contain sensitive or confidential information. We would like to prevent an opponent from learning the contents of these transmissions.



(a) Release of message contents



(b) Traffic analysis

A second type of passive attack, **traffic analysis**, is subtler (above b). Suppose that we had a way of masking the contents of messages or other information traffic so that opponents, even if they captured

the message, could not extract the information from the message. The common technique for masking contents is encryption. If we had encryption protection in place, an opponent might still be able to

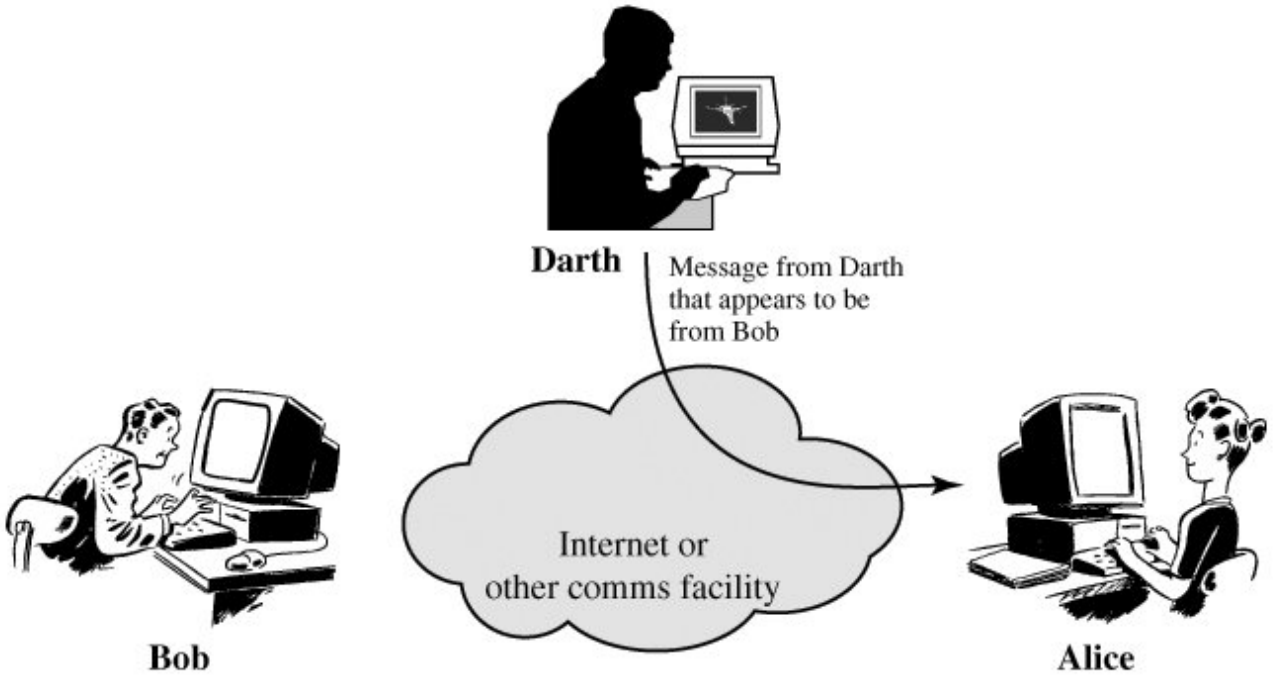
observe the pattern of these messages. The opponent could determine the location and identity of communicating hosts and could observe the frequency and length of messages being exchanged. This information might be useful in guessing the nature of the communication that was taking place.

Passive attacks are very difficult to detect because they do not involve any alteration of the data. Typically, the message traffic is sent and received in an apparently normal fashion and neither the sender nor receiver is aware that a third party has read the messages or observed the traffic pattern. However, it is feasible to prevent the success of these attacks, usually by means of encryption. Thus, the emphasis in dealing with passive attacks is on prevention rather than detection.

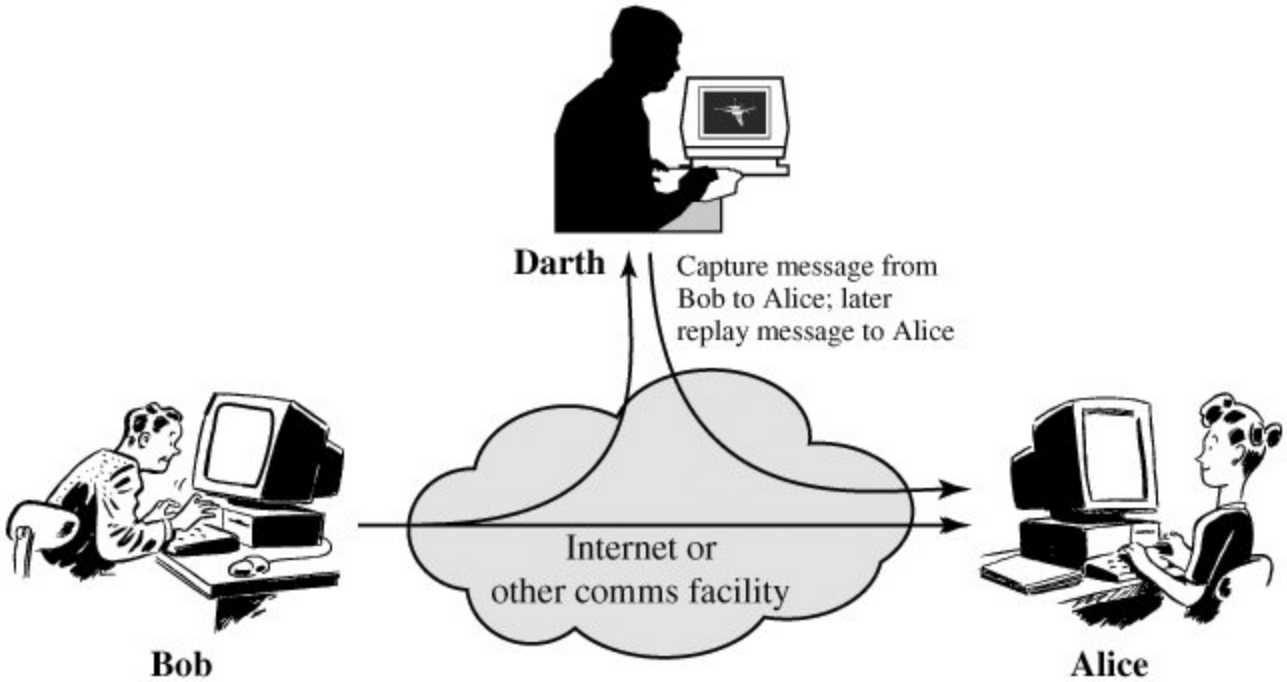
- ✓ ***Active Attacks***

Active attacks involve some **modification of the data stream** or the creation of a false stream and can be subdivided into four categories: masquerade, replay, modification of messages, and denial of service.

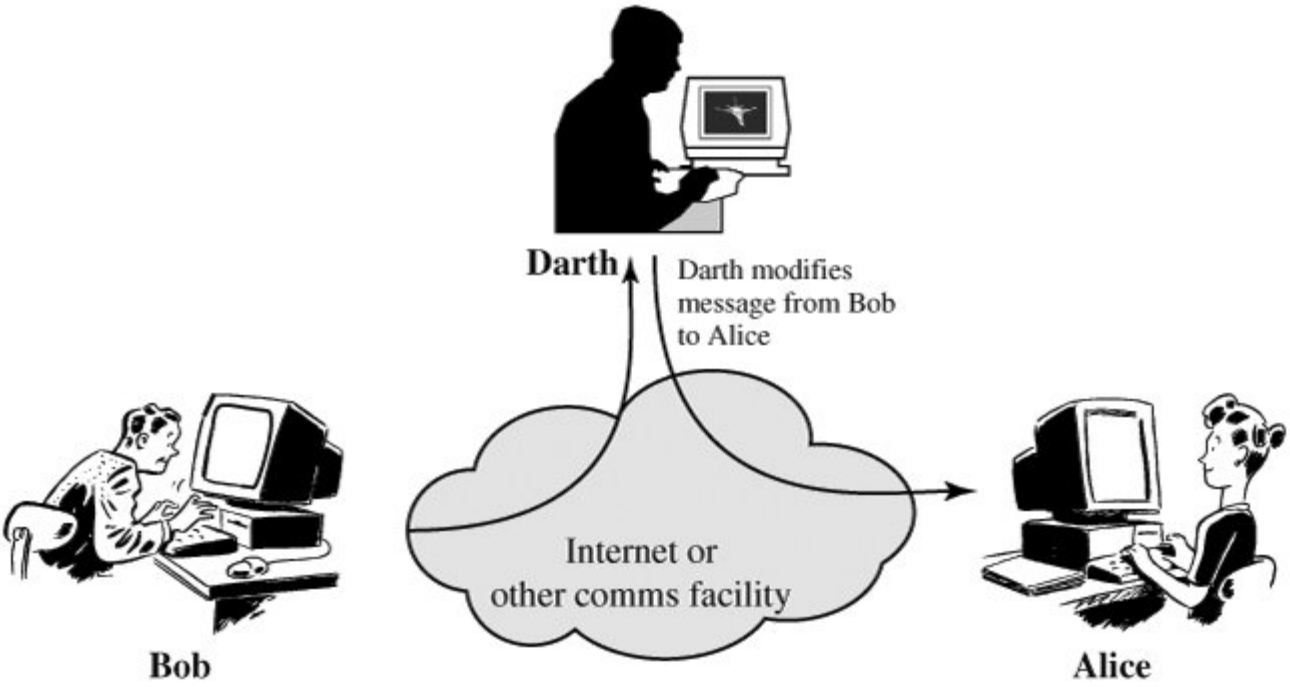
A **masquerade** takes place when one entity pretends to be a different entity (following Figure a). A masquerade attack usually includes one of the other forms of active attack. For example, authentication sequences can be captured and replayed after a valid authentication sequence has taken place, thus enabling an authorized entity with few privileges to obtain extra privileges by impersonating an entity that has those privileges.



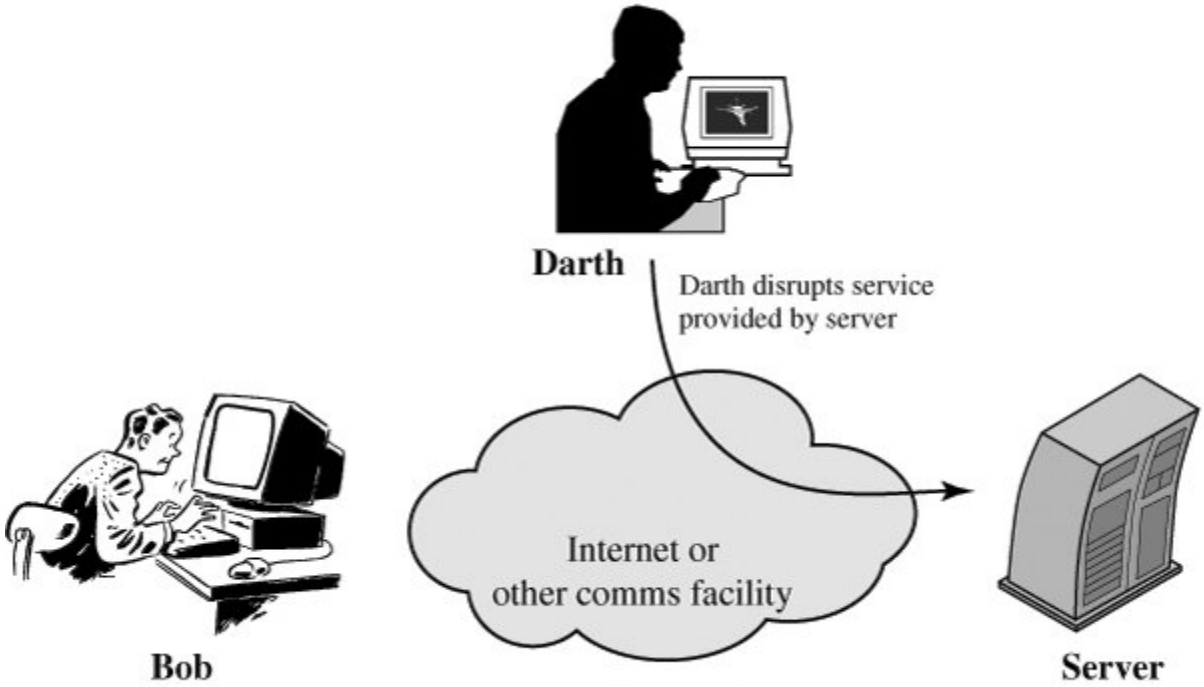
(a) Masquerade



(b) Replay



(c) Modification of messages



(d) Denial of service

Replay involves the passive capture of a data unit and its subsequent retransmission to produce an unauthorized effect(Figure b)

Modification of messages: simply means that some portion of a legitimate message is altered, or that messages are delayed or reordered, to produce an unauthorized effect (Figure c). For example, a message meaning "Allow John Smith to read confidential file accounts" is modified to mean "**Allow Fred Brown to read confidential file accounts.**"

***Denial of service*:** prevents or inhibits the normal use or management of communications facilities (Figure d). This attack may have a specific target; for example, an entity may suppress all messages directed to a particular destination (e.g., the security audit service). Another form of service denial is the disruption of an entire network, either by disabling the network or by overloading it with messages so as to degrade performance.

Active attacks present the opposite characteristics of passive attacks. Whereas passive attacks are difficult to detect, measures are available to prevent their success. On the other hand, it is quite difficult to prevent active attacks absolutely, because of the wide variety of potential physical, software, and network vulnerabilities. Instead, the goal is to detect active attacks and to recover from any disruption or delays caused by them. If the detection has a deterrent effect, it may also contribute to prevention.

Lec-2-
Steam Cipher

Topics

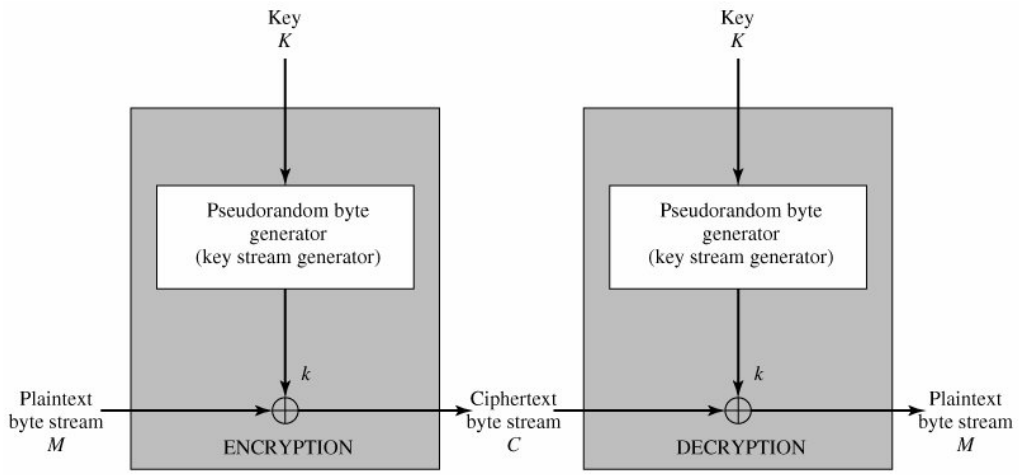
2.1 Stream Cipher Structure

A typical stream cipher encrypts plaintext one byte at a time; although a stream cipher may be designed to operate on one bit at a time or on units larger than a byte at a time. In the following figure is a representative diagram of stream cipher structure. In this structure a key is input to a **pseudorandom** bit generator that produces a stream of 8-bit numbers that are apparently random. For now, we simply say that a pseudorandom stream is one that is unpredictable without knowledge of the input key. The output of the generator, called a keystream, is combined one byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR) operation. For example, if the next byte generated by the generator is 01101100 and the next plaintext byte is 11001100, then the resulting ciphertext byte is

```
  11001100   plaintext
⊕ 01101100   key stream
  10100000   ciphertext
```

Decryption requires the use of the **same** pseudorandom sequence:

```
  10100000   ciphertext
⊕ 01101100   key stream
  11001100   plaintext
```



Stream Cipher Diagram

***Note*:** The stream cipher is similar to the **one-time pad** discussed .The difference is that a one-time pad uses a **genuine**random number stream, whereas a stream cipher uses a **pseudo**random number stream.

2.2 Important element for design a stream cipher

1. The **encryption** sequence should have a **large period.** A pseudorandom number generator uses a function that produces a deterministic stream of bits that eventually repeats. The longer the period of repeat the more difficult it will be to do cryptanalysis. This is essentially the same consideration that was discussed with

reference to the Vigenère cipher, namely that the longer the keyword the more difficult the cryptanalysis.

2. The **keystream** should approximate the properties of a true **random** number stream as close as possible. For example, there should be an approximately equal **numbers of 1s and 0s**. If the keystream is treated as a stream of bytes, then all of the 256 possible byte values should appear approximately equally often. The more random-appearing the keystream is, the more randomized the ciphertext is, making cryptanalysis more difficult.
3. the **output** of the pseudorandom number generator is conditioned on the value of the **input key**. To guard against brute-force attacks, the key needs to be sufficiently long. The same considerations as apply for block ciphers are valid here. Thus, with current technology, a key length of at **least 128 bits** is desirable.

Note: Pseudorandom Number Generators (PRNGs)

Cryptographic applications typically make use of algorithmic techniques for random number generation. These algorithms are deterministic and therefore produce sequences of numbers that are not statistically random. However, if the algorithm is **good**, the resulting sequences will pass many reasonable tests of randomness. Such numbers are referred to as **pseudorandom numbers**.

2.3 Types of stream ciphers

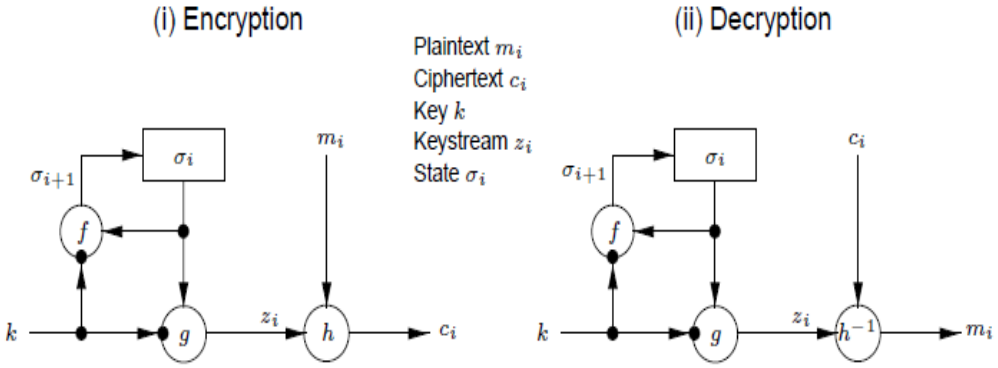
A stream cipher generates successive elements of the **keystream** based on an **internal** state. This state is updated in essentially two ways: if the state changes independently of the **plaintext or ciphertext** messages, the cipher is classified as a synchronous stream cipher. By contrast, self-synchronising stream ciphers update their state based on previous ciphertext digits.

- **Synchronous stream ciphers**

Definition: A *synchronous* stream cipher is one in which the keystream is generated **independently** of the plaintext message and of the ciphertext. The encryption process of a synchronous stream cipher can be described by the equations:

$$\begin{aligned} \sigma_{i+1} &= f(\sigma_i, k), \\ z_i &= g(\sigma_i, k), \\ c_i &= h(z_i, m_i), \end{aligned}$$

where σ_0 is the **initial** state and may be determined from the key **k**, **f** is the next-state function, **g** is the function which produces the key stream zi, and **h** is the output function which combines the keystream and plaintext **mi** to produce cipher text **ci**. The encryption and decryption processes are depicted in following figure. The **OFB** mode of a block cipher



- **Note** (*properties of synchronous stream ciphers*)

(i) ***synchronization requirements***. In a synchronous stream cipher, both the sender and receiver must be *synchronized* – using the **same key** and **operating at the same position (state)** within that key – to allow for proper decryption. If synchronization is lost due to ciphertext digits being inserted or deleted during transmission, then decryption **fails** and can only be restored through additional techniques for re-synchronization. Techniques for re-synchronization include re-initialization, placing special markers at regular intervals in the ciphertext, or, if the plaintext contains enough redundancy, trying all possible keystream offsets.

(ii) ***No error propagation.*** A ciphertext digit that is modified (but not deleted) during transmission does **not affect** the decryption of **other** ciphertext digits.

(iii) ***Active attacks.*** As a consequence of property (i), the insertion, deletion, or replay of ciphertext digits by an active adversary causes immediate loss of synchronization, and hence might possibly be detected by the decrypted.

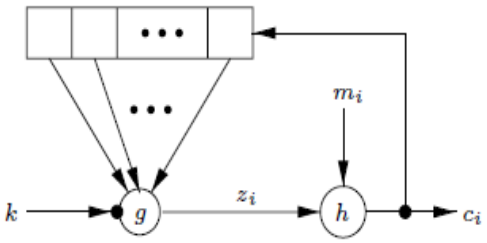
❖ Self-synchronizing stream ciphers

Definition: A *self-synchronizing* or *asynchronous* stream cipher is one in which the **keystream** is generated as a function of the key and a **fixed number of previous ciphertext** digits. The encryption function of a self-synchronizing stream cipher can be described by the equations:

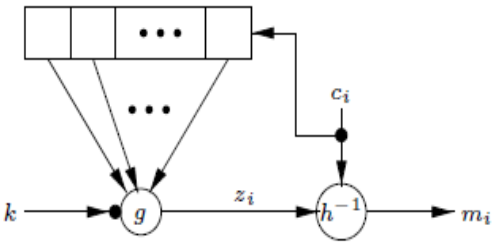
$$\begin{aligned} \sigma_i &= (c_{i-t}, c_{i-t+1}, \cdots, c_{i-1}), \\ z_i &= g(\sigma_i, k), \\ c_i &= h(z_i, m_i), \end{aligned}$$

where σ_0 = (c−t; c−t+1; …. ; c−1) is the (non-secret) *initial state*, **k** is the *key*, **g** is the function which produces the *keystream* **zi**, and **h** is the *output function* which combines the **keystream and plaintext** mi to produce ciphertext ci. The encryption and decryption processes are depicted in following Figure. The most common presently-used self synchronizing stream ciphers are based on block ciphers in 1-bit cipher feedback mode

(i) Encryption



(ii) Decryption



Note (*properties of self-synchronizing stream ciphers*)

(i) ***self-synchronization***. Self-synchronization is possible if ciphertext digits are deleted or inserted, because the decryption mapping depends only on a fixed number of preceding **ciphertext characters**. Such ciphers are capable of re-establishing proper decryption automatically after loss of synchronization, with only a fixed number of plaintext characters unrecoverable.

(ii) ***Limited error propagation.*** Suppose that the state of a self-synchronization stream cipher depends on **t** previous ciphertext digits. If a single ciphertext digit is **modified** (or even deleted or inserted) during transmission, then decryption of up to **t** subsequent ciphertext digits may be **incorrect**, after which correct decryption resumes.

(iii) ***Active attacks.*** Property (ii) implies that any modification of ciphertext digits by an active adversary causes several other ciphertext digits to be decrypted incorrectly, thereby improving (compared to synchronous stream ciphers) the likelihood of being detected by the decryptor. As a consequence of property (i), **it is more difficult** (than for synchronous stream ciphers) to **detect insertion, deletion, or replay of ciphertext digits** by an active adversary. This illustrates that additional mechanisms must be employed in order to provide data origin authentication and data integrity guarantees.

(iv) ***Diffusion of plaintext statistics.*** Since each plaintext digit influences the entire following ciphertext, the statistical properties of the plaintext are dispersed through the ciphertext. Hence, self-synchronizing stream ciphers may be more **resistant** than synchronous stream ciphers against attacks based on plaintext redundancy.

Lec-3-

Polynomial

Topics

2.1. Polynomial

2.2. Polynomial Arithmetic

2.3. Primitive polynomial

2.4. Irreducible Polynomial

3.1 Polynomial

In mathematics, a polynomial is an expression of finite length constructed from variables (also known as indeterminates) and constants, using only the operations of addition, subtraction, multiplication, and non-negative, whole-number exponents. For example, $\mathbf{x^2 - 4x + 7}$ is a polynomial, but $\mathbf{x^2 - 4/x + 7x^{3/2}}$ is not, because its second term involves division by the variable x (4/x) and because its third term contains an exponent that is not a whole number (3/2). The term 'polynomial' indicates a simplified algebraic form such that all polynomials are similarly simple in complexity (cf. polynomial time).

Polynomials appear in a wide variety of areas of mathematics and science. For example, they are used to form polynomial equations, which encode a wide range of problems, from elementary word problems to complicated problems in the sciences; they are used to define polynomial functions, which appear in settings ranging from **basic chemistry** and **physics** to **economics** and **social science**; they are used in calculus and numerical analysis to approximate other functions. In advanced mathematics, polynomials are used to construct polynomial rings, a central concept in abstract algebra and algebraic geometry**.**

A polynomial is either zero, or can be written as the sum of one or more non-zero terms. The number of terms is finite. These terms consist of a constant (called the coefficient of the term) which may be multiplied by

a finite number of variables (usually represented by letters). Each variable may have an exponent that is a non-negative integer, i.e., a natural number. The exponent on a variable in a term is called the **degree** of that variable in that term, the degree of the term is the sum of the degrees of the variables in that term, and the degree of a polynomial is the largest degree of any one term. Since $x = x^1$, the degree of a variable without a written exponent is one. A term with no variables is called a constant term, or just a constant. The degree of a constant term is **0**. The coefficient of a term may be any number from a specified set. If that set is the set of real numbers, we speak of "**polynomials over the real's**". Other common kinds of polynomials are polynomials with integer coefficients, polynomials with **complex coefficients**, and polynomials with coefficients that are integers modulo of some prime number p. In most of the examples in this section, the coefficients are integers.

For example:

$$\mathbf{-5x^2y}$$

Is a term. The coefficient is –5, the variables are x and y, the degree of x is two, and the degree of y is one.The degree of the entire term is the **sum of the degrees** of each variable in it, so in this example the degree is 2 + 1 = 3.

A polynomial is a sum of terms. For example, the following is a polynomial**:**

$$\mathbf{3x^2-5x+4}$$

t consists of three terms: the first is degree two, the second is degree one, and the third is degree zero.

In polynomials in one variable, the terms are usually ordered according to degree, either in "**descending powers of x**", with the term of largest degree first, or in "**ascending powers of x**". The polynomial in the example above is written in descending powers of x. The first term has coefficient **3**, variable **x**, and exponent **2**. In the second term, the coefficient is **–5**. The third term is a **constant**. Since the degree of a non-zero polynomial is the largest degree of any one term, this polynomial has **degree two**.

- Characteristic Polynomial: is the polynomial define by

$$F(x)=1+C_1x^1+C_1x^2+\ldots\ldots+C_nx^n$$

Let $C_0=1$

3.2 Polynomial Arithmetic

Before pursuing our discussion of finite fields, we need to introduce the interesting subject of polynomial arithmetic. We are concerned with polynomials in a single variable x, and we can distinguish three classes of polynomial arithmetic:

- **Ordinary polynomial arithmetic**, using the basic rules of algebra
- Polynomial arithmetic in which the arithmetic on the coefficients is performed **modulo p**; that is, the coefficients are in GF(p)
- Polynomial arithmetic in which the coefficients are in GF(p), and the polynomials are defined modulo a **polynomial m(x)** whose highest power is some integer n

3.2.1 Ordinary Polynomial Arithmetic

A **polynomial** of degree n (integer n >=0) is an expression of the form

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n} a_i x^i$$

where the a_i are elements of some designated set of numbers S, called the **coefficient set**, and a_n <>0. We say that such polynomials are defined over the coefficient set S.

A ***zeroth-degree*** polynomial is called a **constant polynomial** and is simply an element of the set of coefficients. An ***nth-degree*** polynomial is said to be a **monic polynomial** if $a_n = 1$.

Polynomial arithmetic includes the operations of **addition**, **subtraction**, and **multiplication**. These operations are defined in a natural way as though the variable x was an element of S. Division is similarly defined, but requires that S be a ***field***. Examples of fields include the real numbers, rational numbers, and Z_p for p prime. Note that the set of all integers is not a field and does not support polynomial division.

Addition and subtraction are performed by adding or subtracting corresponding coefficients. Thus, if

$$f(x) = \sum_{i=0}^{n} a_i x^i; \quad g(x) = \sum_{i=0}^{m} b_i x^i; \quad n \geq m$$

Then addition is defined as

$$f(x) + g(x) = \sum_{i=0}^{m} (a_i + b_i) x^i + \sum_{i=m+1}^{n} a_i x^i$$

And multiplication is defined as

$$f(x) \times g(x) = \sum_{i=0}^{n+m} c_i x^i$$

Where

$c_k = a_0 b_{k1} + a_1 b_{k1} + \ldots + a_{k1} b_1 + a_k b_0$

In the last formula, we treat a_i as zero for i > n and b_i as zero for i > m. Note that the degree of the product is equal to the sum of the degrees of the two polynomials.

Example: let **$f(x) = x^3 + x^2 + 2$** and **$g(x) = x^2\ x + 1$**, where S is the set of integers. Then

$F(x) + g(x) = x^3 + 2x^2\ x + 3$

Example:

$$\begin{array}{r} x^3 + x^2 \quad + 2 \\ +\ (x^2 - x + 1) \\ \hline x^3 + 2x^2 - x + 3 \end{array}$$

(a) Addition

$$\begin{array}{r} x^3 + x^2 \quad + 2 \\ -\ (x^2 - x + 1) \\ \hline x^3 \quad + x + 1 \end{array}$$

(b) Subtraction

$$\begin{array}{r} x^3 + x^2 \quad + 2 \\ \times\ (x^2 - x + 1) \\ \hline x^3 + x^2 \quad + 2 \\ -x^4 - x^3 \quad - 2x \\ x^5 + x^4 \quad + 2x^2 \\ \hline x^5 \quad + 3x^2 - 2x + 2 \end{array}$$

(c) Multiplication

$$\begin{array}{r|l} & x + 2 \\ \hline x^2 - x + 1 & x^3 + x^2 \quad + 2 \\ & x^3 - x^2 + x \\ \hline & 2x^2 - x + 2 \\ & 2x^2 - 2x + 2 \\ \hline & x \end{array}$$

(d) Division

3.3 Primitive polynomial

In field theory, a branch of mathematics, a primitive polynomial is the minimal polynomial of a primitive element of the finite extension field GF(p^m). In other words, a polynomial F(X) with coefficients in GF(p) = Z/p^Z is a primitive polynomial if it has a root a in GF(p^m) such that $\{0, 1, \alpha, \alpha^2, \alpha^3, \ldots, \alpha^{p^m-2}\}$ is the entire field GF(p^m), and moreover, F(X) is the smallest degree polynomial having a as root.

3.3.1 Properties

Because all minimal polynomials are irreducible, all primitive polynomials are also irreducible. A primitive polynomial must have a non-zero constant term, for otherwise it will be divisible by x. Over the field of two elements, x+1 is a primitive polynomial and all other primitive polynomials have an odd number of terms, since any polynomial mod 2 with an even number of terms is divisible by x+1.

An irreducible polynomial of degree m, F(x) over GF(p) for prime p, is a primitive polynomial if the smallest positive integer n such that F(x) divides x^n - 1 is n = p^m − 1.Over GF(p^m) there are exactly φ(p^m − 1)/m primitive

polynomials of degree m, where φ is Euler's totient function.The roots of a primitive polynomial all have order $p^m - 1$.

- Any primitive has period (2^n-1)
- $\varphi(n) = \varphi(2^n-1)/n$ $\varphi(m)/n$=the number of permeation of equation

φ (m):-is the number of positive integers which are less than or equal to M but coprime to it

- To compute the φ (M) Euler's by

1. If m is prime the Euler's is m-1
2. If m product to large prime m=p*q then (p-1)*(q-1)
3. By compute the numbers of equal or less than of m

Example: order

N	φ $(2^n-1)/n$		No. permeation	
1	φ $(2^1-1)/1$	φ (1)/1	1/1	1
2	φ $(2^2-1)/2$	φ (3)/2	2/2	1
3	φ $(2^3-1)/3$	φ (7)/3	6/3	2
4	φ $(2^4-1)/4$	φ (15)/4	(φ (5)* φ (3))/4	2

Example: primitive

Let $f(x)=X^2+x+1$ the order of f(x) is

$2^2-1=2^2-1=3$

Then

$(x^2+x+1)(x+1)=x^3+1$ $(1+x/x^2+x+1)$ mod 2

i.e.

$x^2 + x + 1 / x^3 + 1 = x + 1$

3.3.3 Pseudo-Random bit generation

Primitive polynomials define a recurrence relation that can be used to generate pseudorandom bits. In fact every linear feedback shift register with maximum cycle (that is 2^{lfsr} length - 1) is related with primitive polynomial.

For example, given the primitive polynomial $x^{10} + x^3 + 1$, we start with a user-specified bit seed (it need not randomly be chosen, but it can be). We then take the 10^{th}, 3^{rd}, and 0^{th} bits of it, starting from the least significant bit, and Xor them together, obtaining a new bit. The seed is then shifted left and the new bit is made the least significant bit of the seed. This process can be repeated to generate $2^{10}-1 = 1023$ pseudo-random bits.

In general, for a primitive polynomial of degree m, this process will generate 2^m-1 pseudo-random bits before repeating the same sequence

3.4 Irreducible Polynomial

(**Mathematics**) A polynomial is irreducible over a field **K** if **it cannot be written as the product of two polynomials** of lesser degree whose

In mathematics, the adjective irreducible means that an object cannot be expressed as the product of two or more non-trivial factors in a given set. See also factorization.

For any field F, the ring of polynomials with coefficients in F is denoted by F[x]. A polynomial p(x) in F[x] is called irreducible over F if it is non-constant and cannot be represented as the product of two or more non-constant polynomials from F[x]. The property of irreducibility depends on the field F; a polynomial may be irreducible over some fields but reducible over others. Some simple examples are discussed below.

Galois theory studies the relationship between a field, its Galois group, and its irreducible polynomials in depth. Interesting and non-trivial applications can be found in the study of finite fields.

It is helpful to compare **irreducible polynomials** to prime numbers: prime numbers (together with the corresponding negative numbers of equal modulus) are the irreducible integers. They exhibit many of the general properties of the concept of 'irreducibility' that equally apply to irreducible polynomials, such as the essentially unique factorization into prime or irreducible factors.

Every polynomial p(x) in F[x] can be factorized into polynomials that are irreducible over F. This factorization is unique up to permutation of the factors and the multiplication of the factors by constants from F (because the ring of polynomials over a field is a unique factorization domain).

Any polynomial over F must share either no roots or all roots with any given irreducible polynomial; this is Abel's irreducibility theorem.

- ***Irreducible polynomial*** *:- if f(x) and g(x) and h(x) is polynomial over GF(2)*

  h(x)=f(x).g(x) then f(x)/h(x) and g(x)/h(x) ***reducible***

  and

  if f(x) not divide by h(x) i.e. f(x)(not divide)h(x)

  f(x)/1 and f(x)/f(x) is called ***irreducible***

- ***Exponent polynomial*** *:-if f(x) over GF(2) and C_0=1 then polynomial is exponent* ***e***

$f(x)/x^e+1$

But

$f(x)/x^r+1$

When r in ***1<r<e***

And polynomial (characteristic) has exponent (x^e+1)

Simple examples

The following five polynomials demonstrate some elementary properties of reducible and irreducible polynomials:

$$p_1(x) = x^2 + 4x + 4 = (x+2)(x+2),$$
$$p_2(x) = x^2 - 4 = (x-2)(x+2),$$
$$p_3(x) = x^2 - 4/9 = (x-2/3)(x+2/3),$$
$$p_4(x) = x^2 - 2 = (x-\sqrt{2})(x+\sqrt{2}),$$
$$p_5(x) = x^2 + 1 = (x-i)(x+i).$$

Over the ring of integers, the first two polynomials are reducible, the last two are irreducible. (The third, of course, is not a polynomial over the integers.)

Over the field of rational numbers, the first three polynomials are reducible, but the other two polynomials are irreducible.

Over the field of real numbers, the first four polynomials are reducible, but p5(x) is still irreducible.

Over the field of complex numbers, all five polynomials are reducible. In fact, every nonzero polynomial p(x) over can be factored as:

$$p(x) = a(x - z_1) \cdots (x - z_n)$$

Lec-4-

Shift Register

Topics

5.1 Introduction SR

In digital circuits, a shift register is a cascade of flip flops, sharing the same clock, which has the output of anyone but the last flip-flop connected to the "data" input of the next one in the chain, resulting in a circuit that shifts by one position the **one-dimensional** "bit array" stored in it, shifting in the data present at its input and shifting out the last bit in the array, when enabled to do so by a transition of the clock input. More generally, a shift register may be **multidimensional**; such that its "data in"

input and stage outputs are themselves bit arrays: this is implemented simply by running several shift registers of the same bit-length in parallel.

Shift registers can have both **parallel** and **serial** inputs and outputs. These are often configured as serial-in, parallel-out (**SIPO**) or as parallel-in, serial-out (**PISO**). There are also types that have both serial and parallel input and types with serial and parallel output. There are also bi-directional shift registers which allow shifting in both directions: L→R or R→L. The serial input and last output of a shift register can also be connected together to create a circular shift register.

5.2 key stream generator

The basic element in **stream ciphers** is the key stream generators, which will generator the key stream (sequence) to be combining with plain text stream and produce the cipher text and often called *feedback shift register* (**FSR**).

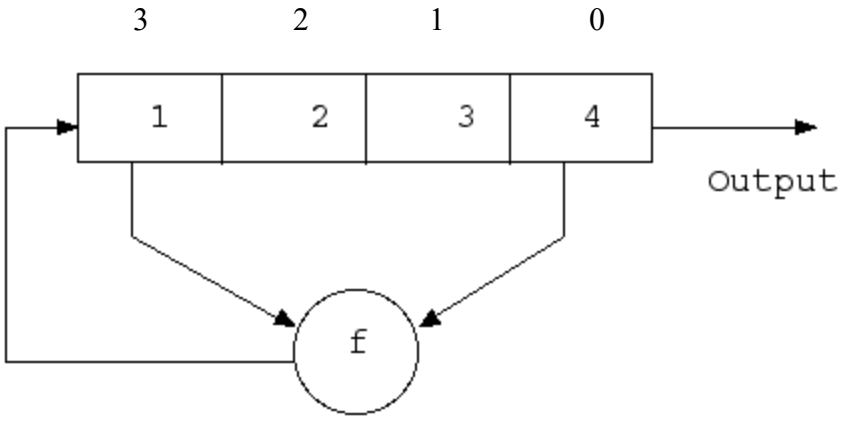
The shift register controlled by a block input. On every plus of the block the bits are shift one stage to right the bits that are generated at stage 0 from the output of the shift register .As the shift to the right, it is necessary to supply new bits to stage m-1, these bits can be obtained from feedback loop, according to a feedback function based on the values, which the feedback from the outputs of the ether stages.

For any m-stage register with feedback constants $c_0,c_1,c_2,\ldots..,c_{m-1}$,the characteristic polynomial f(x) is define

$F(x)=c_0x^0+c_1x^1+\ldots..+x^{m-1}$

$F(x)=1+ c_1x^1+.....+x^{m-1}$

The period of the sequence generator by the circit depend on whether polynomial E(x) is primitive and irreducible .maximal length sequence with period (2^m-1) are generator only if the case when the characteristic generating polynomial is primate and irreducible.



- A linear feedback shift register (**LFSR**) is a shift register whose input bit is a linear function of its previous state.

5.3 Feedback Function

In an LFSR, the bits contained in selected positions in the shift register are combined in some sort of function and the result is feed back into the register's input bit. By definition, the selected bit values are collected before the register is clocked and the result of the feedback function is inserted into the shift register during the shift, filling the position that is emptied as a result of the shift.

The feedback function in an LFSR has several names: **XOR**, **odd parity, sum modulo 2**. Whatever the name, the function is simple: 1) Add the selected bit values, 2) If the sum is odd, the output of the function is one; otherwise the output is zero. In following Table shows the output for a 3 input XOR function.

Input A	Input B	Input C	XOR Output
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

5.4 TAP Sequence

An LFSR is one of a class of devices known as state machines. The contents of the register, the bits tapped for the feedback function, and the output of the feedback function together describe the state of the LFSR. With each shift, the LFSR moves to a new state. (**There is one exception to this -- when the contents of the register are all zeroes, the LFSR will never change state.**) For any given state, there can be only one succeeding state. The reverse is also true: any given state can have only one preceding state. For the rest of this discussion, only the contents of the register will be used to describe the state of the LFSR.

A state space of an **LFSR** is the list of all the states the LFSR can be in for a particular tap sequence and a particular starting value. Any tap sequence will yield at least two state spaces for an LFSR. (One of these spaces will be the one that contains only one state -- the all zero one.) Tap sequences that yield only two state spaces are referred to as maximal length tap sequences.

The state of an LFSR that is n bits long can be any one of 2^n different values. The largest state space possible for such an LFSR will be $\mathbf{2^n - 1}$ (all possible values minus the zero state). Because each state can have only once succeeding state, an LFSR with a maximal length tap sequence will pass through every non-zero state once and only once before repeating a state.

One corollary to this behavior is the output bit stream. The period of an LFSR is defined as the length of the stream before it repeats. The period, like the state space, is tied to the tap sequence and the starting value. As

a matter of fact, the period is equal to the size of the state space. The longest period possible corresponds to the largest possible state space, which is produced by a maximal length tap sequence. (Hence "maximal length")

In the table is a listing of the internal states and the output bit stream of a 4-bit LFSR with tap sequence [4, 1]. (This is the LFSR shown first figure.)

Register States				
Bit 1 (Tap)	Bit 2	Bit 3	Bit 4 (Tap)	Output Stream
1	1	0	1	
0	1	1	0	1
0	0	1	1	0
1	0	0	1	1
0	1	0	0	1
0	0	1	0	0
0	0	0	1	0
1	0	0	0	1
1	1	0	0	0
1	1	1	0	0
1	1	1	1	0
0	1	1	1	1
1	0	1	1	1
0	1	0	1	1
1	0	1	0	1
1	1	0	1	0

5.5 Maximal Length Tap Sequences

LFSR's can have multiple maximal length tap sequences. A maximal length tap sequence also describes the exponents in what is known as a primitive polynomial mod 2. For example, a **tap** sequence of 4, 1 describes

the primitive polynomial $x^4 + x^1 + 1$. Finding a primitive polynomial mod 2 of degree n (the largest exponent in the polynomial) will yield a maximal length tap sequence for an LFSR that is n bits long.

There is no quick way to determine if a tap sequence is maximal length. However, there are some ways to tell if one is not maximal length:

1) Maximal length tap sequences always have an ***even number of taps***.
2) The tap values in a maximal length tap sequence are all ***relatively prime***. A tap sequence like **12, 9, 6, 3** will not be maximal length because the tap values are all divisible by 3.

Discovering one maximal length tap sequence leads automatically to another. If a maximal length tap sequence is described by [**n, A, B, C**], another maximal length tap sequence will be described by [**n, n-C, n-B, n-A**]. Thus, if [**32, 3, 2, 1**] is a maximal length tap sequence, [**32, 31, 30, 29**] will also be a maximal length tap sequence. An interesting behavior of two such tap sequences is that the output bit streams are mirror images in time.

Some polynomials for maximal LFSRs

The following table lists maximal-length polynomials for shift-register lengths up to 18. Note that more than one maximal-length polynomial may exist for any given shift-register length.

Bits	Feedback polynomial	Period
n		$2^n - 1$
2	$x^2 + x + 1$	3
3	$x^3 + x^2 + 1$	7
4	$x^4 + x^3 + 1$	15
5	$x^5 + x^3 + 1$	31
6	$x^6 + x^5 + 1$	63
7	$x^7 + x^6 + 1$	127
8	$x^8 + x^6 + x^5 + x^4 + 1$	255
9	$x^9 + x^5 + 1$	511
10	$x^{10} + x^7 + 1$	1023
11	$x^{11} + x^9 + 1$	2047
12	$x^{12} + x^{11} + x^{10} + x^4 + 1$	4095
13	$x^{13} + x^{12} + x^{11} + x^8 + 1$	8191
14	$x^{14} + x^{13} + x^{12} + x^2 + 1$	16383
15	$x^{15} + x^{14} + 1$	32767
16	$x^{16} + x^{14} + x^{13} + x^{11} + 1$	65535
17	$x^{17} + x^{14} + 1$	131071
18	$x^{18} + x^{11} + 1$	262143

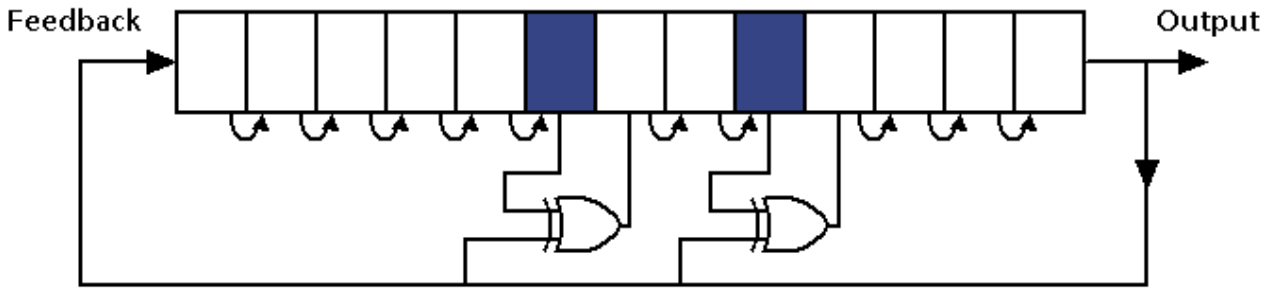
Example: Galois LFSRs

A Galois LFSR, or a LFSR in Galois configuration, is an alternate structure that can generate the same sequences as a conventional LFSR.

In Galois configuration, when the system is clocked, bits that are not taps are shifted as normal. The taps, on the other hand, are XOR'd with the new output, which also becomes the new input. To generate the same sequence, the order of the taps is the reverse of the order for the conventional LFSR.

Galois LFSRs do not concatenate every tap to produce the new input (the XOR'ing is done within the LFSR and no XOR's are run in serial, therefore the propagation times are reduced to that of one XOR rather than a whole chain), thus it is possible for each tap to be computed in parallel, increasing the speed of execution.

In a software implementation of an LFSR, the Galois form is more efficient **as the XOR operations can be implemented a word at a time: only the output bit must be examined individually.**



5.6 Linear Equivalence

Linear equivalence is define as the length of the smallest linear shift register which can be used to generate the same sequence .the primitive polynomial for the linear equivalence is called minimal polynomial .linear equivalence determines the complexity of the generate sequence .for the sequence with ***m*** linear equivalence ,it needs only ***2m*** of the generate

sequence to deduce the whole sequence .hence for good cipher secrecy ,it need to have a generator with large linear equivalence .BerleKamp-Massey algorithm can be used to determine the linear equivalence and the minimum polynomial as shown in the following algorithm:

- **Berlekamp-Massey algorithm**

INPUT: a binary sequence $s^n = s_0, s_1, s_2, \ldots, s_{n-1}$ of length n.
OUTPUT: the linear complexity $L(s^n)$ of s^n, $0 \leq L(s^n) \leq n$.

1. *Initialization.* $C(D) \leftarrow 1,\ L \leftarrow 0,\ m \leftarrow -1,\ B(D) \leftarrow 1,\ N \leftarrow 0$.
2. While $(N < n)$ do the following:
   2.1 *Compute the next discrepancy d.* $d \leftarrow (s_N + \sum_{i=1}^{L} c_i s_{N-i}) \bmod 2$.
   2.2 If $d = 1$ then do the following:
   $T(D) \leftarrow C(D),\ C(D) \leftarrow C(D) + B(D) \cdot D^{N-m}$.
   If $L \leq N/2$ then $L \leftarrow N + 1 - L,\ m \leftarrow N,\ B(D) \leftarrow T(D)$.
   2.3 $N \leftarrow N + 1$.
3. Return(L).

Example of the Massey Algorithm:

Let S^n=011 n=3 find the L (complexity)

Sol:-

S_N	d	T(D)	C(D)	L	m	B(D)	N
-	-	-	1	0	-1	1	0
0	0	-	1	0	-1	1	0
1	1	1	$1+D^2$	2	1	1	1
1	1	$1+D^2$	$1+D+D^2$	2	2	$1+D^2$	2
							3

L=2

C(D)= $1+D+D^2$

Lec-5-
Nonlinear Shift Register

Topics

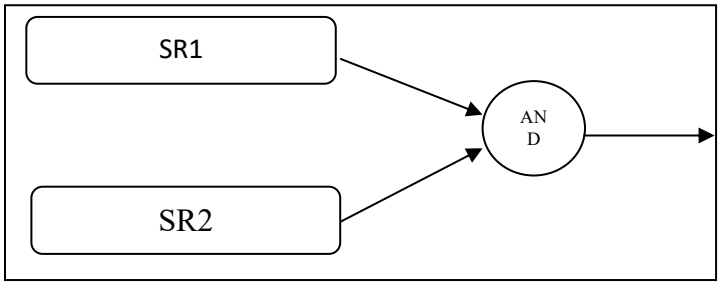
4.1 Nonlinear Shift Register
4.2 Nonlinear combination generators

6.1 Nonlinear Shfit Register

LFSR are not always useful ,since they are not resistence to a given know-plaintext attack.as an alternative ,nonlinear feedback shfit register with nonlinear feed back funtion are often used.

Linear feedback shfit register are unsafe because they have relative small linear complxity ,and hence a relatively small fragment of the key stream(LFSR sequnce) can be used to obtain the entire sequnce by solving as set of linear equation .to increase the linear complxity of LFSR ,one or more output sequnce of LFSR's are combined with some nonlinear function to produce relative higher linear complxity ,for example shfit register ***SR1*** genertates sequnce(***S1***) with sequnce length of (2^n-1),and shfit register ***SR2*** generates sequnce (***S2***) with sequnce length (2^m-1),then output sequnce (**S3**) will be :

S3=S1*S2 with period (sequnce length)= $(2^n-1)*(2^m-1)$



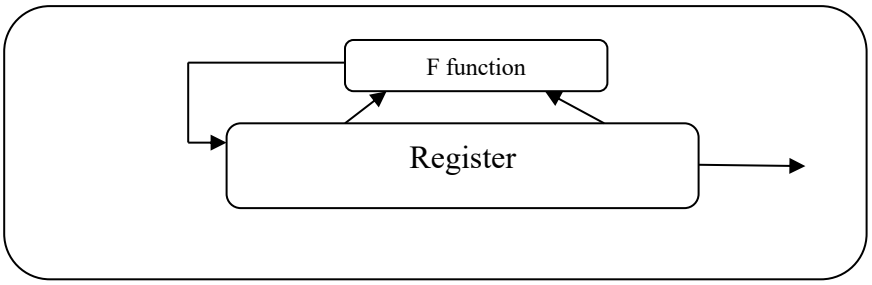
- **Nonlinear Feedback Shift Register**

  In which the key stream generator is a shift register with non-linear feedback function .as illustrated in following figure.

  In this type one LFSR is used with n-stages and non-linear feedback function. The simplest nonlinear function is **"AND"** functions, for example:

$F=1+X^1X^2+X^2X^3+X^2X^3X^4$

Where X1X2 are (**X^1 and X^2**)



- **Non Linearity Filtered LFSR System:**

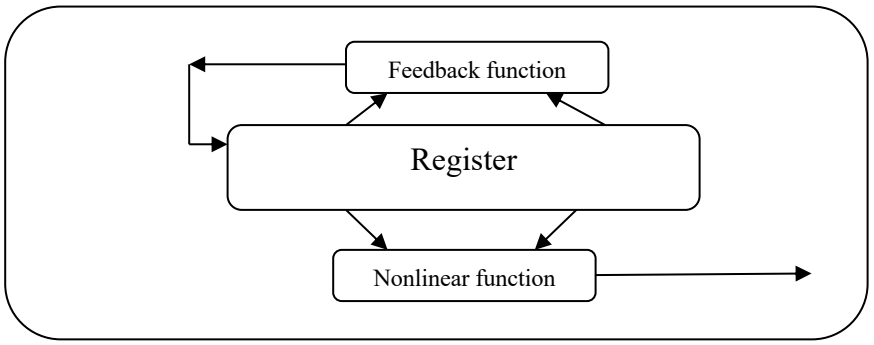
  In which a nonlinear logical function is applied to the contented of the LFSR. **Gorth** generator is an example of this type ,as illustrated in following figure **Gorth** sequence consists of:

  - Linear feedback function given by:

    F1=$S_1+S_2+S_3$

  - Non linear filter given by:

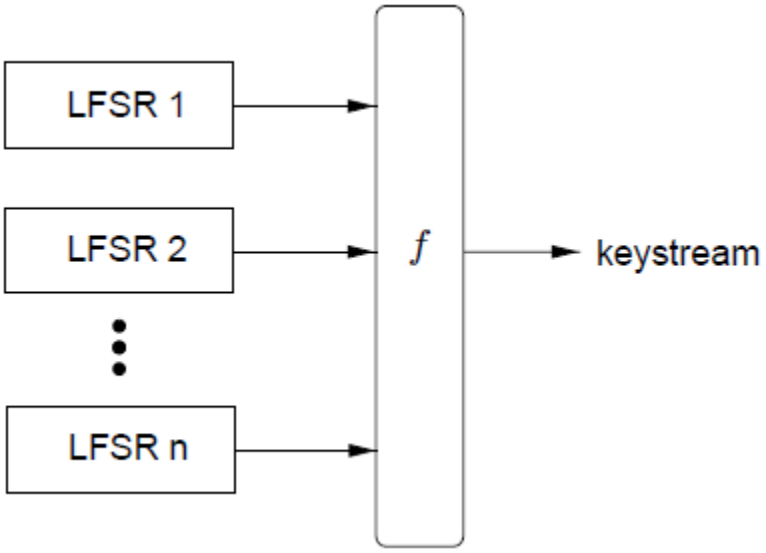
    F2=$S_0S3+S_1S_5+S_2S_4$



6.2 Nonlinear combination generators

One general technique for destroying the linearity inherent in LFSRs is to use several LFSRs in parallel. The key stream is generated as a nonlinear function **f** of the outputs of the component LFSRs; this construction is

illustrated in the following Figure. Such key stream generators are called ***nonlinear combination*** *generators*, and **f** is called the *combining function*. The remainder of this subsection demonstrates that the function **f** must satisfy several criteria in order to withstand certain particular cryptographic attacks.



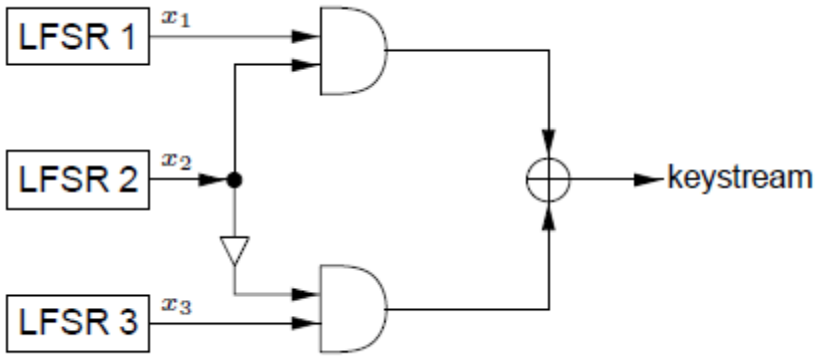
A product of m distinct variables is called an m^{th} order product of the variables. Every Boolean function f(x1, x2,……, xn) can be written as a modulo 2 sum of distinct m^{th} order products of its variables, 0 <= m <= n; this expression is called the algebraic normal form of ***f***. The nonlinear order of ***f*** is the maximum of the order of the terms appearing in its algebraic normal form.

For example, the Boolean function **f(x1, x2, x3, x4, x5)** = X1 **Xor** x2 **Xor** x3 **Xor** x4x5 **Xor** x1x3x4x5 has nonlinear order **4**. Note that the maximum possible nonlinear order of a Boolean function in n variables is n. demonstrates that the output sequence of a nonlinear combination generator has high linear complexity, provided that a combining function f of high nonlinear order is employed.

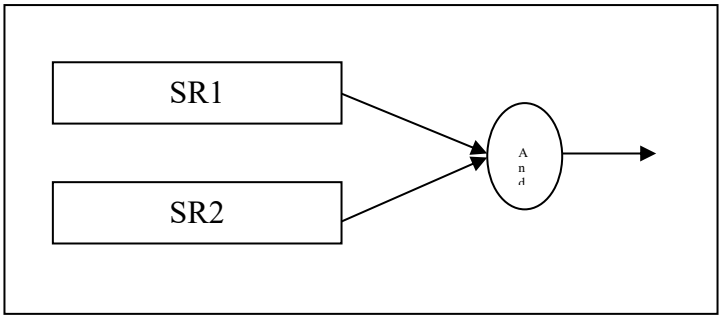
- (***Geffe generator***) The Geffe generator, as depicted in following Figure. is defined by three maximum-length LFSRs whose lengths ***L1, L2, L3*** are pairwise relatively prime, with nonlinear combining function

  $F(X_1, X_2, X_3)=(X_1$ and $X_2)$ Xor (not $(X_2)$and $X_3)$

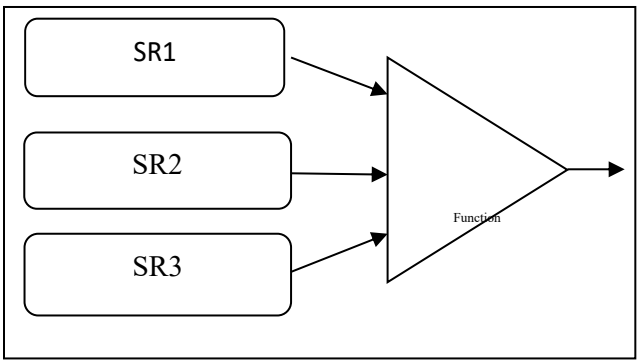
  And the maximal length of Geffe is
  Max period = $(2^{L1}-1)^* (2^{L2}-1)^* (2^{L3}-1)$



- **(Hardmard Generator)** :Nonlinear generator consists of two LFSR of(**X1,X2**) with nonlinear function
  F(x) = (S1 and S2)
  And the maximal
  Max period =$(2^{L1}-1)^*(2^{L2}-1)$



- **(Threshold Generator)**: nonlinear generator consist of three LFSR with nonlinear function
  F(x) = (S1 and S2) Xor (S1 and S3) Xor (S2 and S3)
  And the maximal
  Max period = $(2^{L1}-1)*(2^{L2}-1)*(2^{L3}-1)$



LFSR/FCSR Summation/Parity Cascade

The theory is that addition with carry destroys the algebraic properties of LFSRs, and that XOR destroys the algebraic properties of FCSRs. This generator combines those ideas, as used in the LFSR/FCSR Summation Generator and the LFSR/FCSR Parity Generator just listed, with the Gollmann cascade.

The generator is a series of arrays of registers, with the clock of each array controlled by the output of the previous array. In the following figure is one stage of this generator. The first array of LFSRs is clocked and the results are combined using addition with carry. If the output of this combining function is 1, then the next array (of FCSRs) is clocked and the output of those FCSRs is combined with the output of the previous combining function using XOR. If the output of the first combining function is 0, then the array of FCSRs is not clocked and the output is simply added to the

carry from the previous round. If the output of this second combining function is 1, then the third array of LFSRs is clocked, and so on.

This generator uses a lot of registers: *n*m*, where *n* is the number of stages and *m* is the number of registers per stage. recommend ***n* = 10 and *m* = 5.**

Example:

- ***Algorithm Alternating step generator***

SUMMARY: a control LFSR R1 is used to selectively step two other LFSRs, R2 and R3.

OUTPUT: a sequence which is the bitwise XOR of the output sequences of R2 and R3.

The following steps are repeated until a keystream of desired length is produced.

1. Register R1 is clocked.

2. If the output of R1 is 1 then:

- *R2 is clocked; R3 is not clocked but its previous output bit is repeated.*

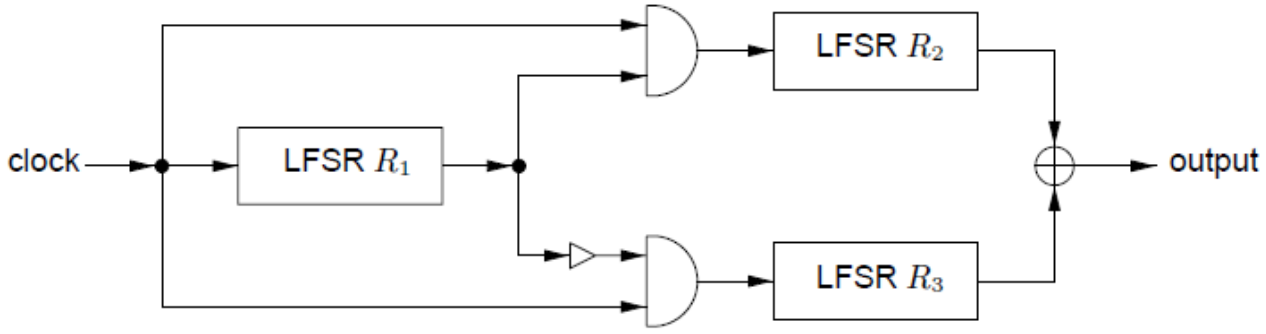
(For the first clock cycle, the "previous output bit" of R3 is taken to be 0.)

3. If the output of R1 is 0 then:

- *R3 is clocked; R2 is not clocked but its previous output bit is repeated.*

(For the first clock cycle, the "previous output bit" of R2 is taken to be 0.)

4. The output bits of R2 and R3 are XORed; the resulting bit is part of the key stream

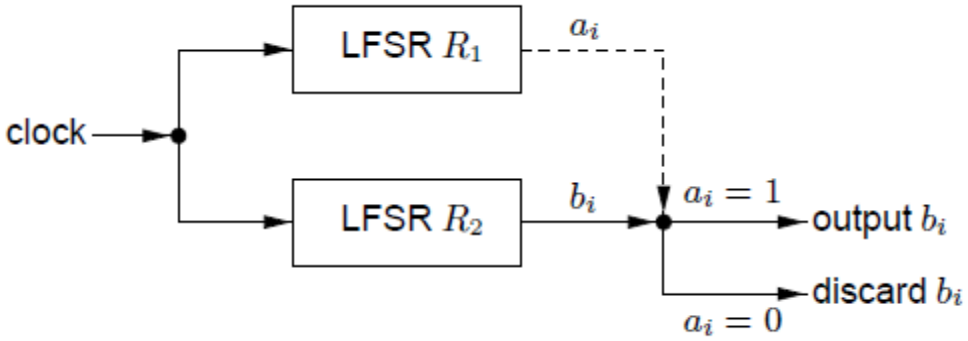


❖ ***Shrinking Generators***

SUMMARY: a control LFSR R1 is used to control the output of a second LFSR R2.

The following steps are repeated until a keystream of desired length is produced.

1. Registers R1 and R2 are clocked.
2. If the output of R1 is 1, the output bit of R2 forms part of the keystream.
3. If the output of R1 is 0, the output bit of R2 is discarded..



Lec-6-

Measure of Randomness

Topics

- Five Basic Tests

- **Definition:**

Run: sequence of identical bits (**0** or **1**).

EX. 01110000111

Runs are 0, 111, and 0000,111

Gap: run of zeroes. 1000011 contain of gap 0000

Block: run of ones. 1111001110 contain block 1111,111

- **Five Basic Tests**

Let s = s_0, s_1, s2, s_{n-1} be a binary sequence of length n. This subsection presents five statistical tests that are commonly used for determining whether the binary sequence **S** possesses some specific characteristics that a truly random sequence would be likely to exhibit. It is emphasized again that the outcome of each test is not definite, but rather probabilistic. If a sequence passes all five tests, there is no guarantee that it was indeed produced by a random bit generator.

(i) Frequency test (monobit test)

The purpose of this test is to determine whether the number of 0's and 1's in s are approximately the same, as would be expected for a random sequence. Let n0, n1 denote the number of 0's and 1's in s, respectively. The statistic used is

$$X_1 = (n0 - n1)^2 / n$$

Which approximately follows a χ^2 distribution with **1** degree of freedom if n ≥ 10.

(ii) Serial test (two-bit test)

The purpose of this test is to determine whether the number of occurrences of 00, 01, 10, and 11 as subsequences of s are approximately the same, as would be expected for a random sequence. Let n0, n1 denote the number of 0's and 1's in s, respectively, and let n00, n01, n10, n11 denote the number of occurrences of 00, 01, 10, 11 in s, respectively. Note that n00 + n01 + n10 + n11 = (n − 1) since the subsequences are allowed to overlap. The statistic used is

$$X_2 = \frac{4}{n-1}\left(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2\right) - \frac{2}{n}\left(n_0^2 + n_1^2\right) + 1$$

Which approximately follows a χ^2 distribution with **2** degrees of freedom if n ≥ 21.

Note: if block (000) is a (00) and (00)

(iii) Poker test

Let m be a positive integer such that [n /m] ≥ 5.(2^m), and let k = [n /m]. Divide the sequence **S** into k non-overlapping parts each of length m, and let n_i be the number of occurrences of the i^{th} type of sequence of length m, 1 ≤ i ≤ 2^m. The poker test determines whether the sequences of length m each appear approximately the same number of times in s, as would be expected for a random sequence. The statistic used is

$$X_3 = \frac{2^m}{k}\left(\sum_{i=1}^{2^m} n_i^2\right) - k$$

Which approximately follows a χ^2 distribution with **$2^m - 1$** degrees of freedom. Note that the poker test is a generalization of the frequency test: setting m = 1 in the poker test yields the frequency test.

Note: must divides the s to blocks length each is (m)

(iv)Runs test

The purpose of the runs test is to determine whether the number of runs (of either zeros or ones) of various lengths in the sequence s is as expected for a random sequence. The expected number of gaps (or blocks) of length i in a random sequence of length n is $e_i = (n-i+3)/2^{i+2}$. **Let k** be equal to the largest integer i for which **$e_i \geq 5$**. Let B_i, G_i be the number of **blocks and gaps**, respectively, of length i in s for each i, $1 \leq i \leq k$. The statistic used is

$$X_4 = \sum_{i=1}^{K} \frac{(b_i - e_i)^2}{e_i} + \sum_{i=1}^{K} \frac{(g_i - e_i)^2}{e_i}$$

Which approximately follows a χ^2 distribution with **2k − 2** degrees of freedom

Note: (K) depend on the higher (i)

(v) Autocorrelation test

The purpose of this test is to check for correlations between the sequence s and (non-cyclic) shifted versions of it. Let d be a fixed integer, $1 \leq d \leq [n/2]$. The number of bits in s not equal to their d-shifts is

A(d) = $\sum_{i=0}^{n-d-1}$ s_i+s_{i+d}, where + denotes the XOR operator. The statistic used is

$$X_5=2(A(d)-(n-d/2))/\sqrt{n-d}$$

Which approximately follows an N(0,1) distribution if n − d ≥ 10. Since small values of A(d) are as unexpected as large values of A(d), a two-sided test should be used.

Example :(*basic statistical tests*) Consider the (non-random) sequence s of length n =160 obtained by replicating the following sequence four times:

11100 01100 01000 10100 11101 11100 10010 01001

(i) (***Frequency test***) n0 = 84, n1 = 76 and the value of the statistic X_1 is 0.4.

(ii) (***Serial test***) n00 = 44, n01 = 40, n10 = 40, n11 = 35, and the value of the statistic X_2 is 0.6252.

(iii) (***poker test***) Here m = 3 and k = 53. The blocks 000, 001, 010, 011, 100, 101, 110,111 appear 5, 10, 6, 4, 12, 3, 6, and 7 times, respectively, and the value of the statistic X_3 is 9.6415.

(iv) (***runs test***) Here e1 = 20.25, e2 = 10.0625, e3 = 5, and k = 3. There are 25, 4, 5 **blocks** of lengths 1, 2, 3, respectively, and 8, 20, 12 **gaps** of lengths 1, 2, 3, respectively .The value of the statistic X_4 is 31.7913.

(v) (***autocorrelation test***) If d = 8, then A(8) = 100. The value of the statistic X5 is 3.8933.

For a significance level of α = 0.05, the threshold values for X1, X2, X3, X4, and X5 are 3.8415, 5.9915, 14.0671, 9.4877, and 1.96,. Hence, the given sequence **s** passes the **frequency**, **serial**, and **poker** tests, but fails the **runs** and **autocorrelation** tests.