USN | | | | | | | | | |

CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A++ GRADE BY NAAC

INTERNAL ASSESSMENT TEST – III

Sub:								Code:	18EC744
Date:	03/ 01 / 2024	Duration:	90 mins	Max Marks:	50	Sem:	VII	Branch:	ECE

Answer any 5 full questions

		Marks	CO	RBT
1	What is Diffie Hellman key exchange algorithm? Describe how the secret is computed by Alice and Bob to encrypt and decrypt the information.	[10]	CO2	L3
2	On the elliptic curve over the real number $y^2=x^3+x+6$, Let P=(2,7)and Q = (8,3).Find P+Q and 2P.	[10]	CO2	L3
3	Write a note on how ECC is used in encryption and decryption of the message.	[10]	CO2	L2
4.	Write an explanatory note on Linear Feedback shift registers.	[10]	CO5	L1
5.	Explain the following with necessary diagrams: a. Generalized Geffe Generator b. Threshold Generator	[10]	CO5	L2
6.	Explain Additive Generators. Also explain fish and pike Additive Generator.	[10]	CO5	L2
7.	Write a short note on Algorithm M and write the C code for it.	[10]	CO5	L2

IAT-III Scheme of solutions

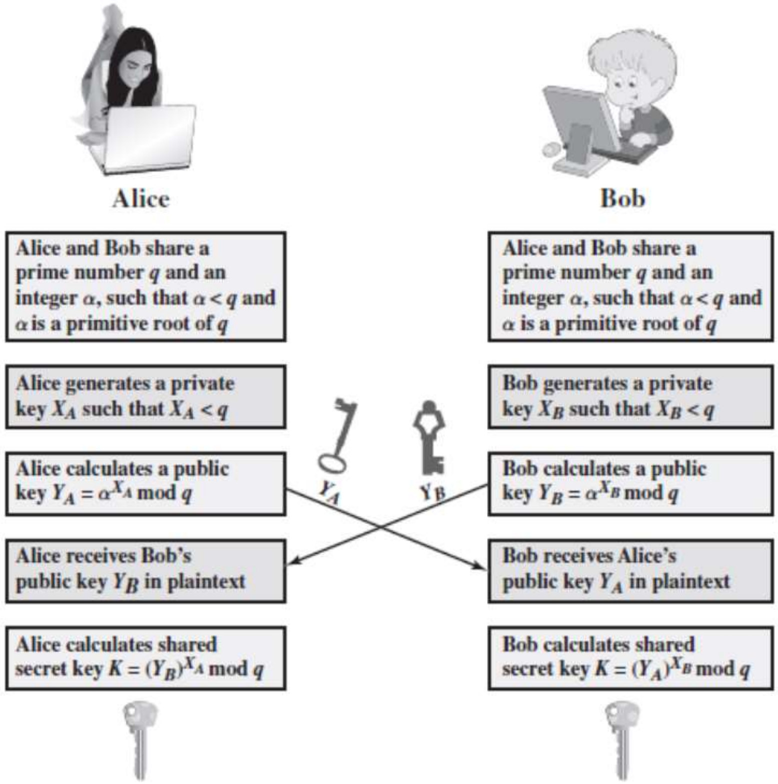
Q. no.	Questions	Marks

1. What is Diffie Hellman key exchange algorithm? Describe how the secret is computed by Alice and Bob to encrypt and decrypt the information. **10M**

Diffie Hellman Key Exchange Algorithm:

1. In this scheme, there are two publicly known numbers those are: a prime number q and an integer α that is a primitive root of q.
2. User A selects a random integer $X_A < q$ and compute $Y_A = \alpha^{X_A} \bmod q$.
3. User B selects a random integer $X_B < q$ and compute $Y_B = \alpha^{X_B} \bmod q$.
4. User A computes the key as $K_A = {Y_B}^{X_A} \bmod q$
5. User B computes the key as $K_B = {Y_A}^{X_B} \bmod q$

$K_A = {Y_B}^{X_A} \bmod q$
$K_A = (\alpha^{X_B} \bmod q)^{X_A} \bmod q$
$K_A = (\alpha^{X_B})^{X_A} \bmod q$
$K_A = \alpha^{X_B X_A} \bmod q$
$K_A = (\alpha^{X_A})^{X_B} \bmod q$
$K_A = (\alpha^{X_A} \bmod q)^{X_B} \bmod q$
$K_A = {Y_A}^{X_B} \bmod q$
$K_A = K_B$



The Diffie-Hellman Key Exchange

2. Let's examine the following elliptic curve $y^2 = x^3 + x + 6$ over $\mathbb{Z}_{11}$

P= (2,7) now to get 2P

$\lambda = \frac{3x_1^2 + 1}{2y_1}$ | $x_3 = \lambda^2 - x_1 - x_2$, $y_3 = \lambda(x_1 - x_3) - y_1$ | $\lambda = \frac{3\times2^2+1}{2\times7} = \left(\frac{13}{14}\right) mod 11 = \left(\frac{2}{3}\right) \text{mod } 11$

Multiplicative inverse of 3 mod 11 is 4 $\therefore \lambda = 2 \times 4$ mod 11 = 8

$\therefore x_3 = (64 - 2 - 2)$ mod 11 = 5

$\therefore y_3 = \{8(2 - 5) - 7\}$ mod 11 = -31 mod 11=2 $\quad\therefore$2P= (5,2) **5M**

To compute P+Q

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

P= (2,7) and Q = (8,3)

$\lambda = \left(\frac{3-7}{8-2}\right) mod 11 \equiv \left(\frac{-4}{6}\right) mod\ 11$, Multiplicative inverse of 6 mod 11 is 2

$\therefore \lambda = -8 mod 11 \equiv 3, x_3 = (9-2-8) mod\ 11 = -1\ mod\ 11 = 10$

$y_3 = (3(2-8)-7) mod\ 11 = -3\ mod\ 11 = 8$; ∴ P+Q = (10,8)

5M

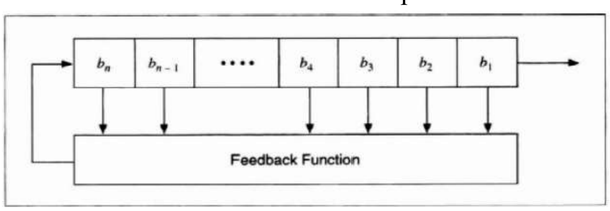
3.

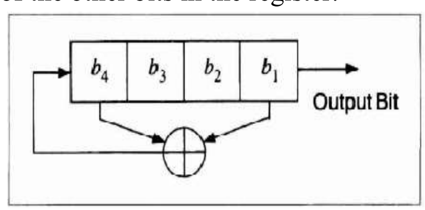
- Consider the equation $Q = kP$ where $Q, P \in E_P(a, b)$ and $k < p$.
- It is relatively easy to calculate Q given k and P, but it is hard to determine k given Q and P.
- This is called the discrete logarithm problem for elliptic curves.
- Consider a = (2,7), 9a = (10,9)
- If P= a and Q= 9a Because $9P = (10, 9) = Q$, the discrete logarithm $Q = (10, 9)$ to the base $P = (2, 7)$ is $k = 9$.
- In a real application, k would be so large as to make the brute force approach infeasible.
- The first task in ECC system is to encode the plaintext message m to be sent as an (x, y) point P_m.
- As with the key exchange system, an encryption/ decryption system requires a point G and an elliptic group $E_q(a, b)$ as parameters.
- Each user A selects a private key n_A and generates a public key $P_A = n_A * G$.
- To encrypt and send a message P_m to B, A chooses a random positive integer k and produces the ciphertext C_m
- $C_m = \{kG, P_m + kP_B\}$
- To decrypt the ciphertext, B multiplies the first point in the pair by B's private key and subtracts the result from the second point
- $P_m + kP_B - n_B(kG) = P_m + k(n_BG) - n_B(kG) = P_m$
- For an attacker to recover the message, the attacker would have to compute k given G and kG, which is assumed to be hard.

10M

4.

- Stream ciphers based on shift registers have been the workhorse of military cryptography.
- A **feedback shift register** is made up of two parts: a shift register and a **feedback function.**
- Each time a bit is needed, all of the bits in the shift register are shifted 1 bit to the right.
- The new left-most bit is computed as a function of the other bits in the register.





- Figure above shows a 4-bit LFSR tapped at the first and fourth bit. $b_4 = b_4 \oplus b_1$
- It is initialized with the value 1111**.**
- Cryptographers like to analyze sequences to convince themselves that they are random enough to be secure

1	1	1	1
0	1	1	1
1	0	1	1
0	1	0	1
1	0	1	0
1	1	0	1
0	1	1	0
0	0	1	1
1	0	0	1

- O/p sequence is 11110101100……
- An n-bit LFSR generate a 2^n–1bit long pseudo-random sequence
  - Only LFSRs with certain tap sequences will cycle through all 2^n-1 internal states; these are the maximal-period LFSRs.
  - The resulting output sequence is called an **m-sequence**
  - For a maximal-period LFSR, the polynomial formed from a tap sequence plus the constant 1 must be a primitive polynomial mod 2.
  - The **degree** of the polynomial is the length of the shift register.
  - For example, in the Table 16.2 listing of (32, 7, 5, 3, 2, 1, 0) means that the following polynomial is primitive modulo 2
  - $x^{32} + x^7 + x^5 + x^3 + x^2 + x + 1$
-

10M

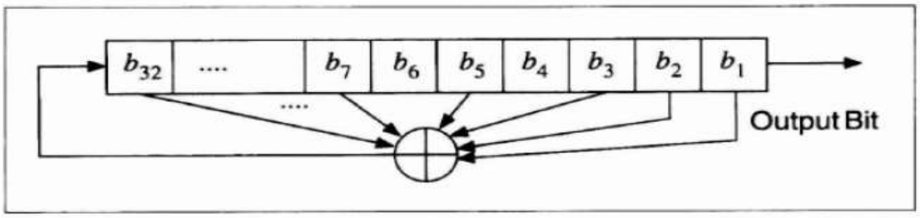


Figure 16.4 32-bit long maximal-length LFSR.

5. Explain the following with necessary diagrams: **10M**

a. Generalized Geffe Generator **6M**

b. Threshold Generator **4M**

a. **Geffe Generator:**

This generator uses three LFSRs, combined in a nonlinear manner. Two of the LFSRs are input a multiplexer and the third LFSR controls the output of the multiplexer. If $a1$, $a2$ and $a3$ are the output
of the three LFSRs, the output of the Geffe generator can be represented as: $b = (a1 \wedge a2) \oplus ((\neg a1 \wedge a3))$**.** If the LFSR have length $n1$, $n2$ and $n3$ respectively, then the linear complexity is: $(n1 + 1)n2 + n1n3$

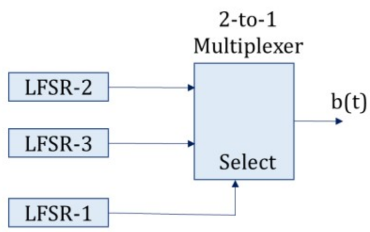


Figure: Geffe generator

Although this generator looks good on paper, it is cryptographically weak and falls to a correlation attack.

Generalized Geffe Generator:

Instead of choosing between two LFSRs, this scheme chooses between k LFSRs, where k is power of 2. There are $k + 1$ LFSRs total. LFSR-1 must be clocked log2 k times faster than the other k LFSRs. Though this scheme is complex than Gaffe generator, same kind of correlation attack is possible.

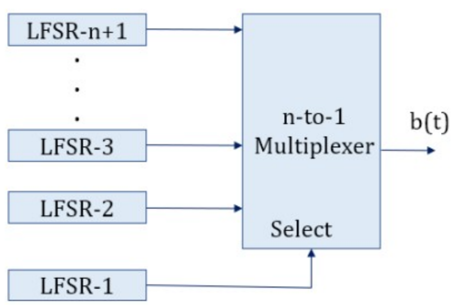
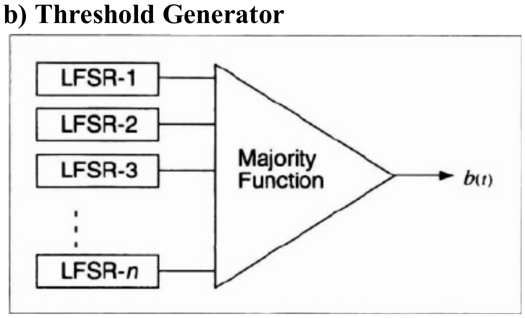


Figure: Generalized Geffe generator

b) Threshold Generator



This generator tries to get around the security problems of the previous generators by using a variable number of LFSRs . The theory is that if you use a lot of LFSRs, it's harder to break the cipher.

This generator is illustrated in Figure above. Take the output of a large number of LFSRs (use an odd number of them). Make sure the lengths of all the LFSRs are relatively prime and all the feedback polynomials are primitive: maximize the period. If more than half the output bits are 1, then the output of the generator is 1. If more than half the output bits are 0, then the output of the generator is 0. With three LFSRs, the output generator can be written as:

$$b = (a_1 \wedge a_2) \oplus (a_1 \wedge a_3) \oplus (a_2 \wedge a_3)$$

This is very similar to the Geffe generator, except that it has a larger linear complexity of

$$n_1n_2 + n_1n_3 + n_2n_3$$

where n_1, n_2, and n_3 are the lengths of the first, second, and third LFSRs.

This generator isn't great. Each output bit of the generator yields some information about the state of the LFSRs—0.189 bit to be exact—and the whole thing falls to a correlation attack. It's not recommended for use.

---

6. Explain Additive Generators. Also explain fish and pike Additive Generator. **10M**

ADDITIVE GENERATORS:

• *Additive generators* are extremely efficient because they produce random words instead of random bits. They are not secure on their own, but can be used as building blocks or secure generators.

• The initial state of the generator is an array of n-bit words: 8-bit words, 16-bit words, 32-bit words. The initial state is the key. The *i*th word of the generator is

$Xi = (Xi\text{-}a + Xi\text{-}b + Xi\text{-}c + \cdots + Xi\text{-}m) mod 2n$

• If the coefficients a, b, c, … m are chosen right, the period of this generator is at least $2n - 1$

Example: (55,24,0) is a primitive polynomial mod 2. This means that the following additive generator
is maximal length.

$Xi = (Xi\text{-}55 + Xi\text{-}24) od 2n$

This works because, primitive polynomial has three coefficients. If it has more coefficient, then we need some additional requirements to make it maximal length.

Fish:

➢ Fish is an additive generator based on techniques used in the shrinking generator. It produces a stream of 32-bit words which can be XORed with the plaintext stream to produce ciphertext, or XORed with ciphertext stream to produce plaintext.

➢ The algorithm is named as it is Fibonacci Shrinking generator.

➢ First, it uses two additive generators. The key is the initial values of these generators.

$Ai = (Ai\text{-}55 + Ai\text{-}24) od 232$

$Bi = (Bi\text{-}52 + Ai\text{-}19) mod 232$

➢ These sequences are shrunk, as a pair, depending on the least significant bit of *Bi*: if it is 1, use the pair; if it is 0, ignore the pair.

➢ *Cj* is the sequence of used words from *Ai* and *Dj* is the sequence of used words from *Bi*. These words are used in pairs- *C2j, C2j+1, D2j and D2j+1*- to generate two 32-bit output words: *K2j and K2j+1*.

$E2j = C2j \oplus (D2j \wedge D2j+1)$

$F2j = D2j+1 \wedge (E2j \wedge C2j+1)$

$K2j = E2j \oplus F2j$

$K2i = C2i+1 \oplus F2j$

➢ This algorithm is fast, Unfortunately, it is also insecure; an attack has a work factor of about *240.*

Pike:

➢ Pike is the leaner, meaner version of Fish, developed by Ross Anderson, the man who broke Fish.

➢ It uses three additive generators. For example:

$Ai = (Ai\text{-}55 + Ai\text{-}24) mod 232$

$Bi = (Bi\text{-}57 + Ai\text{-}7) mod 232$

$Ci = (Ci\text{-}58 + Ci\text{-}19) mod 232$

➢ To generate the keystream word, look at the additional carry bits.

➢ If all the three agree, then clock all three generators. If they don't, then just clock the two generators that agree. Save the carry bit for the next time. The final output is the XOR of the three generators.

➢ Pike is faster than Fish, as on average it requires 2.75 steps per output rather than 3 steps.

7.	The name is from Knuth [863]. It's a method for combining multiple pseudo-random streams that increases their security. One generator's output is used to select a delayed output from the other generator [996,1003]. In C:
	#define ARR_SIZE (8192) /* for example — the larger the better
	*/ static unsigned char delay[ARR_SIZE] ; unsigned char
	prngA(void) ; long prngB(void) ; void init_algM(void) {
	long i ; for (i = 0 ; i < ARR_SIZE ; i++) delay =
	prngA() ; } /* init_algM */ unsigned char algM(void) {
	long j,v ;
	j = prngB() % ARR_SIZE ; /* get the delay[] index */ v =
	delay[j] ; /* get the value to return */ delay[j] =
	prngA() ; /* replace it */ return (v) ; } /* algM
	*/
	This has strength in that if prngA were truly random, one could not learn anything about prngB (and could therefore not cryptanalyze it). If prngA were of the form that it could be cryptanalyzed only if its output were available in order (i.e., only if prngB were cryptanalyzed first) and otherwise it was effectively truly random, then the combination would be secure.
		10M

CCI HOD