****

**Selected Topics in Information Technology**

**Project-report**

**Students names:**

**Abdulaziz Moussa/202111203**

**Fahad Hashem Alhashimi/202110305**

**Hussein A H Abuhajjaj/202110404**

**Mohammed Abdulaziz Alsuwaidi/202111119**

**Abdel Rahman Ibrahim Qasem/202111287**

**The design of your algorithm**

Number of Rounder = 16

Key Size = 128 bits

Plain text size = 128 bits

Plain text : AE12FCD5AE12FCD5

1010 1110 0001 0010 1111 1100 1101 0101 1010 1110 0001 0010 1111 1100 1101 0101

Step 1 expansion

To expand these to 96 bit we will see the letter For Ex : A is 1010 so we should make it to 6 digits , what the letter after A is B , and B in binary is 1011 so we will take the first digit and we will add it to A so it will be 10101 then we will see the previous letter to A which is 9 and the binary is 1001 also we will take the first digit so the A 101011, so here we take the next and previous letter to A but in the next round we will go next 2 letter and previous 2 letter and so on.

After Expansion : 101011 111011 000101 001000 111101 110011 110111 010100 101011 111011 000101 001000 111101 110011 110111 010100

Step 2 permutation

We will have array of 96 bits then we will spilt them into 4 parts , then we will take first bit from each part and we will put it into the new first part , then second bit from each part and we will put it into the new second part and the same for the third and fouth , then we will repeat the same thing until that are full.

Part 1 : 101011 111011 000101 001000

Part 2 : 111101 110011 110111 010100

Part 3 : 101011 111011 000101 001000

Part 4 : 111101 110011 110111 010100

Part 1 After : 1111 1010 1010 0101 0101 1010

Part 2 After: 0101 1111 0000 0101 1111 0101

Part 3 After: 1111 1111 1111 0000 0000 0000

Part 4 After: 0101 1111 1111 1111 0101 0000

Then we will merge all the parts together

1111 1010 1010 0101 0101 1010 0101 1111 0000 0101 1111 0101 1111 1111 1111 0000 0000 0000 0101 1111 1111 1111 0101 0000

Step 3

XOR (Step 2 XOR Key Generation)

C: 1111 1010 1010 0101 0101 1010 0101 1111 0000 0101 1111 0101 1111 1111 1111 0000 0000 0000 0101 1111 1111 1111 0101 0000

K: 0101 1001 0101 0010 0011 0100 0011 1001 0101 0111 0001 0110 1010 0101 1010 0001 1100 1100 1111 0101 1110 1000 0110 1001

Result : 1010 0011 1111 0111 0110 1110 0110 0110 0101 0010 1110 0011 0101 1010 0101 0001 1100 1100 1010 1110 0111 0111 0010 1001

Step 4 sub

For example we will use this 110101, first we will take first 4 bits which is 1101 then we will convert it into hexadecimal , the result is D ,then we will take the last 2 bits and also convert it to hexadecimal it will be 1, so because the result here is 1 we will move the D one step front. So the D will be now E which is in binary 1110,

Next step we will convert the 1 that we got to binary and it will be 0001, the we will do XOR for E and 1

1110

0001

=1111 so he we got F , and we will name this a Ks1

Ks1 = F (1111)

Cipher text = E(1110)

To decrypt this , we XOR the Ks1 and E

1111

1110

=0001 which 1 then we will move E one step to the back so it will be D which is 1101 and we will add the 1 to as 01 , so it will be 110101

KEY Generation

Key :

10111111010011010010101110010111111000111100110111111000010111111110 1110 0111 1101 0111 1011 1100 0100 0101 1111 0010 1110 0001 1110 1110 1100

* Spilt the 128 bits into 2 part , delete 16 bit form each part (delete any bit that comes after the 0s for 16 time for each part, the 0s are known as pointer.) (We will do this only one time only for round 1).

Part 1

1011111101001101001010111001011111100011110011011111100001011111

After deleting :

101111100110010011010111110011101101111100101111

Part 2 :

1110 1110 0111 1101 0111 1011 1100 0100 0101 1111 0010 1110 0001 1110 1110 1110

After deleting :

1110 110 111 110 011 101 110 00 00 1111 010 110 01 1110 110 110

* We check every bit that comes before the pointer (YELLOW), will give it a name (shifter) , and we replace the shifter with the bit after the pointer. (every 4 bit will be together, and every time we move the shifter he will move only between them)

Then in every round we will shift the shifter one time to the right.

Part 1 : 1011 1110 0110 0100 1101 0111 1100 1110 1101 1111 0010 1111

Part 1 After Shifting : 0111 1101 0101 0010 1011 0111 0011 1101 1011 1111 0001 1111

Part 2 : 1110 1101 1111 0011 1011 1000 0011 1101 0110 0111 1011 0110

Part 2 After Shifting : 1101 1011 1111 0011 0111 0100 0011 1011 0101 0111 0111 0110

* We will take the first part and we will make AND operation with second part , then we will take the second part and we will do XOR operation with first part.

P1 AND P2

0111 1101 0101 0010 1011 0111 0011 1101 1011 1111 0001 1111 = P1

1101 1011 1111 0011 0111 0100 0011 1011 0101 0111 0111 0110 = P2

0101 1001 0101 0010 0011 0100 0011 1001 0101 0111 0001 0110 = P1 AND P2

P2 XOR P1

1101 1011 1111 0011 0111 0100 0011 1011 0101 0111 0111 0110 = P2

0111 1101 0101 0010 1011 0111 0011 1101 1011 1111 0001 1111 = P1

1010 0101 1010 0001 1100 1100 1111 0101 1110 1000 0110 1001 = P2 XOR P1

Then we bind them all

K1 = 0101 1001 0101 0010 0011 0100 0011 1001 0101 0111 0001 0110 1010 0101 1010 0001 1100 1100 1111 0101 1110 1000 0110 1001

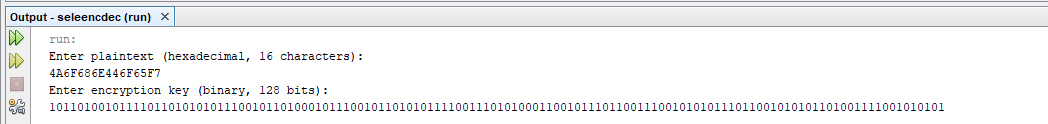
**Snapshots of plaintext encrypted decrypted using your algorithm**

Message= **JohnDoe** in Hexa = **4A6F686E446F65F7** followed by some filler characters.

Random key = 10110100101111011010101011100101101000101110010110101011110011101010001100101110110011100101010111011001010101101001111001010101

**In encryption:**

This is the input:



The output:



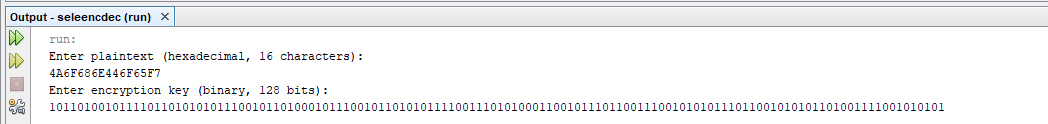
**In decryption:**

**A close-up of a computer code

Description automatically generated**

**Avalanche of your algorithm**

**First plaintext : 4A6F686E446F65F7**



**After encryption:**

A black text on a white background

Description automatically generated

**Second plaintext:** change one-bit **4A6F686E446F65F1A black and white image of a number

Description automatically generated**

**After encryption:**

**A black and white text

Description automatically generated**

|  |  |
| --- | --- |
| **First plaintext: 4A6F686E446F65F7** | **Second plaintext: 4A6F686E446F65F1** |
| **Result:**  **F83C4D35BF7E1453436EA3A4** | **Result :** **F83C4435BF7E1453436EA3B4** |

**Challenges in the project and how you faced them**

1. Algorithm Design Complexity It was quite challenging to develop a new and strong safeguarding symmetric encryption algorithm. These competing concerns of security, performance and creation of innovative solutions had to be met. Newer steps such as dynamic expansion and multi-step permutation also created even more complexity in the model formulation. Making the algorithm immune to such attacks was more challenging in this phase and was tying all the loose ends.

2. Implementation Difficulties The greatest challenge occurred from translating the designed algorithm in to code. The logic is entangled, binary operators and transforms must be manipulated accurately, which made implementation a slow affair. Substitution and XOR processes also contributed to the challenges that require debugging and guaranteeing correctness at all stages.

3. Testing and Validation To ensure the efficacy of the encryption and the decryption the process had to be tested fully. Slight mistakes such as incorrect calculations in the binary arithmetic or relating a wrong logical step to generate the output were complicated to rectify when occurred. As a result, the validation of the functionality and security attributes of the algorithm under different contexts was performed as an ongoing process throughout this work.