MASTER IN DATA SCIENCE

ASSIGNMENT 1

COURSE CODE : WQD7010

COURSE TITLE : NETWORK & SECURITY

NAME : NUR HIDAYAH BINTI AHMAD SHAFII

MATRIC NUMBER : 22120931

CLASS : 1

LECTURER : DR. SAAIDAL RAZALLI BIN AZZUHRI

1.

i) The publication type where the utmost priority is the confidentiality of financial reports in government organizations due to the sensitive nature of the data they contain. Financial reports often contain sensitive information such as earnings, expenses, investments, and other financial metrics. Ensuring the confidentiality of such data is crucial to prevent unauthorized access, manipulation, or leakage of sensitive information. Therefore, robust security measures and access controls are essential to safeguard the data.

ii) The scientific research paper is a publication type where data integrity is the primary concern. Any form of manipulation or inaccuracies could jeopardize the validity of research findings, potentially leading to misinformation, wasted resources, and harm to the reputation of researchers or institutions. Therefore, upholding data integrity throughout the publication process is crucial for maintaining trust in academic work.

iii) System availability for document production is crucial in legal settings like law firms and corporate legal departments. Lawyers, paralegals, and assistants heavily rely on desktop publishing systems to create legal documents. Interruptions or downtime can disrupt workflow, delay document preparation, and jeopardize critical deadlines with serious implications for legal proceedings or business deals. Therefore, ensuring the availability of the desktop publishing system is crucial for efficient document production and compliance with legal obligations in the legal profession.

2. i)

Let secret key = K = E0E0E0E0F1F1F1F1
The Binary of of secret key, K
K = 11100000 11100000 11100000 11100000 11110001 11110001 11110001 11110001
K+ = 11111111 11111111 11111111 11110000 00000000 00000000 00000000

Left denoted by C0 and right halve is denoted by D0
C0 = 1111111 1111111 1111111 1111111
D0 = 0000000 0000000 0000000 0000000

C1 = 1111111 1111111 1111111 1111111
D1 = 0000000 0000000 0000000 0000000

C2 = 1111111 1111111 1111111 1111111
D2 = 0000000 0000000 0000000 0000000

C3 = 1111111 1111111 1111111 1111111
D3 = 0000000 0000000 0000000 0000000

Since the pattern is the same,

C16 = 1111111 1111111 1111111 1111111
D16 = 0000000 0000000 0000000 0000000

Since it has predictive pattern and the results is the same after 16^{th} shift operation, it is considered as weak key. If we encrypt a block with a weak key and subsequently encrypt the result with the same weak key,we get the original block

ii)
From
K+ = 1111 1111 1111 1111 1111 1111 1111 0000 0000 0000 0000 0000 0000 0000
Let, left denoted by C0 and right halve is denoted by D0
C0 = 1111 1111 1111 1111 1111 1111 1111
D0 = 0000 0000 0000 0000 0000 0000 0000

C1 D1 = 1111 1111 1111 1111 1111 1111 1111 0000 0000 0000 0000 0000 0000 0000
Apply the permutation PC-2,
K1= 1111 1111 1111 1111 1111 1111 0000 0000 0000 0000 0000 0000

Since it consists either of all 0's,all 1's or half 0's and half 1's, it is a weak key. If we continue to the final round, performing rotations and permutations will produce the same result.

3. For M= MALAYSIA and K = ISSOGOOD

i) M = (4D 41 4C 41 59 53 49 41)hex
K = (49 53 53 4F 47 4F 4F 44)hex

ii)
M = (01001101 01000001 01001100 01000001 01011001 01010011 01001001 01000001)bin
K = (01001001 01010011 01010011 01001111 01000111 01001111 01001111 01000100)bin

iii)
Using DES,
For M+= 00000000 11111111 00000000 00110010 00000000 01010101 01010000
Left denoted by C0 and right halve is denoted by D0
C0 = 0000 0000 1111 1111 0000 0000 0011
D0 = 0010 0000 0000 0101 0101 0101 0000

Using initial permutation,
IP(M) = 11111111 00110000 00000101 11111011 00000000 00000000 01010101 00100000
L0 = 11111111 00110000 00000101 11111011
R0 = 00000000 00000000 01010101 00100000

iv)
IP(M) = 11111111 00110000 00000101 11111011 00000000 00000000 01010101 00100000

v)
M = (01001101 01000001 01001100 01000001 01011001 01010011 01001001 01000001)bin
Applying initial permutation to the block text M
IP(M) = 11111111 00110000 00000101 11111011 00000000 00000000 01010101 00100000

Divide the permuted block IP into a left half L0 of 32 bits, and a right half R0 of 32 bits,
L0 = 11111111 00110000 00000101 11111011
R0 = 00000000 00000000 01010101 00100000

Ln = Rn-1
Rn = Ln-1 + f(Rn-1,Kn)

For n = 1,
L1 = R0 = 00000000 00000000 01010101 00100000
R1 = L0 + f(R0, K1)

To calculate f, expand R0 to 48 bits using Expansion P-Box
E(R0) = 000000 000000 000000 000000 001010 101010 100100 000000

vi)
K = (01001001 01010011 01010011 01001111 01000111 01001111 01001111 01000100)bin
Using Key Permutation Table,
K+ = 0000000 0111111 1100000 0000000 0110111 0111110 0001101 0010110

vii)
K = 01001001 01010011 01010011 01001111 01000111 01001111 01001111 01000100

K+ = 0000000 0111111 1100000 0000000 0110111 0111110 0001101 0010110

C0 = 0000000 0111111 1100000 0000000
D0 = 0110111 0111110 0001101 0010110

C1 = 0000000 1111111 1000000 0000000
D1 = 1101110 1111100 0011010 0101101

C1D1 = 0000000 1111111 1000000 0000000 1101110 1111100 0011010 0101101

Apply Permutation Compression table to C1D1, hence

K1= 10100000 10010010 01000010 00010011 11110010 11100111

viii)
E(R0) = 00000000 00000000 00000000 00101010 10101001 00000000
K1 = 10100000 10010010 01000010 00010011 11110010 11100111

K1 + E(R0) = K1 XOR E(R0)
K1 XOR E(R0) = 10100000 10010010 01000010 00111001 01011011 11100111

ix)
Using S-Box, rearranging in 6-bits per group

K1 XOR E(R0) = 101000 001001 001001 000010 001110 010101 101111 100111

Applying S-Box,

S1 = 1101 1111 0011 1101 0110 1101 0111 0111

x)
From the output of the S-Box,
f = permutation of the S-Box
f= 1101 0110 1001 1110 1111 1100 1111 1110
From IP (M), L0 = 11111111 00110000 00000101 11111011

R1 = L0 + f(R0,K1)
R1 = L0 XOR f
R1 = 1111 1111 0011 0000 0000 0101 1111 1011 XOR 1101 0110 1001 1110 1111 1100 1111 1110
R1 = 0010 1001 1010 1110 1111 1001 0000 0101

4.

i) To modify the ciphertext C into a new ciphertext C1, the attacker can exploit the property of stream ciphers. These ciphers use the same keystream to XOR the plaintext and generate the ciphertext. With knowledge of the positions of bytes in the ciphertext representing the transfer amount, they can calculate the difference between the desired amount (MYR 500,000) and the original amount (MYR 500). By XORing this variance with specific bytes in the ciphertext, an attacker effectively increases or modifies the original transfer amount to match their desired value. This is possible because XORing a byte with a certain value and then performing another XOR operation using that same value cancels out its effect, effectively altering that byte to achieve its desired value. As a result, the attacker can modify the ciphertext by XORing the difference between these two amounts with the bytes corresponding to the original transfer amount, resulting in a new ciphertext C1. Upon decryption, this yields a message containing their desired transfer amount of MYR 500,000.

ii) Let Δ = the difference between the desired amount and the original amount
C[n] = n-th byte of ciphertext C,
M[n] = n-th byte of the original message M,
K[n] = n-th byte of the keystream K.

Then, the modified ciphertext C1 can be expressed as:
C1[n] =C[n] XOR (Δ[n] XOR M[n])

When the recipient decrypts C1 using the same keystream K, they obtain:
M1 [n]=C1[n] XOR K[n]

Substituting the expression for C1[n]:

M1[n] = (C[n] XOR (Δ[n] XOR M[n])) XOR K[n]
M1[n] = ((M[n] XOR K[n]) XOR (Δ[n] XOR M[n])) XOR K[n]
M1[n] = Δ[n] XOR M[n] XOR M[n]
M1[n] = Δ[n]

Therefore, the recipient decrypts C1 to obtain Δ, which represents the difference between the original and desired transfer amounts.

iii) Stream ciphers fail to protect message integrity as they rely solely on XORing plaintext with a keystream to generate ciphertext. Attackers can manipulate the ciphertext by XORing it with a calculated difference, allowing them to modify specific parts of the plaintext without knowing the entire keystream or original message. This lack of integrity protection makes stream ciphers vulnerable to targeted modifications. Without integrity checks, recipients decrypting modified ciphertext cannot verify if the message has been tampered with, highlighting the importance of using cryptographic methods that provide both confidentiality and integrity protection, such as authenticated encryption schemes or adding a message authentication code (MAC) alongside encryption.

5. S = [7 2 5 1 0 3 6 4]
K = [2 0 1]
PT = [5 3 1 6]

CT = PT XOR KeyStream
PT = CT XOR KeyStream
Keylen = 8
[K0 K1 K2] = [2 0 1]

First step:
/* Initialization */
For i = 0 to 8 do
S[i] = i;
T[i] = K[i mod keylen]
T[i] is the new K[i]
The K array thus is [K0 K1 K2] = [2 0 1]

Next:
/* Initial Permutation */
j = 0
for i = 0 to 7 do
    j = j + S[i] + K[i] mod 8
    swap S[i] and S[j]
end for K[0]= 2, K[1]= 0, K[2]= 1

For i = 0:
j = (0 + S[0] + K[0]) mod 8 = (0 + 7 + 2) mod 8 =1
Swap S[0] and S[1]: S = [2 7 5 1 0 3 6 4]

For i = 1:
j = (1 + S[1] + K[1]) mod 8 = (1 + 2 + 0) mod 8 = 3
Swap S[1] and S[3]: S = [2 1 5 7 0 3 6 4]

For i = 2:
j = (3 + 5 + 1) mod 8 = 7
Swap S[2] and S[7]: S = [2 1 4 7 0 3 6 5]

For i = 3:
j = (7 + 1 + 2) mod 8 = 2
Swap S[3] and S[2]: S = [2 1 7 4 0 3 6 5]

For i = 4:
j = (2 + 0 + 0) mod 8 = 2
Swap S[4] and S[2]: S = [2 1 0 4 7 3 6 5]

For i = 5:
j = (2 + 3 + 1) mod 8 = 6
Swap S[5] and S[6]: S = [2 1 0 4 7 6 3 5]

For i = 6:
j = (6 + 6 + 0) mod 8 = 4
Swap S[6] and S[4]: S = [2 1 0 4 3 6 7 5]

For i = 7:
j = (4 + 4 + 1) mod 8 = 1
Swap S[7] and S[1]: S = [7 1 0 4 3 6 2 5]

So, after the permutation, the array S becomes [7 1 0 4 3 6 2 5].

Next, a simplified stream generation:
set i and j back to 0
for i = i + 1 to length of PT
    j = j + S[i] mod 8
    swap S[i] and S[j]
    t = S[i] + S[j] mod 8
    KeyStream = S[t]
end for

For i = 1:
j = (0 + 1) mod 8 = 1
Swap S[1] and S[1]: S = [7 1 0 4 3 6 2 5]
t = (1 + 1) mod 8 = 2
KeyStream = S[2] = 0

For i = 2:
j = (1 + 0) mod 8 = 1
Swap S[2] and S[1]: S = [7 0 1 4 3 6 2 5]
t = (0 + 1) mod 8 = 1
KeyStream = S[1] = 0

For i = 3:
j = (1 + 1) mod 8 = 2
Swap S[3] and S[2]: S = [7 0 1 4 3 6 2 5]
t = (1 + 1) mod 8 = 2
KeyStream = S[2] = 1

For i = 4:
j = (2 + 4) mod 8 = 6
Swap S[4] and S[6]: S = [7 0 1 4 3 2 6 5]
t = (4 + 6) mod 8 = 2
KeyStream = S[2] = 1

So, the generated key stream is [0 0 1 1].
CT = PT XOR K = [5 3 1 6] XOR [0 0 1 1]
CT = [0101 0011 0001 0110] XOR [0000 0000 0001 0001]
    = [0101 0011 0000 0111] = 5307
Hence, the ciphertext is [5 3 0 7]

6. In CBC mode, each plaintext block relies on the previous ciphertext block for encryption. This prevents parallel execution of encryption operations for different message blocks in the same stream. Hence, encryption in CBC mode cannot be executed in parallel due to dependency on the previous ciphertext.

Decryption in CBC mode can be partially parallelized. While each block still depends on the previous ciphertext, once a ciphertext block is known, its decryption can proceed independently. However, full parallelism is limited due to the dependence on the previous ciphertext block.

7. In 8-bit CFB mode, a single-bit error in the ciphertext transmission affects decryption of that character. In CFB mode, decrypted ciphertext is XORed with plaintext to produce next block. Error in one block propagates to the next during decryption, limited to 8-bit block size since each block is processed independently.