**DELHI TECHNOLOGICAL UNIVERSITY**



Digital Design-II (DD-II) EC204

**IMPLEMENTATION OF 32-BIT PASSWORD ENCRYPTION/DECRYPTION USING VHDL.**

Submitted by-

Ayussh Vashishth(2K19/EC/039)

Bhupinder Singh Saini(2K19/EC/043)

Submitted to-

Prof. Poornima Mittal

Department of Electronics and Communication Engineering

Delhi Technological University, Delhi

Delhi-110042

May, 2021.



Electronics & Communication Engg. Deptt.

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Shahbad Daulatpur, Bawana Road-Delhi-42

**Candidate’s Declaration**

We, hereby, declare that the work embodied in this project entitled “Implementation of 32-bit password encryption/decryption using VHDL.” Submitted to the Department of Electronics & Communication Engineering, Delhi Technological University, Delhi is an authentic record of our own Bonafide work and is correct to the best of our knowledge and belief. This work has been undertaken taking care of engineering ethics.

Name of the Students:

Ayussh Vashishth(2K19/EC/039)

Bhupinder Singh Saini (2K19/EC/043)

**ACKNOWLEDGEMENTS**

The success and final outcome of this project required a lot of guidance and assistance from many people, and we are extremely privileged to have got this all along the completion of my project. All that we have done is only due to such supervision and assistance, and we would not forget to thank them.

We respect and thank Prof. Poornima Mittal, for providing me an opportunity to do the project work and giving us all support and guidance, which made me complete the project duly. We are extremely thankful to her for providing us such a nice support and guidance.

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **Topic** | **Page No.** |
| Acknowledgements | 3 |
| Brief about VHDL | 5 |
| Brief on DES | 5 |
| Brief on DSSA | 5 |
| Detailed Project Description | 6 |
| Part 1(Proposal) | 6 |
| Part 2(Implementation of what was proposed) | 8 |
| Part 3(Our Final work) | 14 |
| Conclusion | 27 |
| Future Scope | 27 |
| References | 28 |

**Brief about VHDL-**



VHDL stands for very high-speed integrated circuit hardware description language. It is a programming language used to model a digital system by dataflow, behavioral and structural style of modeling. This language was first introduced in 1981 for the department of Defense (DoD) under the VHSIC program.

### Describing a Design

In VHDL an entity is used to describe a hardware module. An entity can be described using,

* Entity declaration
* Architecture
* Configuration
* Package declaration
* Package body

**Brief on DES(Data Encryption Standard) from which we took inspiration -**

The DES (Data Encryption Standard) algorithm is a symmetric-key block cipher created in the early 1970s by an IBM team and adopted by the National Institute of Standards and Technology (NIST). The algorithm takes the plain text in 64-bit blocks and converts them into ciphertext using 48-bit keys. Since it’s a symmetric-key algorithm, it employs the same key in both encrypting and decrypting the data.

**Brief on DSSA(Distributed System Security Architecture) -**

Distributed System Security Architecture or (DSSA) is a computer security architecture that provides a suite of functions including login, authentication, and access control in a distributed system.

**Detailed Project Description:**

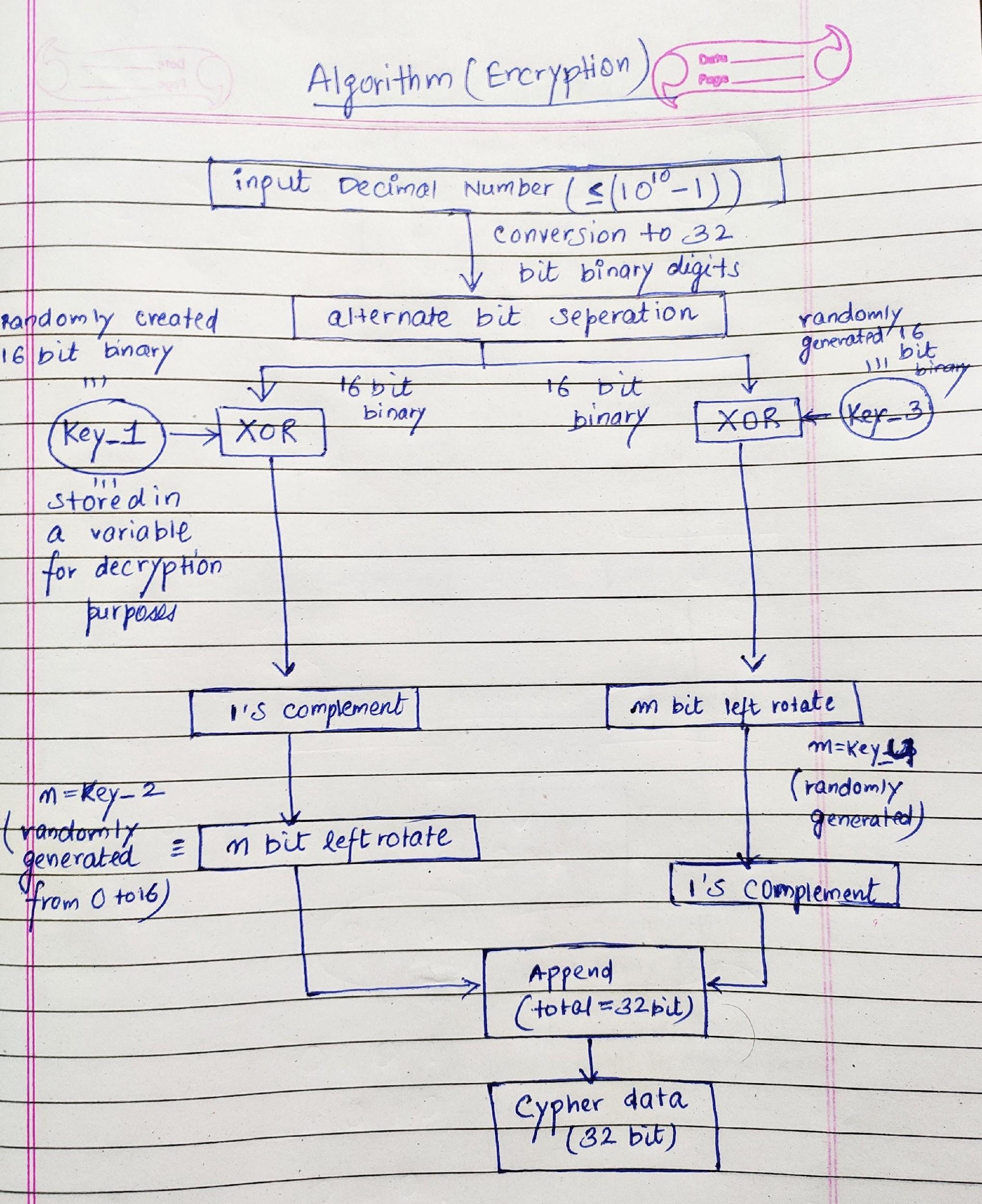
**PART – 1(Proposal)**

**Topic-** Implementation of 32-bit password encryption/decryption using VHDL.

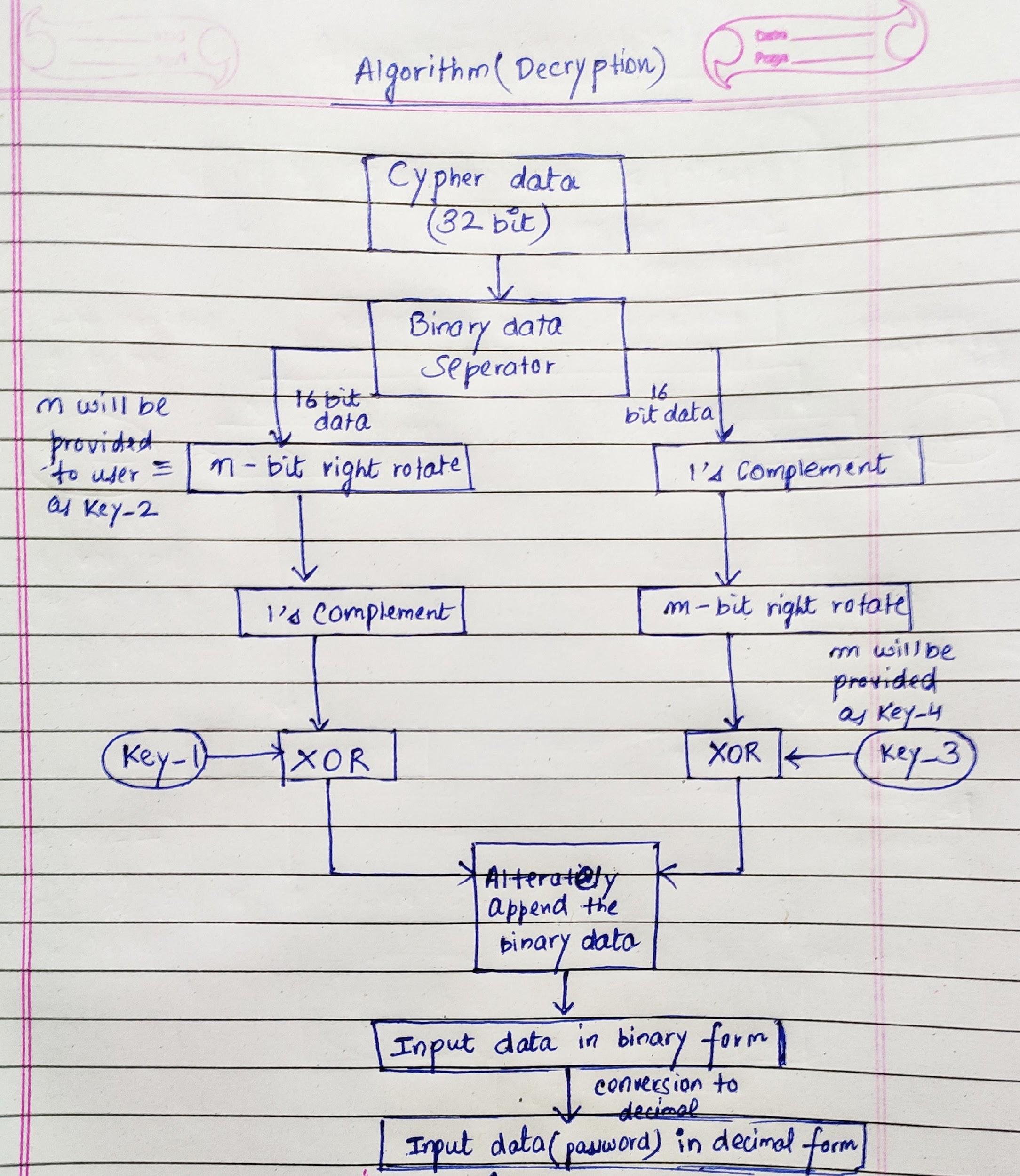
**Objective-** To develop a strong algorithm for highly secure password encryption and decryption (if required by the user).

**Abstract-** In this project we will develop a VHDL program which will provide a 4-step secure encryption to the password. For convenience of the user this password will be taken as a decimal value. Encryption will be achieved by performing a series of steps shown in the algorithm block diagram below. As can be observed from the above algorithm 4 random keys will also get generated automatically which will be saved in corresponding variables.

This algorithm will help in encryption of entered data -



This code will help in decryption of the encrypted data -



Now, let’s see our algorithm for decryption-Expected outcome- A specialty of the above algorithm is that even we as the developers could not judge the 4 keys which will be generated beforehand since they will only generate randomly at runtime. This possesses a great advantage as this makes our encrypted data more secure and less prone to cyber-attacks.

So, exactly what we are planning to do?

The plan is as explained above to encrypt a data (here password) and obtaining all the keys as output. Finally, these will be passed to our user (here my project partner) who will have the decryption code beforehand. And with the keys he can access the password even being remotely away and too with a good level of security. It should be noted that as we will progress in the project additions of more complex operations will be added to increase the security of the data provided.

**PART – 2(Implementation of what was proposed)**

We, designed an 8-bit encryption and decryption algorithm based on the one proposed earlier in our proposal.

**ENCRYPTION-**

We first designed cpp program to convert entered decimal number to n-bit binary (here for 8-bit).

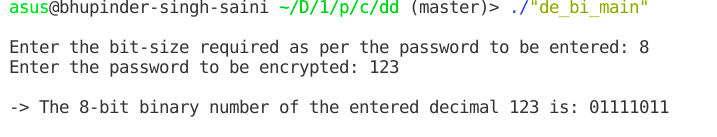
**Code-**

**Our cpp program -**

|  |
| --- |
| **#include <iostream> int \*decimalToBinary(int decimal, int bit\_size) {  *//dynamic array to store binary number*  int \*binaryNum = new int[bit\_size]();  *//dynamic array to store the binary number in reverse to binaryNum array*  int \*binary = new int[bit\_size + 1];  for (int i = 0; i < bit\_size + 1; i++)  binary[i] = -1;  *//counter for binary array*  int i = 0;  while (decimal > 0)  { *//storing remainder in binary array*  binaryNum[i] = decimal % 2;  decimal = decimal / 2;  i++;  }  i--;  *//copying data members of binaryNum to binary in reversed form wrt binaryNum*  for (int j = 0; i >= 0; j++)  binary[j] = binaryNum[i--];  i++;  delete[] binaryNum;  return binary; } int main() {  int decimal, bit\_size;  std::cout << std::endl;  std::cout << "Enter the bit-size required as per the password to be entered: ";  std::cin >> bit\_size;  std::cout << "Enter the password to be encrypted: ";  std::cin >> decimal;   *//final\_binary is a dynamic array with all elements initialized to 0*  int \*final\_binary = new int[bit\_size]();  *//the pointer binary point towards the binary array of decimaoToBinary function*  int \*binary = decimalToBinary(decimal, bit\_size);  *//loop for finding out the size of the binary number*  int size = 0, i = 0;  while (binary[i] != -1)  {  size++;  i++;  }  *//copying the data members of binary array as trailing members of final\_binary*  int k = 0;  for (int j = bit\_size - size; j < bit\_size; j++)  final\_binary[j] = binary[k++];  *//printing out the binary conversion of decimal in total bit size provided initially*  std::cout << std::endl;  std::cout << "-> The " << bit\_size << "-bit binary number of the entered decimal " << decimal << " is: ";  for (int j = 0; j < bit\_size; j++)  std::cout << final\_binary[j];  std::cout << std::endl;  std::cout << std::endl;  delete[] binary;  delete[] final\_binary; }** |

Since, we took 8-bit data encryption (hence from 0 to 255) therefore we will input a decimal (here we took 123) which will be converted into 8-bit binary by the program written.

**Terminal-**

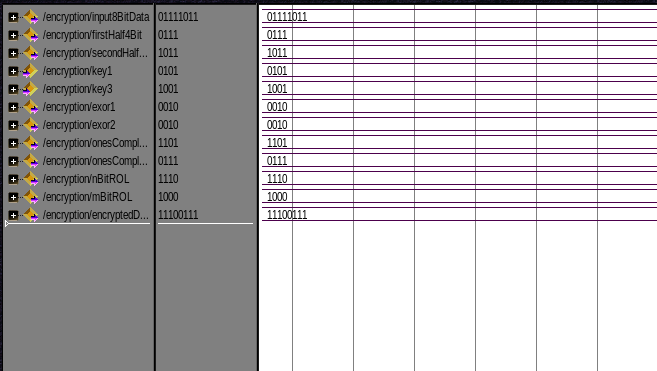


Now “01111011” will be the input as input8BitData which will be converted into cypher data.

**Our VHDL program CODE -**

|  |
| --- |
| **library ieee; use ieee.std\_logic\_1164.all; use ieee.std\_logic\_arith.all;  entity encryption is port(input8BitData: buffer bit\_vector(7 downto 0) := "01111011";  firstHalf4Bit: buffer bit\_vector(3 downto 0);  secondHalf4Bit: buffer bit\_vector(3 downto 0);  key1: in bit\_vector(3 downto 0) := "0101";  key3: in bit\_vector(3 downto 0) := "1001";  exor1: buffer bit\_vector(3 downto 0);  exor2: buffer bit\_vector(3 downto 0);  onesComplement1: buffer bit\_vector(3 downto 0);  onesComplement2: buffer bit\_vector(3 downto 0);  nBitROL: buffer bit\_vector(3 downto 0);  mBitROL: buffer bit\_vector(3 downto 0);  encryptedData: buffer bit\_vector(7 downto 0)); end encryption;   architecture behavior of encryption is begin  *--\*\*divding the 8-bit data into two 4-bit binary numbers*    *--first half 4-bit binary of input data*  process is  begin  for i in 0 to 3 loop  firstHalf4Bit(3 - i) <= input8BitData(7 - i);  end loop;  wait;  end process;    *--second half 4-bit binary of input data*  process is  begin  for i in 0 to 3 loop  secondHalf4Bit(i) <= input8BitData(i);  end loop;  wait;  end process;    *--\*\*first-half of the encryption algorithm*  *--xor of the first half of 4-bit binary and key 1*  exor1 <= firstHalf4Bit xor key1;    *--1's complement of the output exor*  onesComplement1 <= not(exor1);    *--let's take key 2(n) as 3, therefore we will do 3-bit left rotations*  nBitROL <= (onesComplement1) rol 3;    *--\*\*second-half of the encryption algorithm*    *--xor of the second-half of 4-bit binary and key 3*  exor2 <= secondHalf4Bit xor key3;    *--let's take key 3(m) as 2, therefore we will do 2-bit left rotations*  mBitROL <= (exor2) rol 2;    *--1's complement of the output mBitROL*  onesComplement2 <= not(mBitROL);    *--\*\*let's append the 4-bit data's we got from both the sides*  encryptedData <= nBitROL & onesComplement2;  *--the above 8-bit data we got is the cypher data of the the 8-bit password* end behavior;** |

**Simulation -**



**Result –** Here, we got the cypher data as “11100111”.

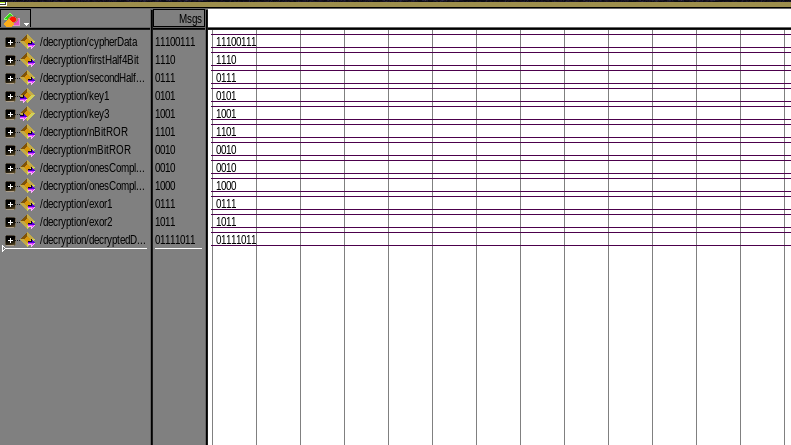
**DECRYPTION -**

Then at the other end of the network this cypher data is entered in the code as cypherData.

**Code-**

|  |
| --- |
| **library ieee; use ieee.std\_logic\_1164.all; use ieee.std\_logic\_arith.all;    entity decryption is port(cypherData: buffer bit\_vector(7 downto 0) := "11100111";  firstHalf4Bit: buffer bit\_vector(3 downto 0);  secondHalf4Bit: buffer bit\_vector(3 downto 0);  key1: in bit\_vector(3 downto 0) := "0101";  key3: in bit\_vector(3 downto 0) := "1001";  nBitROR: buffer bit\_vector(3 downto 0);  mBitROR: buffer bit\_vector(3 downto 0);  onesComplement1: buffer bit\_vector(3 downto 0);  onesComplement2: buffer bit\_vector(3 downto 0);  exor1: buffer bit\_vector(3 downto 0);  exor2: buffer bit\_vector(3 downto 0);  decryptedData: buffer bit\_vector(7 downto 0));  end decryption;   architecture behavior of decryption is  begin    *--\*\*divding the encrypted 8-bit data into two 4-bit binary numbers*    *--first half 4-bit binary of encrypted data*  process is  begin  for i in 0 to 3 loop  firstHalf4Bit(3 - i) <= cypherData(7 - i);  end loop;  wait;  end process;   *--second half 4-bit binary of encrypted data*  process is  begin  for i in 0 to 3 loop  secondHalf4Bit(i) <= cypherData(i);  end loop;  wait;  end process;    *--\*\*first-half of the decryption algorithm*  *--since, key 2(n) was taken 3, so we will do 3-bit right rotations*  nBitROR <= (firstHalf4Bit) ror 3;    *--1's complement of the nBitROR;*  onesComplement1 <= not(nBitROR);    *--xor of the first half of 4-bit binary (achieved as output) and key 1*  exor1 <= onesComplement1 xor key1;    *--\*\*second-half of the decryption algorithm*  *--1's complement of second half 4-bit binary*  onesComplement2 <= not(secondHalf4Bit);  *--since, key 4(m) was taken 2, so we will do 2-bit right rotations*  mBitROR <= (onesComplement2) ror 2;  *--xor of the second half of 4-bit binary (achieved as output) and key 3*  exor2 <= mBitROR xor key3;  *--\*\*let's append the 4-bit data's we got from both the sides*  decryptedData <= exor1 & exor2;  *--the above 8-bit data we got is the decrypted data of the the 8-bit password* end behavior;** |

**Simulation -**



**Result -**After applying the decryption algorithm, we got decrypted data as 01111011(123 in decimal) which was the encrypted data.

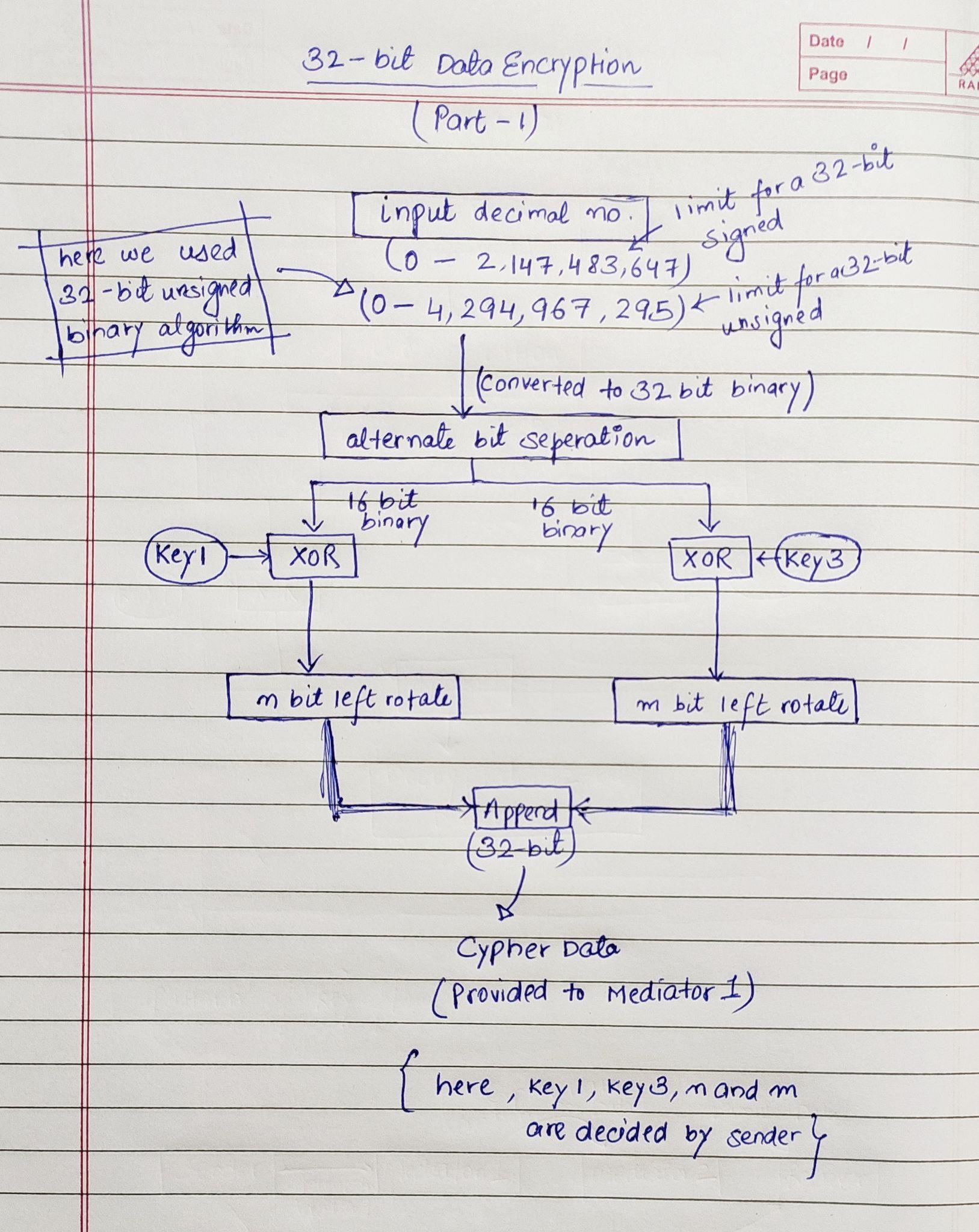
**PART – 3(Our Final work)**

We, designed a 32-bit encryption and decryption algorithm based on the one proposed earlier in our proposal.

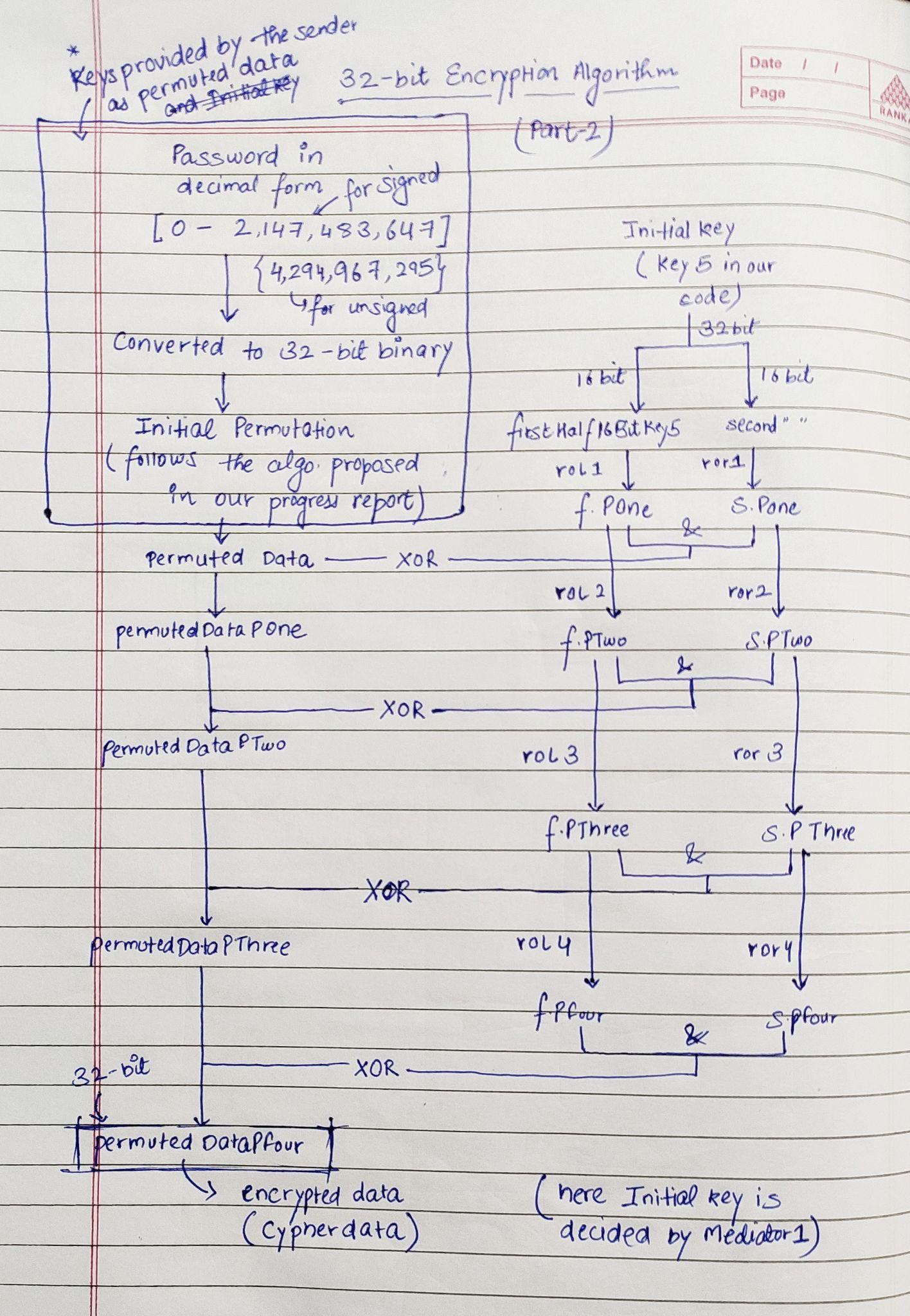
**ENCRYPTION-**

**ALGORITHMS -**

**PART 1 -**



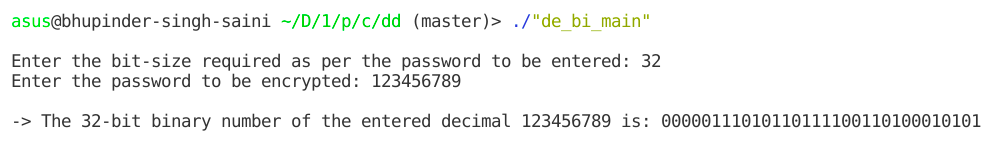
**PART 2 -**



We are using the same cpp program(discussed on Page 8) to convert entered decimal number to n-bit binary (here for 32-bit).

Since, we took 32-bit data encryption (hence from 0 to 2,147,483,647) therefore we will input a decimal (here we took 123456789) which will be converted into 32-bit binary by the program written.

**Terminal-**

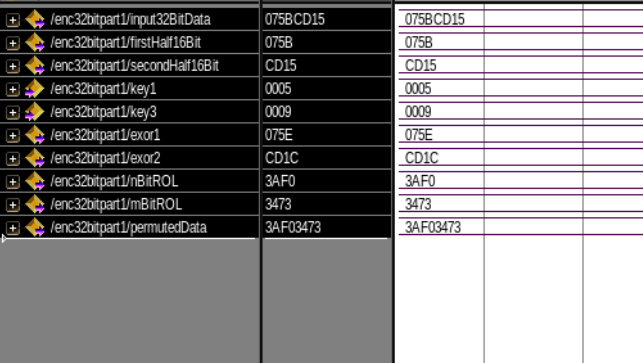


Now, “00000111010110111100110100010101” will be the input as input32BitData which will be converted into cypher data.

**Code(Part 1)-**

|  |
| --- |
| **library ieee; use ieee.std\_logic\_1164.all; use ieee.std\_logic\_arith.all;  entity enc32BitPart1 is port(input32BitData: buffer bit\_vector(31 downto 0) :="00000111010110111100110100010101";  firstHalf16Bit: buffer bit\_vector(15 downto 0);  secondHalf16Bit: buffer bit\_vector(15 downto 0);   key1: in bit\_vector(15 downto 0) := "0000000000000101";  key3: in bit\_vector(15 downto 0) := "0000000000001001";  exor1: buffer bit\_vector(15 downto 0);  exor2: buffer bit\_vector(15 downto 0);  nBitROL: buffer bit\_vector(15 downto 0);  mBitROL: buffer bit\_vector(15 downto 0);  permutedData: buffer bit\_vector(31 downto 0)  );  end enc32BitPart1;   architecture behavior of enc32BitPart1 is begin   *--\*\*divding the 32-bit data into two 16-bit binary numbers*   *--first half 16-bit binary of input data*  process is  begin  for i in 0 to 15 loop  firstHalf16Bit(15 - i) <= input32BitData(31 - i);  end loop;  wait;  end process;   *--second half 16-bit binary of input data*  process is  begin  for i in 0 to 15 loop  secondHalf16Bit(i) <= input32BitData(i);  end loop;  wait;  end process;     *--\*\*first-half of the encryption algorithm*   *--xor of the first half of 16-bit binary and key 1*  exor1 <= firstHalf16Bit xor key1;   *--let's take key 2(n) as 3, therefore we will do 3-bit left rotations*  nBitROL <= (exor1) rol 3;    *--\*\*second-half of the encryption algorithm*   *--xor of the second-half of 16-bit binary and key 3*  exor2 <= secondHalf16Bit xor key3;   *--let's take key 3(m) as 2, therefore we will do 2-bit left rotations*  mBitROL <= (exor2) rol 2;   *--\*\*let's append the 16-bit data's we got from both the sides*  permutedData <= nBitROL & mBitROL;  *--the above 32-bit data is the cypher data of the the 32-bit password*  end behavior;** |

**Simulation -**

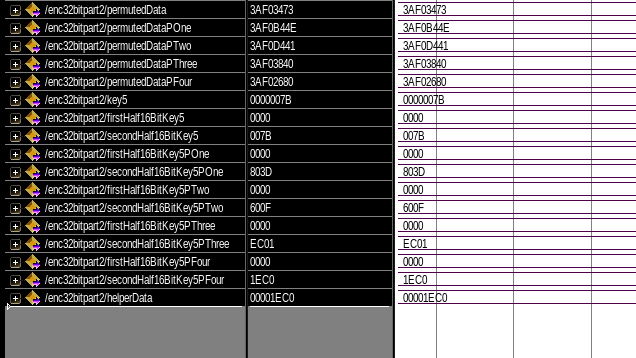
****

**Conclusion -** We got permuted data as ‘3AF03473’ which will serve as input the our second program.

**Code(Part 2)-**

|  |
| --- |
| **library ieee; use ieee.std\_logic\_1164.all; use ieee.std\_logic\_arith.all;  entity enc32BitPart2 is port(  permutedData: buffer bit\_vector(31 downto 0) := x"3AF03473";   permutedDataPOne: buffer bit\_vector(31 downto 0);   permutedDataPTwo: buffer bit\_vector(31 downto 0);   permutedDataPThree: buffer bit\_vector(31 downto 0);   permutedDataPFour: buffer bit\_vector(31 downto 0);   key5: buffer bit\_vector(31 downto 0) := "00000000000000000000000001111011";   firstHalf16BitKey5: buffer bit\_vector(15 downto 0);   secondHalf16BitKey5: buffer bit\_vector(15 downto 0);   firstHalf16BitKey5POne: buffer bit\_vector(15 downto 0) :=(others => '0');   secondHalf16BitKey5POne: buffer bit\_vector(15 downto 0) :=(others => '0');   firstHalf16BitKey5PTwo: buffer bit\_vector(15 downto 0) :=(others => '0');   secondHalf16BitKey5PTwo: buffer bit\_vector(15 downto 0) :=(others => '0');   firstHalf16BitKey5PThree: buffer bit\_vector(15 downto 0) :=(others => '0');   secondHalf16BitKey5PThree: buffer bit\_vector(15 downto 0) :=(others => '0');    firstHalf16BitKey5PFour: buffer bit\_vector(15 downto 0) :=(others => '0');   secondHalf16BitKey5PFour: buffer bit\_vector(15 downto 0) :=(others => '0');   helperData: buffer bit\_vector(31 downto 0) :=(others => '0') );  end enc32BitPart2;   architecture behavior of enc32BitPart2 is begin    *--\*\*dividing intial key(key5) into equal 16 bit halves*   *--first half 16-bit binary of input data*  process is  begin  for i in 0 to 15 loop  firstHalf16BitKey5(15 - i) <= key5(31 - i);  end loop;  wait;  end process;   *--second half 16-bit binary of input data*  process is  begin  for i in 0 to 15 loop  secondHalf16BitKey5(i) <= key5(i);  end loop;  wait;  end process;   *--\*\*loop 4 times*  *--1st loop*  firstHalf16BitKey5POne <= (firstHalf16BitKey5) rol 1;  *--firstHalf16BitKey5P1*  secondHalf16BitKey5POne <= (secondHalf16BitKey5) ror 1;  permutedDataPOne <= permutedData xor (firstHalf16BitKey5POne & secondHalf16BitKey5POne);   *--2nd loop*  firstHalf16BitKey5PTwo <= firstHalf16BitKey5POne rol 2;  secondHalf16BitKey5PTwo <= secondHalf16BitKey5POne ror 2;  permutedDataPTwo <= permutedDataPOne xor (firstHalf16BitKey5PTwo & secondHalf16BitKey5PTwo);   *--3rd loop*  firstHalf16BitKey5PThree <= firstHalf16BitKey5PTwo rol 3;  secondHalf16BitKey5PThree <= secondHalf16BitKey5PTwo ror 3;  permutedDataPThree <= permutedDataPTwo xor (firstHalf16BitKey5PThree & secondHalf16BitKey5PThree);   *--4th loop*  firstHalf16BitKey5PFour <= firstHalf16BitKey5PThree rol 4;  secondHalf16BitKey5PFour <= secondHalf16BitKey5PThree ror 4;  helperData <= (firstHalf16BitKey5PFour & secondHalf16BitKey5PFour);  permutedDataPFour <= permutedDataPThree xor (firstHalf16BitKey5PFour & secondHalf16BitKey5PFour);    *--\*\*cypher text 32 bit = permutedDataP4;*  end behavior;** |

**Simulation -**

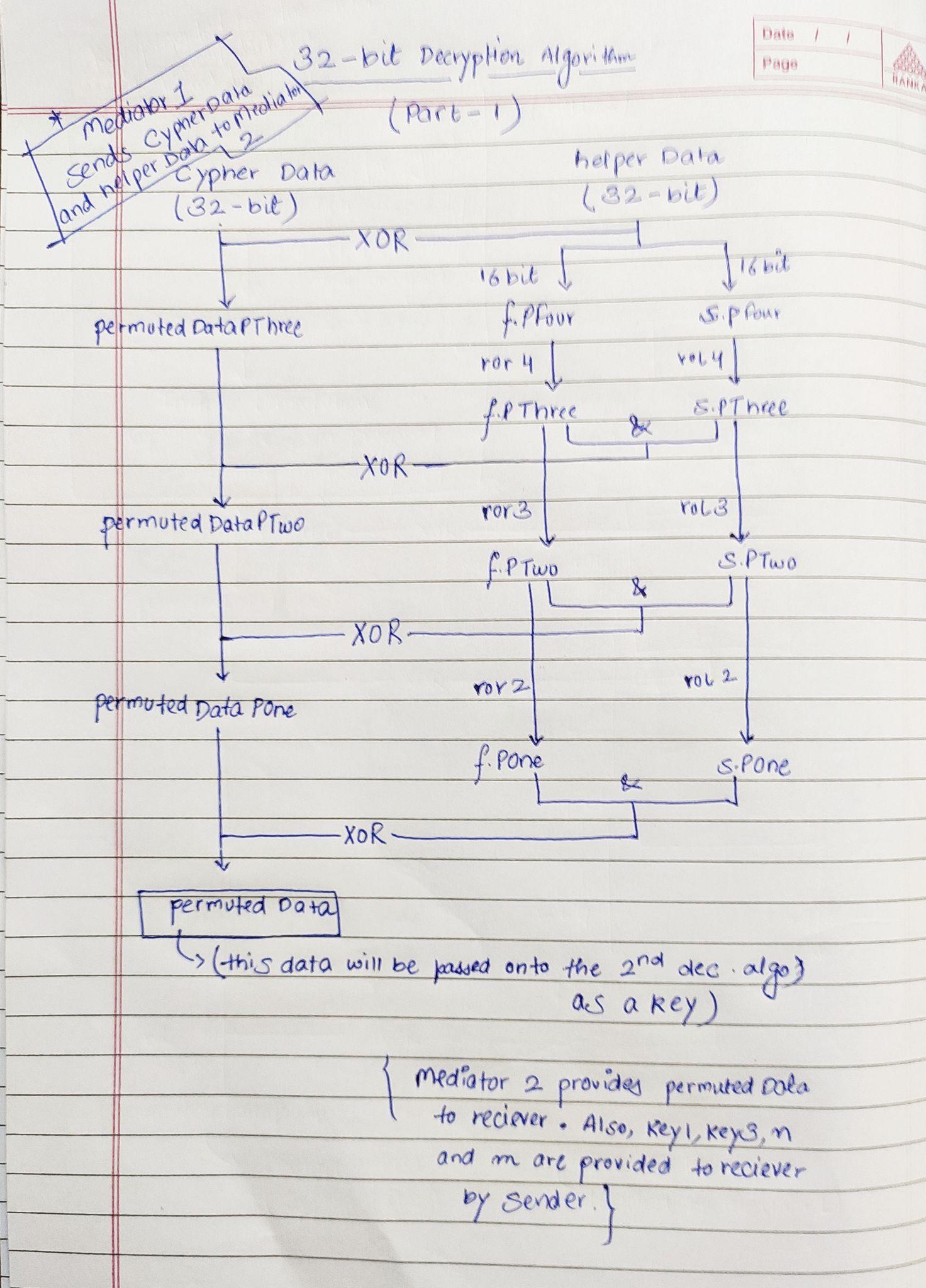


**Conclusion -** We got encrypted data as ‘3AF02680’ and helper data as ‘00001EC0’which will serve as keys for our decryption algorithm.

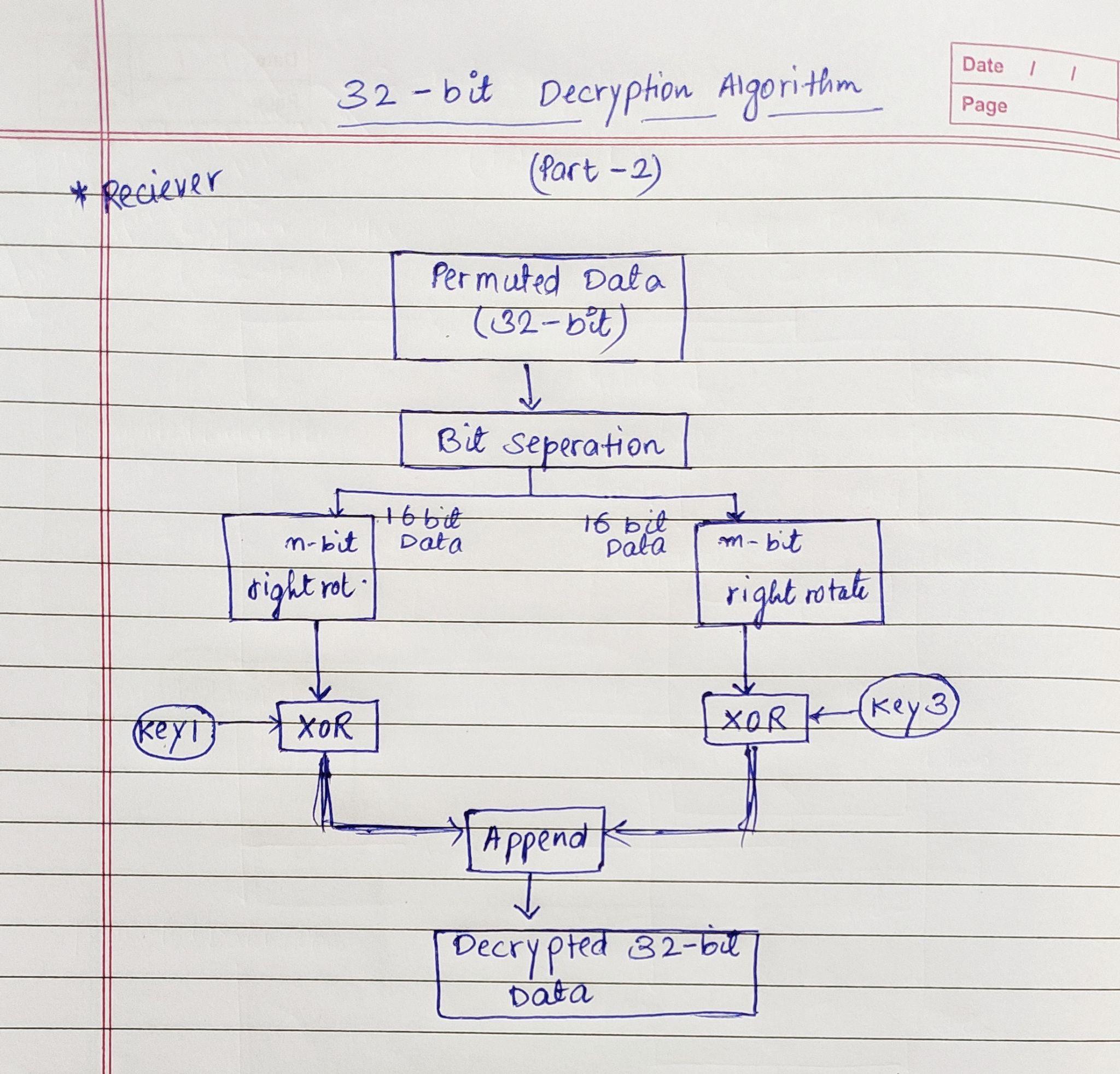
**DECRYPTION -**

**ALGORITHMS -**

**PART 1 -**

****

**PART 2 -**

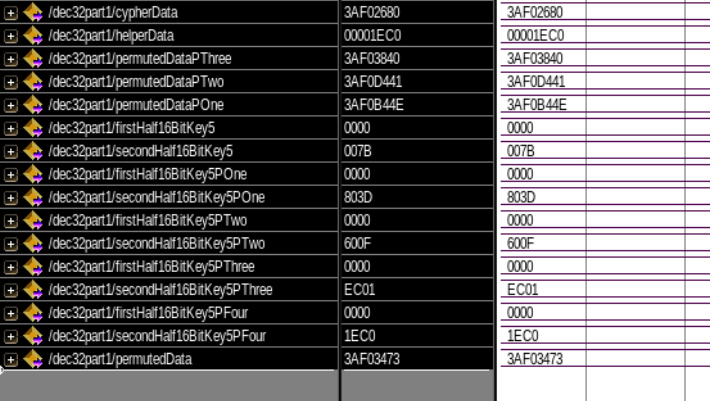
****

Then at the other end of network this cypher data is entered in the code as cypherData.

**Code(Part 1)-**

|  |
| --- |
| **library ieee; use ieee.std\_logic\_1164.all; use ieee.std\_logic\_arith.all;  entity dec32Part1 is port(  cypherData: buffer bit\_vector(31 downto 0) := x"0020ABD8";  helperData: buffer bit\_vector(31 downto 0) := x"0020ABD8";  permutedDataPThree: buffer bit\_vector(31 downto 0);  permutedDataPTwo: buffer bit\_vector(31 downto 0);  permutedDataPOne: buffer bit\_vector(31 downto 0);  firstHalf16BitKey5: buffer bit\_vector(15 downto 0);  secondHalf16BitKey5: buffer bit\_vector(15 downto 0);  firstHalf16BitKey5POne: buffer bit\_vector(15 downto 0) :=(others => '0');  secondHalf16BitKey5POne: buffer bit\_vector(15 downto 0) :=(others => '0');  firstHalf16BitKey5PTwo: buffer bit\_vector(15 downto 0) :=(others => '0');  secondHalf16BitKey5PTwo: buffer bit\_vector(15 downto 0) :=(others => '0');  firstHalf16BitKey5PThree: buffer bit\_vector(15 downto 0) :=(others => '0');  secondHalf16BitKey5PThree: buffer bit\_vector(15 downto 0) :=(others => '0');  firstHalf16BitKey5PFour: buffer bit\_vector(15 downto 0) :=(others => '0');  secondHalf16BitKey5PFour: buffer bit\_vector(15 downto 0) :=(others => '0');  permutedData: buffer bit\_vector(31 downto 0) );  end dec32Part1;   architecture behavior of dec32Part1 is  begin *-- \*\*\*with the help of xoring of cypherData and helperData we* *-- will retrieve back permutedDataPThree* *-- 1st step* permutedDataPThree <= cypherData xor helperData; *--first half 16-bit binary of helper data* process is  begin  for i in 0 to 15 loop  firstHalf16BitKey5PFour(15 - i) <= helperData(31 - i);  end loop;  wait;  end process; *--second half 16-bit binary of input data* process is  begin  for i in 0 to 15 loop  secondHalf16BitKey5PFour(i) <= helperData(i);  end loop;  wait;  end process; *--ROR by 4 the firstHalf16BitKey5PFour and ROL by 4 the secondHalf16BitKey5PFour* firstHalf16BitKey5PThree <= firstHalf16BitKey5PFour ror 4; secondHalf16BitKey5PThree <= secondHalf16BitKey5PFour rol 4; *-- 2nd step* permutedDataPTwo <= permutedDataPThree xor (firstHalf16BitKey5PThree & secondHalf16BitKey5PThree); *--ROR by 3 the firstHalf16BitKey5PThree and ROL by 3 the secondHalf16BitKey5PThree* firstHalf16BitKey5PTwo <= firstHalf16BitKey5PThree ror 3; secondHalf16BitKey5PTwo <= secondHalf16BitKey5PThree rol 3; *-- 3rd step* permutedDataPOne <= permutedDataPTwo xor (firstHalf16BitKey5PTwo & secondHalf16BitKey5PTwo); *--ROR by 2 the firstHalf16BitKey5PTwo and ROL by 2 the secondHalf16BitKey5PTwo* firstHalf16BitKey5POne <= firstHalf16BitKey5PTwo ror 2; secondHalf16BitKey5POne <= secondHalf16BitKey5PTwo rol 2; *-- 4th step* permutedData <= permutedDataPOne xor (firstHalf16BitKey5POne & secondHalf16BitKey5POne); *--ROR by 1 the firstHalf16BitKey5PThree and ROL by 1 the secondHalf16BitKey5PThree* firstHalf16BitKey5 <= firstHalf16BitKey5POne ror 1; secondHalf16BitKey5 <= secondHalf16BitKey5POne rol 1; end behavior;** |

**Simulation -**

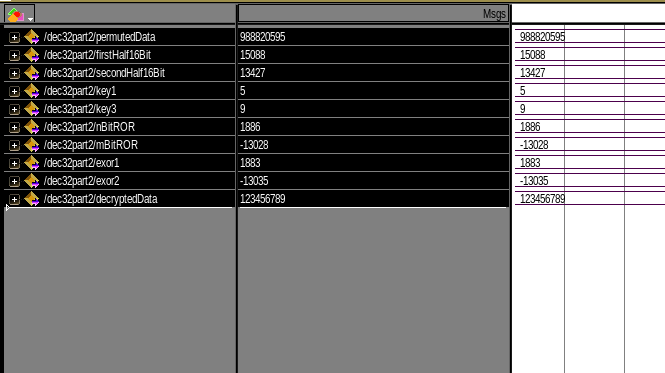


**Conclusion -** We got permuted data as ‘3AF03473’ which will serve as input to our second program of decryption.

**Code(Part 2)-**

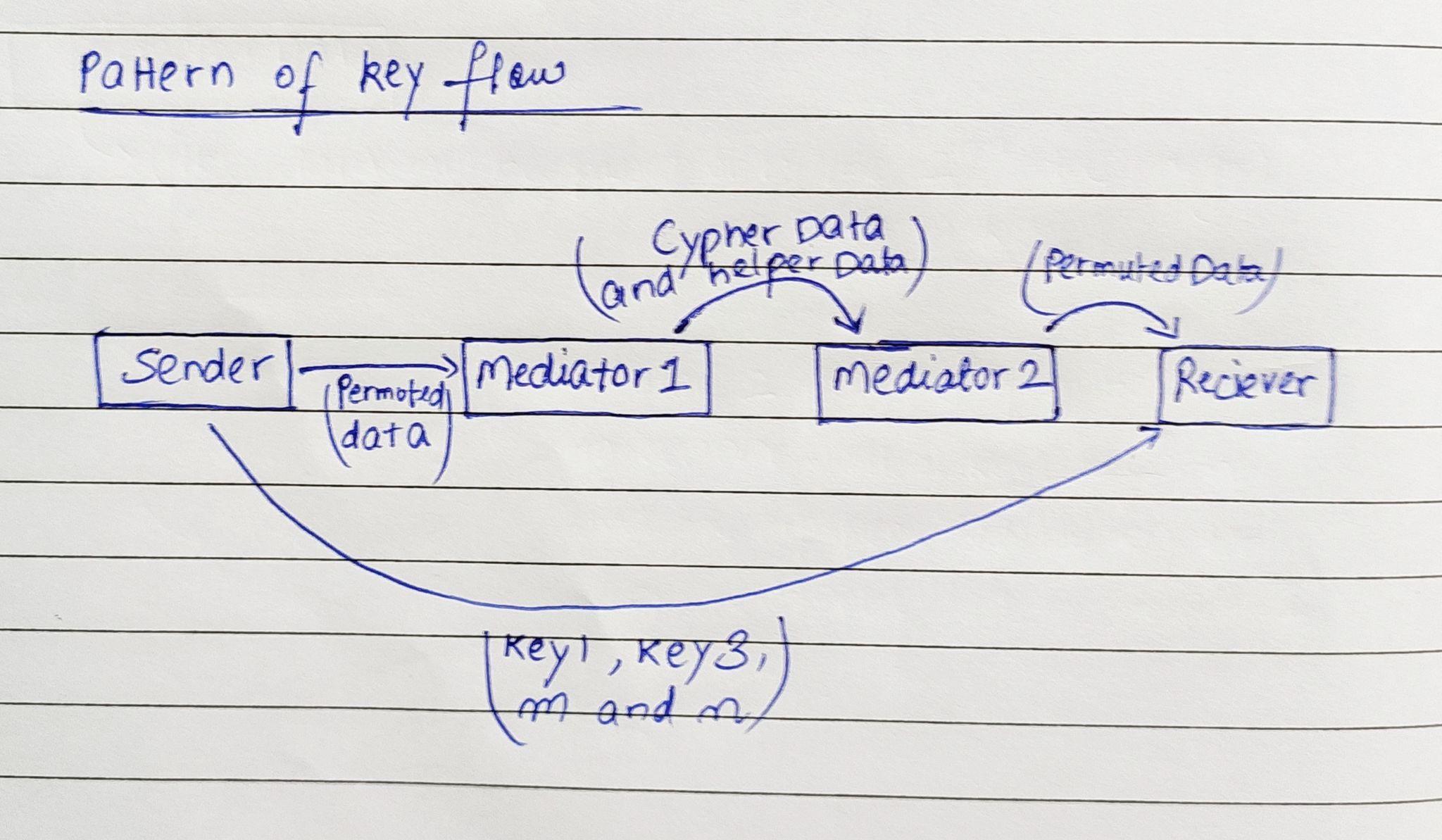
|  |
| --- |
| **library ieee; use ieee.std\_logic\_1164.all; use ieee.std\_logic\_arith.all;  entity dec32Part2 is port(  permutedData: buffer bit\_vector(31 downto 0) := x"3AF03473";  firstHalf16Bit: buffer bit\_vector(15 downto 0);  secondHalf16Bit: buffer bit\_vector(15 downto 0);  key1: buffer bit\_vector(15 downto 0) := "0000000000000101";  key3: buffer bit\_vector(15 downto 0) := "0000000000001001";  nBitROR: buffer bit\_vector(15 downto 0);  mBitROR: buffer bit\_vector(15 downto 0);  exor1: buffer bit\_vector(15 downto 0);  exor2: buffer bit\_vector(15 downto 0);  decryptedData: buffer bit\_vector(31 downto 0));   end dec32Part2;   architecture behavior of dec32Part2 is  begin   *--\*\*divding the encrypted 32-bit data into two 16-bit binary numbers*  *--first half 16-bit binary of encrypted data*  process is  begin  for i in 0 to 15 loop  firstHalf16Bit(15 - i) <= permutedData(31 - i);  end loop;  wait;  end process;   *--second half 16-bit binary of encrypted data*  process is  begin  for i in 0 to 15 loop  secondHalf16Bit(i) <= permutedData(i);  end loop;  wait;  end process;     *--\*\*first-half of the decryption algorithm*  *--since, key 2(n) was taken 3, so we will do 3-bit right rotations*  nBitROR <= (firstHalf16Bit) ror 3;  *--xor of the first half of 16-bit binary (achieved as output) and key 1*  exor1 <= nBitROR xor key1;    *--\*\*second-half of the decryption algorithm*   *--since, key 4(m) was taken 2, so we will do 2-bit right rotations*  mBitROR <= (secondHalf16Bit) ror 2;  *--xor of the second half of 16-bit binary (achieved as output) and key 3*  exor2 <= mBitROR xor key3;  *--\*\*let's append the 16-bit data's we got from both the sides*  decryptedData <= exor1 & exor2;  *--the above 32-bit data is the decrypted data of the the 32-bit password*  end behavior;** |

**Simulation -**

****

**RESULT -** After applying the 2-step decryption algorithm, we got decrypted data as “00000111010110111100110100010101” (123456789 in decimal) which was the encrypted data.

**KEY FLOW -**



**CONCLUSION:**

So, with the help of our knowledge of programming on VHDL and ecryption-decryption fundamentals, we were able to code, simulate and verify our idea inspired from DES(Data Encryption Standard) algorithm. Also, as proposed, we were able to encrypt as well as decrypt a 32 bit binary digit.

Besides, with the help of distributed encryption and decryption codes we were able to achieve a very basic implementation of Distributed System Security.

**FUTURE SCOPE:**

The project is highly usable in practical scenarios where the data to be shared is highly confidential like secret military information, strategical information and even in social media platforms where personal data privacy is required.

**PORTIONS OF REPORT:**

Encryption(Algorithm, code and simulation) - Bhupinder Singh Saini(2K19EC/043)

Decryption(Algorithm, code and simulation) - Ayussh Vashishth(2K19/EC/039)

**REFERENCES:**

[**https://www.simplilearn.com/what-is-des-article#:~:text=The%20DES%20**](https://www.simplilearn.com/what-is-des-article#:~:text=The%20DES%20)

[**https://www.youtube.com/watch?v=8B1rN1rnTiU**](https://www.youtube.com/watch?v=8B1rN1rnTiU)

[**https://en.wikipedia.org/wiki/Distributed\_System\_Security\_Architecture**](https://en.wikipedia.org/wiki/Distributed_System_Security_Architecture)

[**https://www.intel.com/content/www/us/en/programmable/support/support-resources/design-examples/design-software/vhdl/v\_hex.html**](https://www.intel.com/content/www/us/en/programmable/support/support-resources/design-examples/design-software/vhdl/v_hex.html)

[**https://www.ics.uci.edu/~jmoorkan/vhdlref/aggregat.html**](https://www.ics.uci.edu/~jmoorkan/vhdlref/aggregat.html)

[**https://vhdlwhiz.com/**](https://vhdlwhiz.com/)