USN | | | | | | | | | | |

CMRIT

Internal Assessment Test III – Jan. 2022

Sub:	**Cryptography**				Sub Code:	**18EC744**	Branch:	**EC**	
Date:	**27/01/2022**	Duration:	90 min's	Max Marks: **50**	Sem / Sec:	**7 A, B, C, D**			OBE

Answer any FIVE FULL Questions		MARKS	CO	RBT
1	Consider a Diffie Hellman scheme with a common prime $q = 19$ and primitive root $\alpha = 2$ a) Show that 2 is a primitive root of 19. b) If user A has public key $Y_A = 13$, what is $A's$ private key X_A? c) If user B has public key $Y_B = 15$, what is $B's$ private key X_B? d) Find the secret key K_A and K_B.	**[10]**	CO4	L3
2	Write the Diffie-Hellman key exchange algorithm and explain the Man-in-middle attack on Diffie-Hellman algorithm.	**[10]**	CO4	L2
3	Consider the elliptic curve defined over $E_{23}(1,1)$. Let $P = (3,13)$ and $Q = (9,16)$. Find $(P + Q)$ and $2P$.	**[10]**	CO4	L4
4	Write a short note on Linear feedback shift register (LFSRs). Explain the working of a 4-bit LFSR and show the output sequence if the seed state is 1111, justify it as the maximal length code.	**[10]**	CO5	L2
5	List out different types of LFSR-based Keystream generator. Discuss Geffe Generator and generalized gaffe generator in detail with necessary diagram.	**[10]**	CO5	L1
6	Write a short note on the following: a) A5 b) HUGHES XPD/KPD	**[10]**	CO5	L1
7	Write a short note on a) Algorithm M and write the C code for it. b) PKZIP	**[10]**	CO5	L2

----------All The Best---------

CMR
INSTITUTE OF
TECHNOLOGY

USN

Scheme and Solution of Internal Assesment Test – III

Sub:	**Cryptography**		Sec	7 A, B, C, D				Code:	**18EC744**
Date:	**27/01/2022**	Duration:	90 mins	Max Marks:	50	Sem:	VII	Branch:	ECE

Solution

1 *Consider a Diffie Hellman scheme with a common prime $q = 19$ and primitive root $\alpha = 2$*

a) *Show that 2 is a primitive root of 19.*

b) *If user A has public key $Y_A = 13$, what is A's private key X_A?* **[10 marks]**

c) *If user B has public key $Y_B = 15$, what is B's private key X_B?*

d) *Find the secret key K_A and K_B*

Ans $q = 19 \quad and\ \alpha = 2$

$\alpha = 2$ is the primitive root of 19

[2 marks]

$2^1\ mod\ 19 = 2$	$2^{11}\ mod\ 19 = 15$
$2^2\ mod\ 19 = 4$	$2^{12}\ mod\ 19 = 11$
$2^3\ mod\ 19 = 8$	$2^{13}\ mod\ 19 = 3$
$2^4\ mod\ 19 = 16$	$2^{14}\ mod\ 19 = 6$
$2^5\ mod\ 19 = 13$	$2^{15}\ mod\ 19 = 12$
$2^6\ mod\ 19 = 7$	$2^{16}\ mod\ 19 = 5$
$2^7\ mod\ 19 = 14$	$2^{17}\ mod\ 19 = 10$
$2^8\ mod\ 19 = 9$	$2^{18}\ mod\ 19 = 1$
$2^9\ mod\ 19 = 18$	
$2^{10}\ mod\ 19 = 17$	

+

$Y_A = \alpha^{X_A}\ mod\ q \Rightarrow\ 13 = 2^{X_A}\ mod\ 19 \Rightarrow\ X_A = 5$

$Y_B = \alpha^{X_B}\ mod\ q \Rightarrow\ 15 = 2^{X_B}\ mod\ 19 \Rightarrow\ X_B = 11$

[8 marks]

$K_A = {Y_B}^{X_A}\ mod\ q \Rightarrow\ K_A = 15^5\ mod\ 19 = 759375\ mod\ 19 = 2$

$K_B = {Y_A}^{X_B}\ mod\ q \Rightarrow\ K_B = 13^{11}\ mod\ 19 = 2$

$13^{11}\ mod\ 19$
$(11)_{10} = (1011)_2$
$1: 13\ mod\ 19 = 13$
$0: (13)^2\ mod\ 353 = 17$
$1: (17)^2 \times 13\ mod\ 353 = 14$
$1: (14)^2 \times 13\ mod\ 353 = 2$

$K_A = K_B = 2$

2 *Write the Diffie-Hellman key exchange algorithm and explain the Man-in-middle attack on Diffie-Hellman algorithm.* **[10 marks]**

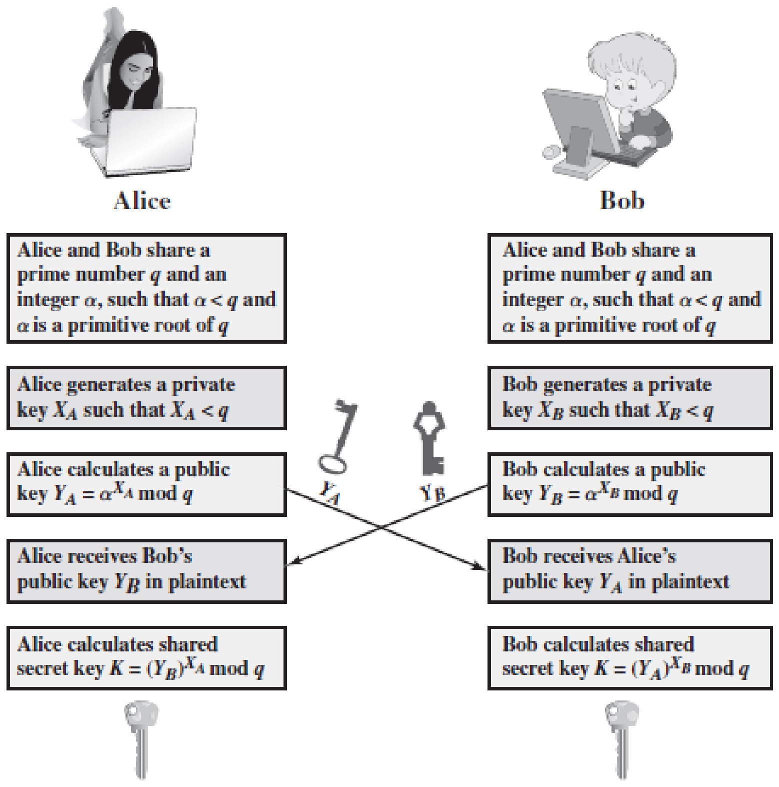
Ans In this scheme, there are two publicly known numbers those are: a prime number q and an integer α that is a primitive root of q.

User A selects a random integer $X_A < q$ and compute $Y_A = \alpha^{X_A}\ mod\ q$.

User B selects a random integer $X_B < q$ and compute $Y_B = \alpha^{X_B}\ mod\ q$. **[5 marks]**

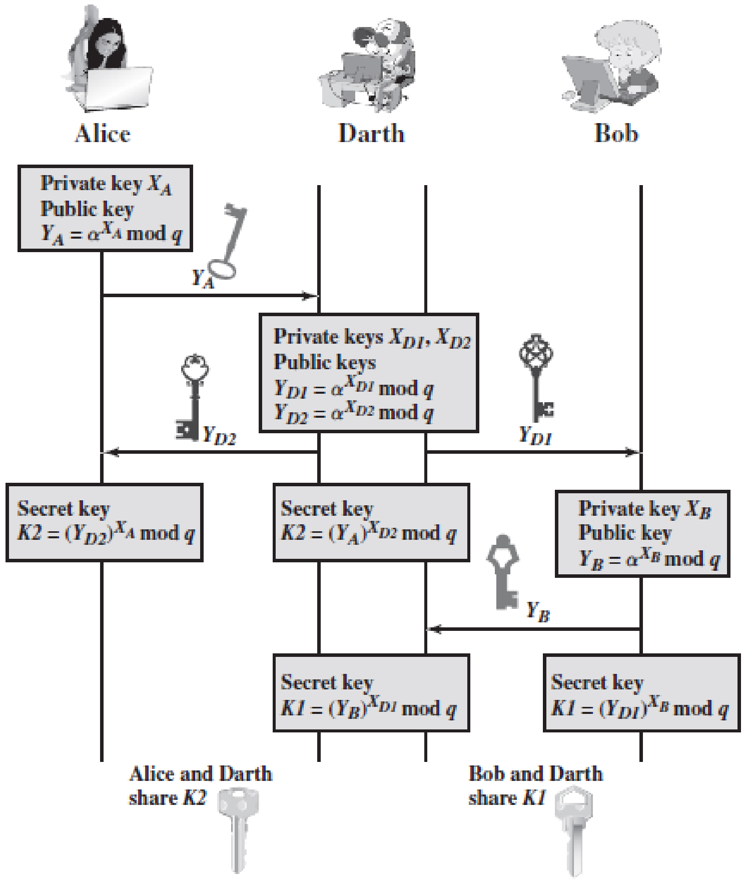
User A computes the key as $K_A = {Y_B}^{X_A}\ mod\ q$

User B computes the key as $K_B = {Y_A}^{X_B}\ mod\ q$


Alice
Bob
Alice and Bob share a prime number q and an integer α, such that $\alpha < q$ and α is a primitive root of q
Alice and Bob share a prime number q and an integer α, such that $\alpha < q$ and α is a primitive root of q
Alice generates a private key X_A such that $X_A < q$
Bob generates a private key X_B such that $X_B < q$
Alice calculates a public key $Y_A = \alpha^{X_A} \bmod q$
Bob calculates a public key $Y_B = \alpha^{X_B} \bmod q$
Y_A
Y_B
Alice receives Bob's public key Y_B in plaintext
Bob receives Alice's public key Y_A in plaintext
Alice calculates shared secret key $K = (Y_B)^{X_A} \bmod q$
Bob calculates shared secret key $K = (Y_A)^{X_B} \bmod q$

The Diffie-Hellman Key Exchange

<u>MAN-IN-MIDDLE ATTACK:</u>


Alice
Darth
Bob
Private key X_A Public key $Y_A = \alpha^{X_A} \bmod q$
Y_A
Private keys X_{D1}, X_{D2} Public keys $Y_{D1} = \alpha^{X_{D1}} \bmod q$ $Y_{D2} = \alpha^{X_{D2}} \bmod q$
Y_{D2}
Y_{D1}
Secret key $K2 = (Y_{D2})^{X_A} \bmod q$
Secret key $K2 = (Y_A)^{X_{D2}} \bmod q$
Private key X_B Public key $Y_B = \alpha^{X_B} \bmod q$
Y_B
Secret key $K1 = (Y_B)^{X_{D1}} \bmod q$
Secret key $K1 = (Y_{D1})^{X_B} \bmod q$
Alice and Darth share $K2$
Bob and Darth share $K1$

[5 marks]

Man-in-the-Middle Attack

a) Diffie Hellman Algorithm is insecure against man in middle attack.

b) <u>The attack proceeds as follows:</u>

(1) Darth prepare for the attack by generating 2 random key X_{D_1} *and* X_{D_2} and computes its corresponding private key Y_{D_1} *and* Y_{D_2}.
(2) Alice sends Y_A to Bob.
(3) Darth intercepts Y_A and transmits Y_{D_1}. Darth also calculate the $K_2 = (Y_A)^{X_{D_2}} \bmod q$
(4) Bob receives Y_{D_1} and calculate $K_1 = \left(Y_{D_1}\right)^{X_B} \bmod q$
(5) Bob transmits the Y_B to Alice.
(6) Darth intercepts Y_B and transmits Y_{D_2} to Alice and Darth calculate $K_1 = (Y_B)^{X_{D_1}} \bmod q$
(7) Alice receives Y_{D_2} and calculate $K_2 = \left(Y_{D_2}\right)^{X_A} \bmod q$

At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth shared secret key K_1 and Alice and Darth shared the secret key K_2. All the future communication between Bob and Alice is compromised.

3 *Consider the elliptic curve defined over* $E_{23}(1,1)$. *Let* $P = (3,13)$ *and* $Q = (9,16)$. *Find* $(P+Q)$ *and* $2P$. **[10 marks]**

Ans
$$\Delta = \left(\frac{y_Q - y_p}{x_Q - x_P}\right) \bmod p \Rightarrow \Delta = \left(\frac{16-13}{9-3}\right) \bmod 23 = \left(\frac{3}{6}\right) \bmod 23 = \left(\frac{1}{2}\right) \bmod 23 = 2^{-1} \bmod 23$$
$$= -11 \bmod 23 = 12$$

q	r_1	r_2	r	t_1	t_2	$t = t_1 - qt_2$
11	23	2	1	0	1	−11
2	2	1	0	1	− 11	23
	1	0		**−11**	23	

[5 marks]

$$x_R = \left(\Delta^2 - x_P - x_Q\right) \bmod p = (12^2 - 3 - 9) \bmod 23 = 132 \bmod 23 = 17$$
$$y_R = (\Delta(x_P - x_R) - y_P) \bmod p = (12(3-17) - 13) \bmod 23 = -181 \bmod 23 = -20 \bmod 23 = 3$$
$$P + Q = (17,3)$$

+

$$\Delta = \left(\frac{3{x_P}^2 + a}{2y_P}\right) \bmod p = \left(\frac{3(3^2)+1}{2 \times 13}\right) \bmod 23 = \left(\frac{28}{26}\right) \bmod 23 = \left(\frac{14}{13}\right) \bmod 23$$
$$= 14 \times 13^{-1} \bmod 23 = 14 \times (-7) \bmod 23 = -98 \bmod 23 = 17$$

[5 marks]

q	r_1	r_2	r	t_1	t_2	$t = t_1 - qt_2$
1	23	13	10	0	1	−1
1	13	10	3	1	− 1	2
3	10	3	1	−1	2	−7
3	3	1	0	2	− 7	23
	1	0		**−7**	23	

$$x_R = (\Delta^2 - 2x_P) \bmod p = (17^2 - 2 \times 3) \bmod 23 = 283 \bmod 23 = 7$$
$$y_R = (\Delta(x_P - x_R) - y_P) \bmod p = (17(3-7) - 13) \bmod 23 = (-81) \bmod 23 = -12 \bmod 23 = 11$$
$$2P = (7,11)$$

4 *Write a short note on Linear feedback shift register (LFSRs). Explain the working of a 4-bit LFSR and show the output sequence if the seed state is 1111, justify it as the maximal length code.* **[10 marks]**

Ans **LINEAR FEEDBACK SHIFT REGISTERS (LFSR):**

Shift register sequences are used in both cryptography and coding theory. E.g., stream ciphers based on shift registers have been used in military cryptography. A **feedback shift register** is made up of **two** parts: **[10 marks]**

- ➢ A shift registers
- ➢ Feedback function.

The shift register is a sequence of bits. (If it is n-bits long, it is called an n-bit shift register.). Each time a bit is needed, all the bits in the registers are shifted 1 bit to the right. The new left-most bit is computed as a function of the other bits. The output of the shift register is the 1 bit, often the least significant bit. The period of a shift register is the length of the output sequence before it starts repeating. Cryptographers have liked stream ciphers made up of shift registers.

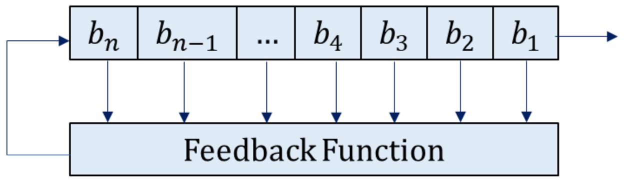


Figure: feedback shift register

- The simplest kind of feedback shift register is a **linear feedback shift register (LFSR)**. The feedback function is simply the XOR of certain bits in the register, the list of these bits is called **tap sequence** also called as **Fibonacci configuration**.

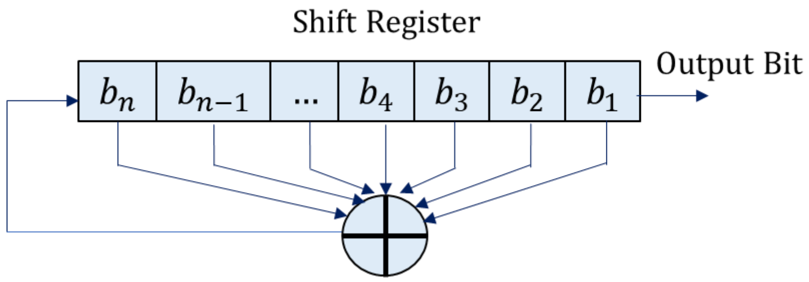


Figure: Linear feedback shift register

- The simple 4-bit LFSR can be shown in figure below. Here the first and the fourth bits are tapped. If it is initialized with the value 1111, it produces the following sequence of internal states before repeating.

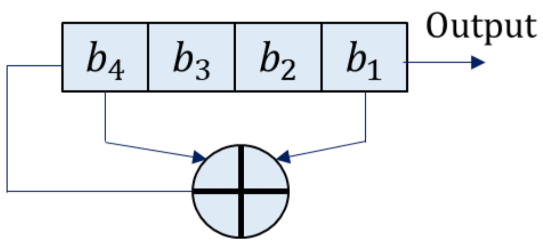


Figure: 4-bit LFSR

S.N.	Register state	Output
1	1111	1
2	0111	1
3	1011	1
4	0101	1
5	1010	0
6	1101	1
7	0110	0
8	0011	1
9	1001	1
10	0100	0
11	0010	0
12	0001	1
13	1000	0
14	1100	0
15	1110	0
16	1111	0

.

The output sequence is the string of least significant bits.: 1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 ......

- The shift registers filled with zeros will cause the LFSR to output a never-ending stream of zeros – this is particularly not useful.
- Only LFSR with certain tap sequences will cycle through all $2^n - 1$ internal states; these are maximal period LFSRs. The resulting output is called an **m-sequence**.
- For maximal period LFSR, the polynomial formed from a tap sequence plus the constant 1 must be primitive polynomial mod 2.
- A primitive polynomial of degree n is an irreducible polynomial if it divides $x^{2^{n-1}} + 1$ but not $x^d + 1$ for any d that divides $2^n - 1$.
- There is no easy way to generate the primitive polynomial mod 2 for a given degree. The easiest way is to choose a random polynomial and test whether it is primitive. But it is complicated as- something like testing random numbers for primality.
- E.g. the polynomial (32, 7, 5, 3, 2, 1, 0) means that the following polynomial is primitive

modulo 2. (Polynomial is: $x^{32} + x^7 + x^5 + x^3 + x^2 + x + 1$).

- It's easy to turn this into maximal-period LFSR. The first number is the length of the LFSR. The last number is always 0 and can be ignored. All the number except 0, specify the tap sequence. It means, if you take a 32-bit shift register and generate the new bit by XORing the thirty-second, seventh, fifth, third, second and first bits together, the resulting LFSR will be maximal length, it will cycle through $2^{32} - 1$ values before repeating. The C code for the LFSR is:

```
Int LFSR(){
     Static unsigned long ShiftRegister = 1;
     /* Anything but 0. */
     ShiftRegister = ((((ShiftRegister >> 31)
                       ^ (ShiftRegister >> 6)
                       ^ (ShiftRegister >> 4)
                       ^ (ShiftRegister >> 2)
                       ^ (ShiftRegister >> 1)
                       ^  ShiftRegister))
                       &   0×00000001)
                       <<  31)
                       | (ShiftRegister >> 1);
     Return ShiftRegister & 0×00000001;
}
```

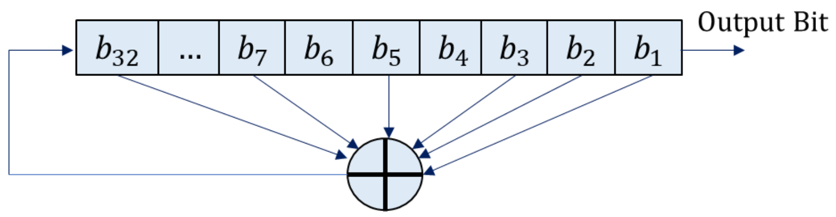


Figure: 32-bit long maximal-length LFSR

- This code is more complicated when the shift register is longer than the computer's word size.
- E.g. if (a, b, 0) is primitive, then (a, a-b, 0) is also primitive. If (a, b, c, d, 0) is primitive, then (a, a-d, a-c, a-b, 0) is also primitive. Mathematically
  If $x^a + x^b + 1$ is primitive, so is $x^a + x^{a-b} + 1$
  If $x^a + x^b + x^c + x^d + 1$ is primitive, so is $x^a + x^{a-d} + x^{a-c} + x^{a-b} + 1$
- Primitive polynomials are faster in software, because only two bits of the shift register have to be XORed to generate each new bit.
- The polynomials are generally **sparse**, means they have very few co-efficient. Sparseness is always a source of weakness, sometimes enough to break the algorithm. It is better to use **dense** primitive polynomial, those with a lot of coefficients for cryptographic applications. Generating dense primitive polynomials modulo 2 is not easy.

5 *List out different types of LFSR-based Keystream generator. Discuss Geffe Generator and generalized gaffe generator in detail with necessary diagram.* **[10 marks]**

Ans The list of LFSR based keystream generators are:

a) Geffe Generator
b) Generalized Geffe Generator
c) Jennings Generator
d) Beth-Piper Stop-and-Go Generator
e) Alternating Stop-and-Go Generator
f) Bilateral Stop-and-go Generator
g) Threshold Generator
h) Self-Decimated Generator
i) Multispeed Inner-Product Generator
j) Summation Generator
k) DNRSG (dynamic random-sequence generator)
l) Gollmann Cascade

[2 marks]
+

m) Shrinking Generator
n) Self-Shrinking Generator

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 a_1, a_2 *and* a_3 are the output of the three LFSRs, the output of the Geffe generator can be represented as: $b = (a_1 \wedge a_2) \oplus ((\neg a_1 \wedge a_3))$. If the LFSR have length n_1, n_2 *and* n_3 respectively, then the linear complexity is: $(n_1 + 1)n_2 + n_1 n_3$ **[4 marks]**

+

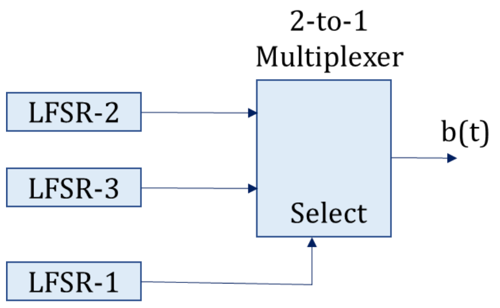


Figure: Geffe generator

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 $\log_2 k$ times faster than the other k LFSRs. Though this scheme is complex than Gaffe generator, same kind of correlation attack is possible. **[4 marks]**

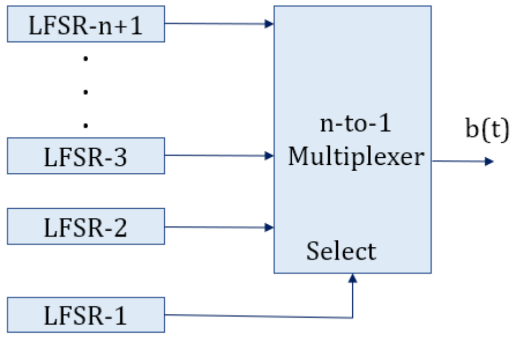


Figure: Generalized Geffe generator

6 *Write a short note on the following:* **[10 marks]**

a) A5
b) HUGHES XPD/KPD

Ans **A5:**

A5 is a stream cipher used to encrypt GSM (Group Special Mobile). That is the non-American standard for digital cellular mobile telephones. It is used to encrypt the link from the telephone to the base station. The rest of the link is unencrypted, the telephone company can easily eavesdrop on our conversation. Originally, it was designed to prohibit export of phones to some country. It is being discussed now, whether A5 might harm export sales. There is also a rumor as, various NATO intelligence agencies had a catfight in the mid-1980s over whether GSM encryption should be strong or weak. Germans wanted strong cryptography, as they were sitting near the Soviet Union. But other counties overruled them. A5 is a French design. A British telephone company gave all the documentation to Bradford University without a nondisclosure agreement. It leaked and was eventually posted to the internet. A5 consists of three LFSRs; the register lengths are 19,22 and 23, all the feedback polynomials are sparse. The output is the XOR of the three LFSRs. A5 uses variable clock control. Each register is clocked based on its own middle bit, XORed with the inverse threshold function of the middle bits of all three registers. Generally, two LFSRs clock in each round. There is a trivial attack requiring 2^{40} encryptions: Guess the contents of the first two LFSRs, then try to determine the third LFSR from the key stream. A5 is very efficient. It passes all known statistical tests, **[5 marks]**

+

its only weakness is, its registers are very short enough to make exhaustive search feasible. Variant of A5 with longer shift registers and denser feedback polynomial should be secure.

HUGHES XPD/KPD:

This algorithm is developed by Hughes Aircraft corp. They put it in army tactical radios and direction-finding equipment for sale to foreign militaries. It was designed in 1986 and called XPD, for **Exportable Protection Device**. Later renamed as KPD – **Kinetic Protection Device**. The algorithm uses a 61-bit LFSR. There are 2^{10} different primitive feedback polynomials, which were approved by the NSA. The key selects one of the polynomials as well as the initial state of the LFSR. It has eight different nonlinear filters, each of which has six taps from the LFSR and which produces 1 bit. The bits combine to generate a byte, which is used to encrypt or decrypt the data stream. This algorithm looks impressive. The NSA allows exports, so there must be some attack on the order of 2^{40} or less. **[5 marks]**

7 *Write a short note on* **[10 marks]**

a) Algorithm M and write the C code for it.

b) PKZIP

Ans **ALGORITHM M:**

The name is from Knuth. 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. This has strength in that if prngA were truly random, one could not learn anything about prngA. 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 is truly random, then the combination would be secure.

[5 marks]

C Program:

```
# define ARR_SIZE (8192) /* for example - 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 */
```

+

PKZIP:

It was designed by Roger Schlafly and built into the PKZIP data compression program. It is a stream cipher that encrypts data one byte at a time. This algorithm is available in version 2.04g. The algorithm uses three 32-bit variables, initialized as follows: **[5 marks]**

$$K_0 = 305419896$$
$$K_1 = 591751049$$
$$K_2 = 878082192$$

It has an 8-bit key, K_3, derived from K_2. Here is the algorithm:

$$C_i = P_i \wedge K_3$$
$$K_0 = crc32(K_0, P_i)$$
$$K_1 = K_1 + (K_0 \& 0x000000ff)$$

$$K_1 = K_1 * 134775813 + 1$$
$$K_2 = crc32(K_2, K_1 \gg 24)$$
$$K_3 = \left((K_2|2) * \left((K_2|2)^\wedge 1\right)\right) \gg 8$$
The function crc32 takes the previous value and a byte, XORs them, and calculates the next value by the CRC polynomial denoted by 0xedb88320. In practice, a 256-entry table can be precomputed and the crc32 calculation becomes:
$$crc32(a,b) = (a \gg 8)^\wedge table[(a\ \&0xff) \oplus b]$$
The table is precomputed by the original definition of crc32: $table[i] = crc32(i, 0)$
To encrypt a plaintext stream, first loop the key bytes through the encryption algorithm to update the keys. Ignore the cipher text in this step. Then encrypt the plaintext, one byte at a time. Twelve random bytes are prepended to the plaintext. Decryption is similar to encryption except that C_i is used in the second step of the algorithm instead of P_i.
Security of PKZIP: PKZIP attack requires 40 to 200 bytes of known plaintext and has a time complexity of about 2^{27}. It can be done in few hours on our personal computer. If the compressed file has any standard header, getting the known plaintext is no problem.