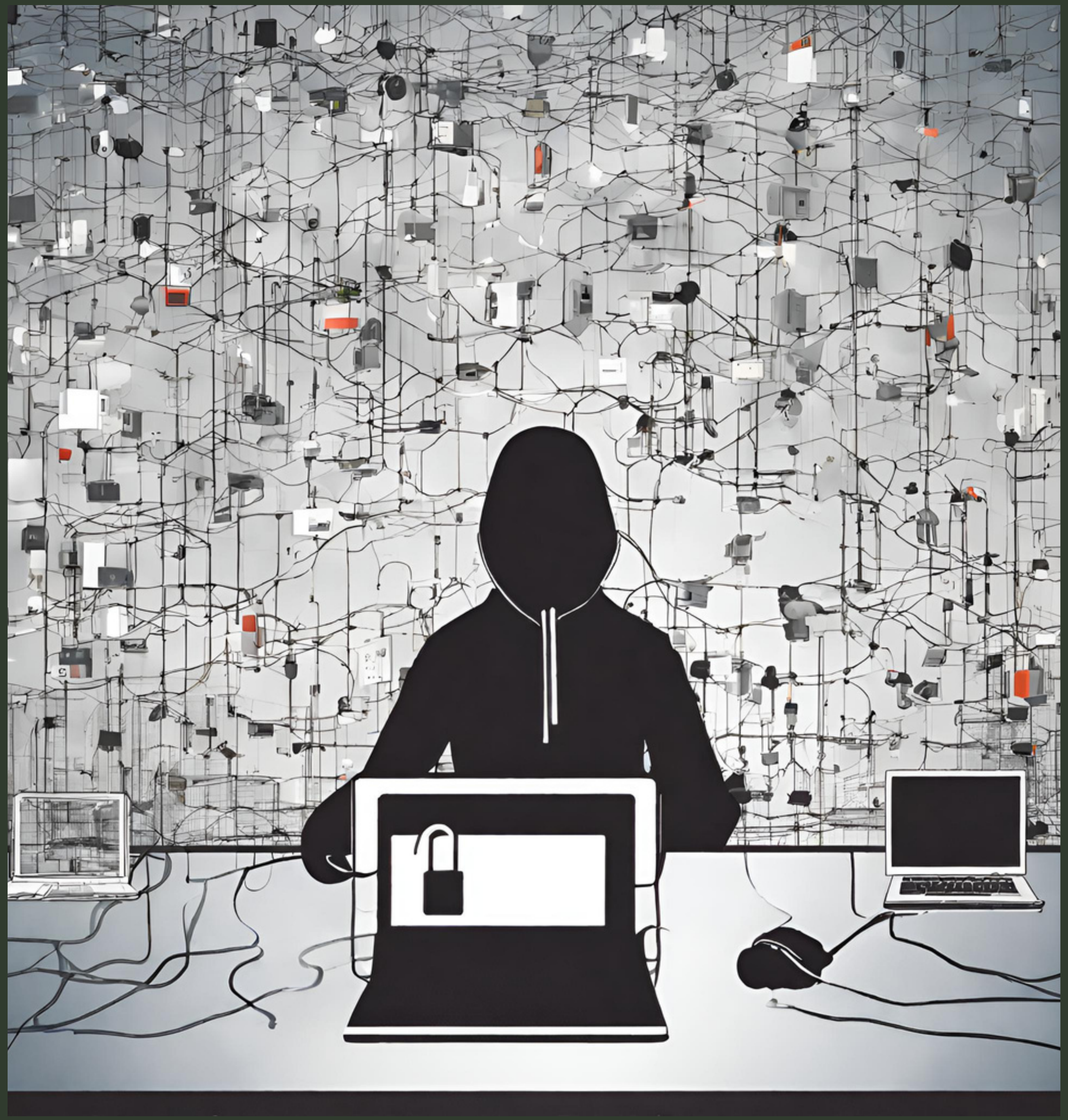
Feb 01, 2024

Assignment 1
Project - 2

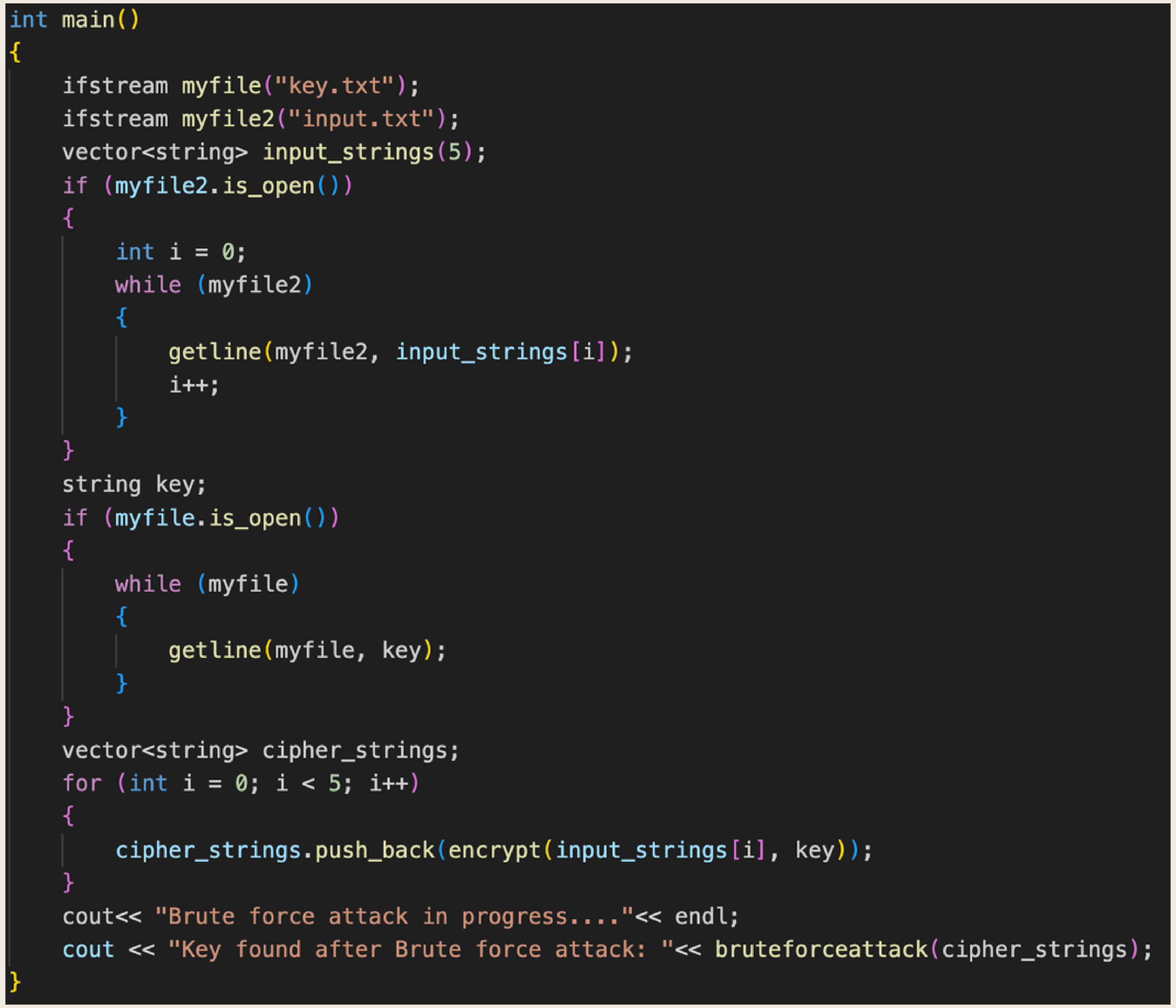
Presented by Sarvagya Kaushik & Kunal Sharma

Winter 2024



Network Security

Prof. B N Jain



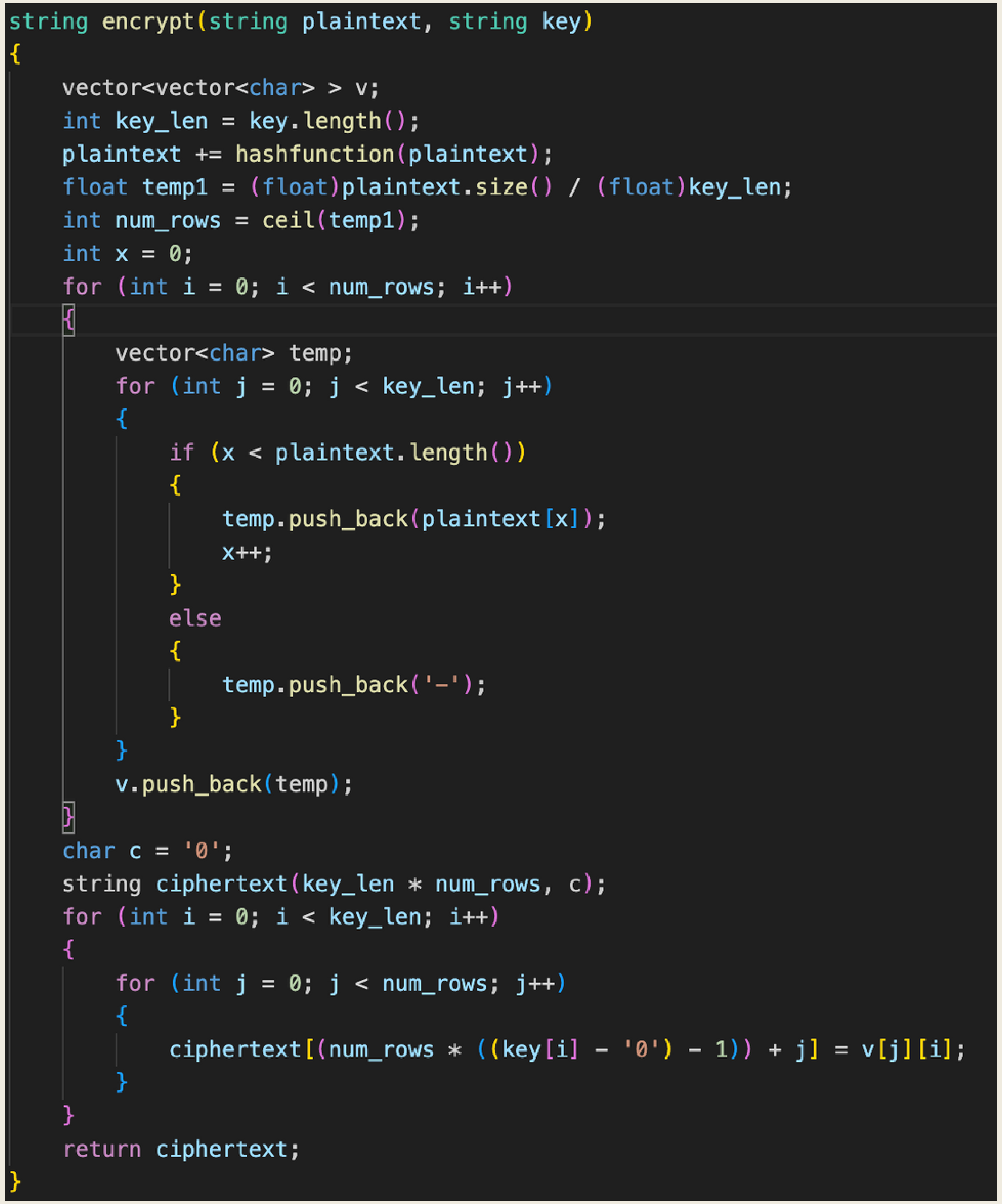
```
int main()
{
    ifstream myfile("key.txt");
    ifstream myfile2("input.txt");
    vector<string> input_strings(5);
    if (myfile2.is_open())
    {
        int i = 0;
        while (myfile2)
        {
            getline(myfile2, input_strings[i]);
            i++;
        }
    }
    string key;
    if (myfile.is_open())
    {
        while (myfile)
        {
            getline(myfile, key);
        }
    }
    vector<string> cipher_strings;
    for (int i = 0; i < 5; i++)
    {
        cipher_strings.push_back(encrypt(input_strings[i], key));
    }
    cout<< "Brute force attack in progress...."<< endl;
    cout << "Key found after Brute force attack: "<< bruteforceattack(cipher_strings);
}
```

Main Function

This is the main function of the code. We have two text files, one containing the key and the other file containing the 5 input plaintexts. In this function we are reading from the input and the key from these files and key is stored in the string variable named key and the plaintexts are stored in a vector named input_strings.

Then we are calling the encryption function to encrypt the plaintexts and then we are doing a brute force attack to find the key used to encrypt the plaintext. At last the key is printed on the console.

Encryption Function

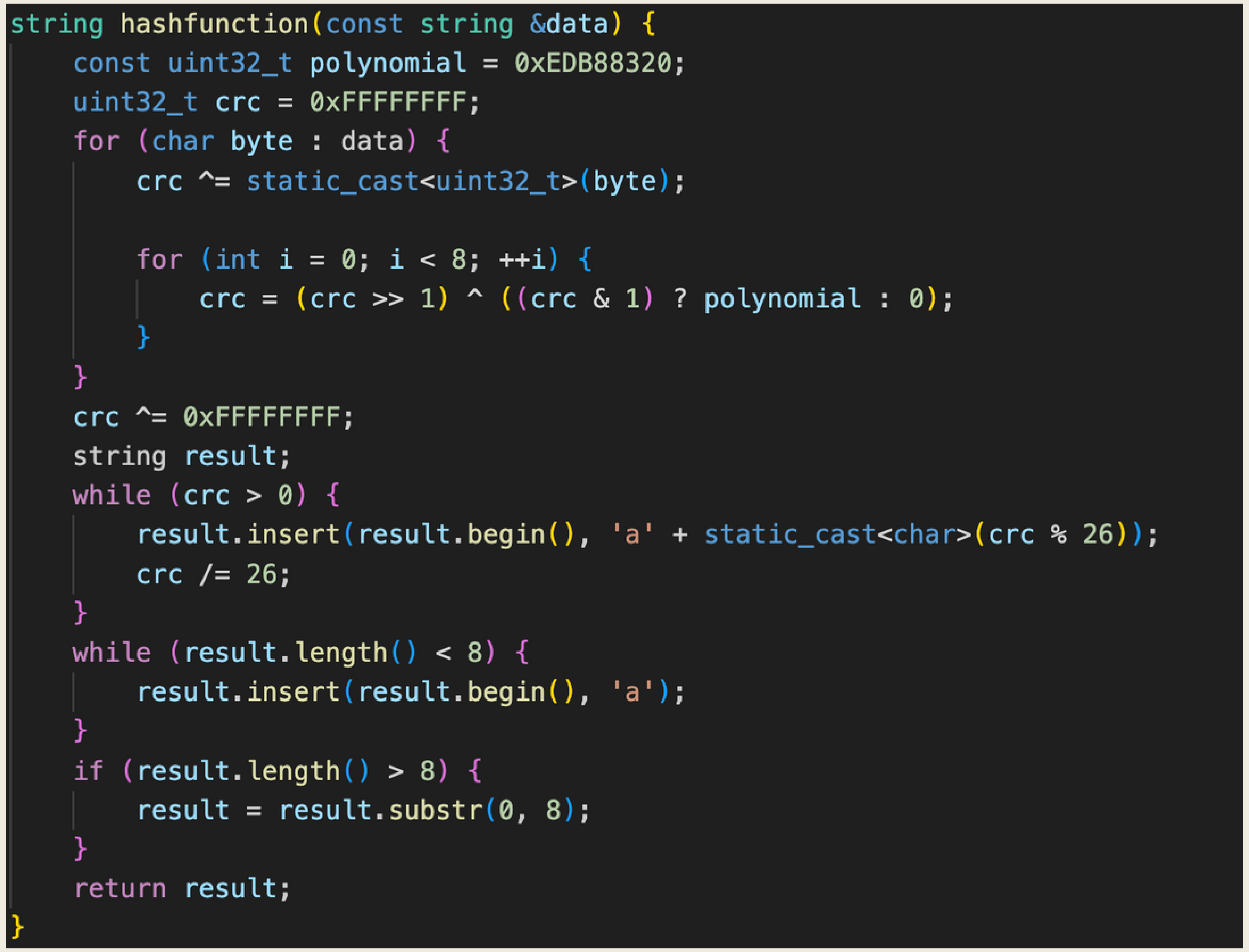


```cpp
string encrypt(string plaintext, string key)
{
    vector<vector<char> > v;
    int key_len = key.length();
    plaintext += hashfunction(plaintext);
    float temp1 = (float)plaintext.size() / (float)key_len;
    int num_rows = ceil(temp1);
    int x = 0;
    for (int i = 0; i < num_rows; i++)
    {
        vector<char> temp;
        for (int j = 0; j < key_len; j++)
        {
            if (x < plaintext.length())
            {
                temp.push_back(plaintext[x]);
                x++;
            }
            else
            {
                temp.push_back('-');
            }
        }
        v.push_back(temp);
    }
    char c = '0';
    string ciphertext(key_len * num_rows, c);
    for (int i = 0; i < key_len; i++)
    {
        for (int j = 0; j < num_rows; j++)
        {
            ciphertext[(num_rows * ((key[i] - '0') - 1)) + j] = v[j][i];
        }
    }
    return ciphertext;
}
```

We have used a transposition cipher to encrypt the plaintext. In this function first we added the hash to the end of plaintext using the function hashfunction(). The function then determines the number of rows needed for the transposition based on the length of the key and the size of the extended plaintext.

The function creates a 2D vector v to store the transposed plaintext. It iterates through the plaintext and fills the vector row-wise. If the vector is not fully filled then we have added the hyphens '-' at the end of the vector. Subsequently, it constructs the ciphertext by rearranging the characters in the vector based on the key's order. At last the final encrypted string is returned.

Hash Function



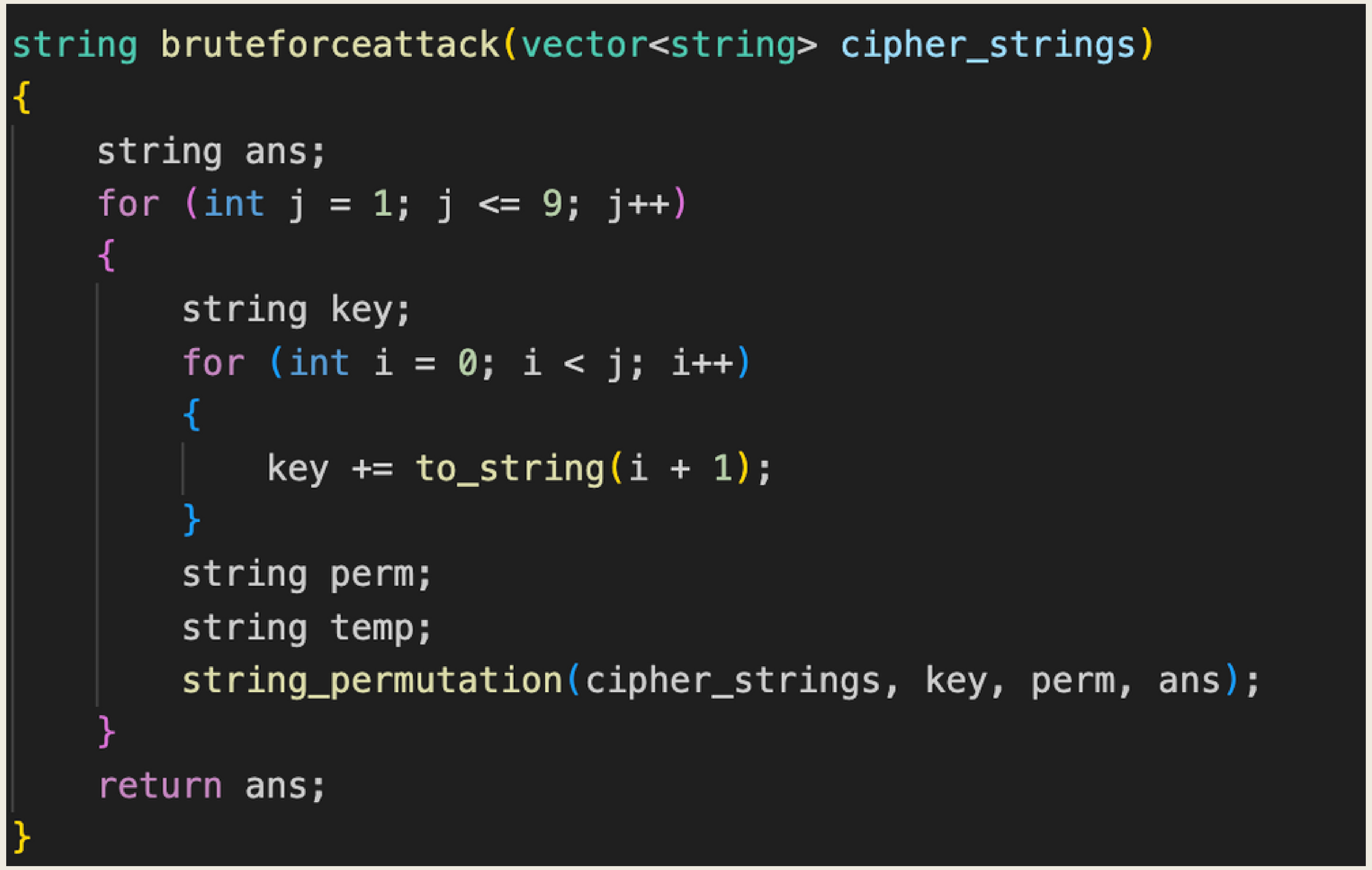
```
string hashfunction(const string &data) {
    const uint32_t polynomial = 0xEDB88320;
    uint32_t crc = 0xFFFFFFFF;
    for (char byte : data) {
        crc ^= static_cast<uint32_t>(byte);

        for (int i = 0; i < 8; ++i) {
            crc = (crc >> 1) ^ ((crc & 1) ? polynomial : 0);
        }
    }
    crc ^= 0xFFFFFFFF;
    string result;
    while (crc > 0) {
        result.insert(result.begin(), 'a' + static_cast<char>(crc % 26));
        crc /= 26;
    }
    while (result.length() < 8) {
        result.insert(result.begin(), 'a');
    }
    if (result.length() > 8) {
        result = result.substr(0, 8);
    }
    return result;
}
```

We have used the CRC32 (Cyclic Redundancy Check) algorithm to generate the hash. This function repeats through every byte of the input string before setting a CRC variable's underlying worth to 0xFFFFFFFF.

For each byte, it performs a bitwise XOR operation and updates the CRC using a polynomial (0xEDB88320) in a loop. After processing all bytes, the function XORs the CRC with 0xFFFFFFFF and converts the resulting CRC value into an 8-character string representation using base-26 encoding, where each digit represents a letter from 'a' to 'z'.

The function guarantees the result string is precisely 8 characters in length, padding with 'a' if necessary or shortening it if longer.

Brute force attack Function

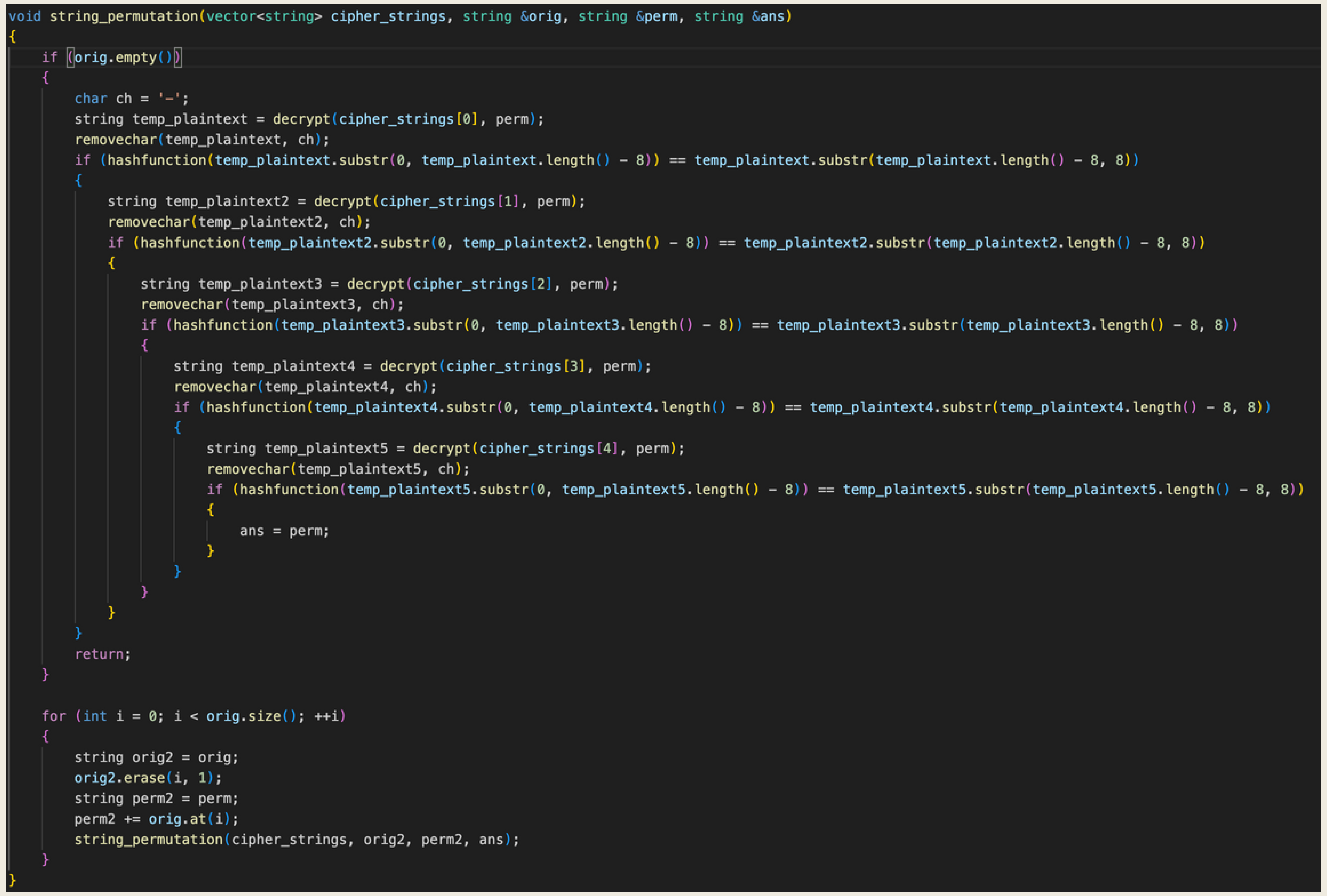


```cpp
string bruteforceattack(vector<string> cipher_strings)
{
    string ans;
    for (int j = 1; j <= 9; j++)
    {
        string key;
        for (int i = 0; i < j; i++)
        {
            key += to_string(i + 1);
        }
        string perm;
        string temp;
        string_permutation(cipher_strings, key, perm, ans);
    }
    return ans;
}
```

In the function first we initialized a string key with a sequential numeric pattern based on the key size which is being incremented in a for loop from range 1 to 9. It then calls the string_permutation function, which recursively generates permutations of the key and decrypts the cipher strings using each permutation.

If a permutation is found that satisfies the specified hash conditions for all cipher strings, it is stored in the ans variable. At last, the function returns the discovered key, representing a successful outcome of the brute-force attack.

String permutation Function

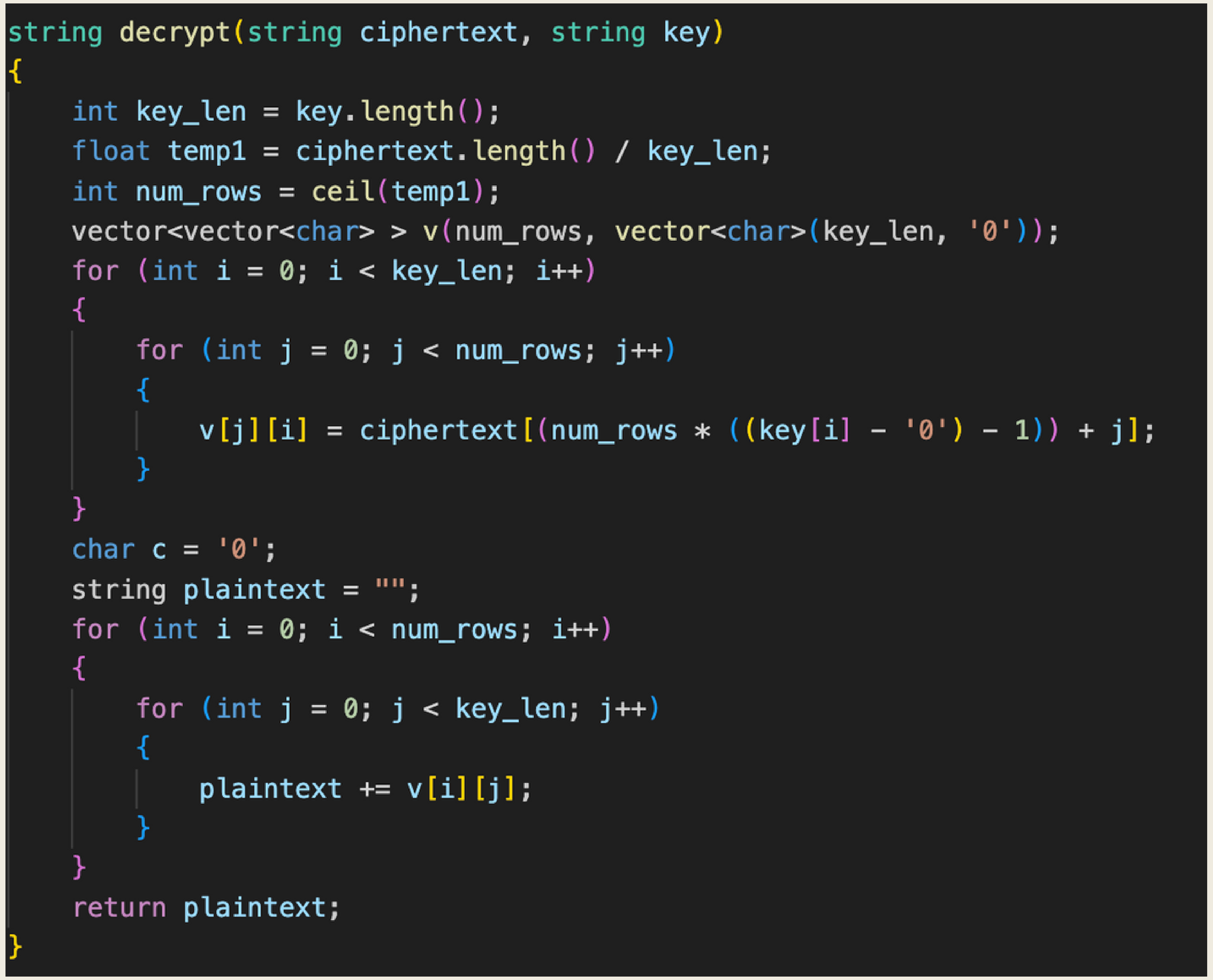


```cpp
void string_permutation(vector<string> cipher_strings, string &orig, string &perm, string &ans)
{
    if (orig.empty())
    {
        char ch = '-';
        string temp_plaintext = decrypt(cipher_strings[0], perm);
        removechar(temp_plaintext, ch);
        if (hashfunction(temp_plaintext.substr(0, temp_plaintext.length() - 8)) == temp_plaintext.substr(temp_plaintext.length() - 8, 8))
        {
            string temp_plaintext2 = decrypt(cipher_strings[1], perm);
            removechar(temp_plaintext2, ch);
            if (hashfunction(temp_plaintext2.substr(0, temp_plaintext2.length() - 8)) == temp_plaintext2.substr(temp_plaintext2.length() - 8, 8))
            {
                string temp_plaintext3 = decrypt(cipher_strings[2], perm);
                removechar(temp_plaintext3, ch);
                if (hashfunction(temp_plaintext3.substr(0, temp_plaintext3.length() - 8)) == temp_plaintext3.substr(temp_plaintext3.length() - 8, 8))
                {
                    string temp_plaintext4 = decrypt(cipher_strings[3], perm);
                    removechar(temp_plaintext4, ch);
                    if (hashfunction(temp_plaintext4.substr(0, temp_plaintext4.length() - 8)) == temp_plaintext4.substr(temp_plaintext4.length() - 8, 8))
                    {
                        string temp_plaintext5 = decrypt(cipher_strings[4], perm);
                        removechar(temp_plaintext5, ch);
                        if (hashfunction(temp_plaintext5.substr(0, temp_plaintext5.length() - 8)) == temp_plaintext5.substr(temp_plaintext5.length() - 8, 8))
                        {
                            ans = perm;
                        }
                    }
                }
            }
        }
        return;
    }

    for (int i = 0; i < orig.size(); ++i)
    {
        string orig2 = orig;
        orig2.erase(i, 1);
        string perm2 = perm;
        perm2 += orig.at(i);
        string_permutation(cipher_strings, orig2, perm2, ans);
    }
}
```

This function recursively generates permutations of a given random key and checks if any permutation satisfies the condition based on the results of the decrypt and hashfunction functions. The function repeatedly removes a character from the key, appends it to the permutation string, and recursively calls itself with the updated keys. When the key becomes empty, the function decrypts each cipher string using the generated permutation and removes a specific character '-' from the resulting plaintext using the function removechar().

It then checks if the hash of the modified plaintext matches the last 8 characters of the plaintext. If this condition holds for all the 5 cipher strings, the permutation is considered a valid solution, and it is stored in the ans variable.

Decryption Function



```cpp
string decrypt(string ciphertext, string key)
{
    int key_len = key.length();
    float temp1 = ciphertext.length() / key_len;
    int num_rows = ceil(temp1);
    vector<vector<char> > v(num_rows, vector<char>(key_len, '0'));
    for (int i = 0; i < key_len; i++)
    {
        for (int j = 0; j < num_rows; j++)
        {
            v[j][i] = ciphertext[(num_rows * ((key[i] - '0') - 1)) + j];
        }
    }
    char c = '0';
    string plaintext = "";
    for (int i = 0; i < num_rows; i++)
    {
        for (int j = 0; j < key_len; j++)
        {
            plaintext += v[i][j];
        }
    }
    return plaintext;
}
```

In this function, the reverse of the transposition cipher is implemented to decrypt the ciphertext. In light of the length of the key and the ciphertext, the capability decides the number of columns that are required for decryption.

It then constructs a 2D vector v and populates it by rearranging the characters in the ciphertext based on the order specified by the key. The final step involves reconstructing the original plaintext by reading the vector column-wise. At last, the decrypted plaintext is returned.

Feb 01, 2024

Thank you !

Name	Sarvagya Kaushik
Roll No.	2021350
Branch	CSD
E-mail	Sarvagya21350@iiitd.ac.in

Name	Kunal Sharma
Roll No.	2021331
Branch	CSD
E-mail	Kunal21331@iiitd.ac.in