Network Security

Assignment 2 - Project 1

Prof. B N Jain

Submitted by:

Sarvagya Kaushik - 2021350

Kunal Sharma - 2021331

The programming language used in the assignment is C++. In our code we have in total **15 functions** including the main() function. Below is the explanation of each function one by one.

1. void rotate(vector<uint8_t> &vec, int d) : This function **rotates the elements** of a vector 'vec' **to the left by 'd' positions**. It achieves this by repeatedly moving the first element to the end of the vector 'd' times.

2. vector<int> matrixMultiplicationAndAddition(const int matrixA[][8], const int matrixB[], const vector<int> &matrixC) : This function performs **matrix multiplication** between 'matrixA' and 'matrixC', followed by element-wise addition with 'matrixB'. It returns a vector named 'result' containing the resulting values. The multiplication is achieved by **bitwise AND** between corresponding elements of rows of 'matrixA' and 'matrixC', followed by **XOR** operations to accumulate the results. Then, each element of 'matrixB' is XORed with the corresponding element of the result vector to perform addition.

3. uint8_t gf_multiply(uint8_t a, uint8_t b) : This function does **Galois Field multiplication** between two 8-bit integers 'a' and 'b'. It iterates through the bits of 'b', XORs 'a' with the result if the least significant bit of 'b' is set, and updates 'a' according to the **Galois Field polynomial**. The final result is returned.

4. uint8_t m_inv(uint8_t x) : This function calculates the **multiplicative inverse** of a given 8-bit integer 'x' within a Galois Field. It iterates through possible values of 'i', computing the product of 'x' and 'i' using the **gf_multiply() function** and returns the first 'i' for which the product is **equal to 0x01**, indicating the multiplicative inverse.

5. uint8_t substitution(uint8_t num) : This function performs a **substitution operation** on an 8-bit integer 'num' using a **predefined matrix multiplication and addition operation**. It first calculates the multiplicative inverse of num using the m_inv() function. Then, it initializes two matrices matrixA and matrixB and converts the multiplicative inverse into a bitset. The function then performs matrix multiplication and addition using matrixMultiplicationAndAddition() function. At last, it converts the resulting bitset back to an integer and returns.

6. uint8_t inv_substitution(uint8_t num) : This function performs an **inverse substitution operation** on an 8-bit integer 'num'. It initializes two matrices 'matrixA' and 'matrixB', converts ‘num’ into a bitset, and performs matrix multiplication and addition using the matrixMultiplicationAndAddition function(). Then, it converts the resulting bitset back to an integer, calculates the multiplicative inverse using m_inv() function, and returns the result.

7. uint8_t RC(int x) : This function **calculates the round constant for AES encryption** using recursion. If the input 'x' is 0x01, it returns 0x01; otherwise, it recursively calculates the round constant by multiplying '0x02' with the round constant of the previous round (x - 1).

8. void g_func(vector<uint8_t> &vec, int round_number) : This function **performs the ‘g’ function** used in AES encryption for a given round. It rotates the input vector, substitutes its elements, and finally **XORs** the first element with the round constant calculated using RC function.

9. void create_round_key(vector<vector<uint8_t>> &prev_round_key, int round_number): This function **creates the round key for AES encryption** based on the previous round key and the current round number. It generates a temporary vector ‘vec’ by extracting the last column of the previous round key, applies the ‘g’ function using g_func(), and then updates the round key by **XORing** each element with either the corresponding element of ‘vec’ or the element of the previous column, based on the column index.

10. void inv_create_round_key(vector<vector<uint8_t>>&prev_round_key,int r_number): This function **creates the round key for AES decryption** based on the previous round key and the current round number. It iteratively XORs elements of the previous round key columns either with the corresponding element from a temporary vector ‘vec’ or the element from the previous column. Before updating each column, it constructs ‘vec’ from the last column of the previous round key and **applies the ‘g’ function in reverse order**.

11. vector<vector<uint8_t>>encrypt_round(vector<vector<uint8_t>> plaintext_matrix_he, vector<vector<uint8_t>>ciphertext_matrix_hex,int r_number, vector<vector<uint8_t>> &prev_round_key) : This function **represents one round of AES encryption**. It performs substitution and row-shifting operations on the plaintext matrix to produce the ciphertext matrix. If it's not the final round, it additionally applies the MixColumns operation by multiplying a fixed matrix with the current state of ciphertext. At last, it combines the result with the round key using **XOR**. If it's the final round, it only combines the ciphertext with the round key.

12. vector<vector<uint8_t>> decrypt_round(vector<vector<uint8_t>> plaintext_matrix_he, vector<vector<uint8_t>> ciphertext_matrix_hex, int r_number, vector<vector<uint8_t>> &prev_round_key) : This function **represents one round of AES decryption**. It starts by combining the ciphertext matrix with the round key using **XOR**. If it's not the first round, it then performs the Inverse MixColumns operation by multiplying the fixed inverse matrix with the current state of ciphertext. Next, it applies the Inverse ShiftRows operation by rotating rows in the opposite direction. Finally, it applies the Inverse Substitution operation to the ciphertext matrix to obtain the plaintext matrix for that round.

13. vector<vector<uint8_t>> encrypt(vector<vector<string>> plaintext_matrix_hex, vector<vector<uint8_t>>&ciphertext_matrix_hex,int r_number, vector<vector<uint8_t>> &key) : This function **performs AES encryption on a plaintext matrix**. It begins with an initial Add Round operation, where each element of the plaintext matrix is **XORed** with the corresponding element of the encryption key to produce the ciphertext matrix. Then, it iterates through 10 rounds of encryption using the encrypt_round() function, updating the ciphertext matrix accordingly and printing the output of the 1st and 9th encryption rounds.

14. vector<vector<uint8_t>> decrypt(vector<vector<string>> ciphertext_matrix_hex, vector<vector<uint8_t>> &plaintext_matrix_hex, int r_number, vector<vector<uint8_t>> &key) : This function **performs AES decryption on a ciphertext matrix**. It first converts the ciphertext matrix to byte format. Then, it iterates through 10 rounds of decryption using the decrypt_round function, updating the plaintext matrix accordingly and optionally printing the output of the 1st and 9th decryption rounds. Finally, it applies the last Add Round operation by combining the result with the final round key and returns the resulting plaintext matrix.

15. int main(): The main() function **reads plaintext data from a file named ‘input.txt’**, converts it into hexadecimal format, encrypts it using AES, and then decrypts the ciphertext back to plaintext. It iterates this process three times for different input data. The encryption and decryption are performed using AES encryption and decryption functions respectively, with a predefined encryption key. Finally, it prints out the ciphertext and decrypted plaintext for each iteration.

Sample inputs and outputs

Inputs: The plaintexts are taken from the file input.txt and key is defined in the main() function.

Plaintexts:

```
kunalsharma12345
sarvagyakaushik1
thereRonly2peeps
```

Key:

```
{{0xea, 0x2d, 0x82, 0x7f},
  {0xd2, 0x37, 0x34, 0x8d},
  {0x73, 0xbf, 0x5c, 0x29},
  {0x21, 0x09, 0xa8, 0x2f}};
```

Outputs:

```
Initial Plaintext: kunalsharma12345
Output of 1st encryption round: 358dc9c066aa26802168be6c87ce122
Output of 9th encryption round: d7f8de5d16b5f5310b5d9934b801f
Ciphertext: 1f7c3d35c45c70dd6228195f17427f8
Output of 1st decryption round: d7f8de5d16b5f5310b5d9934b801f
Output of 9th decryption round: 358dc9c066aa26802168be6c87ce122
Final Plaintext: kunalsharma12345

Initial Plaintext: sarvagyakaushik1
Output of 1st encryption round: dbc524411409bd47a5bdf6c41f81123
Output of 9th encryption round: fc5e37f93f93ef3aadc74b98828d5b6
Ciphertext: f4cc8bb429283f16c463e319c499ce
Output of 1st decryption round: fc5e37f93f93ef3aadc74b98828d5b6
Output of 9th decryption round: dbc524411409bd47a5bdf6c41f81123
Final Plaintext: sarvagyakaushik1

Initial Plaintext: thereRonly2peeps
Output of 1st encryption round: 7f3f58b979e79c4f72a4fa4d7aadb998
Output of 9th encryption round: 86b4e2db33c794e162475017f88a3095
Ciphertext: bb3debecdc60ac62ab45dea5f72c6a4
Output of 1st decryption round: 86b4e2db33c794e162475017f88a3095
Output of 9th decryption round: 7f3f58b979e79c4f72a4fa4d7aadb998
Final Plaintext: thereRonly2peeps
```