資訊安全導論 HW1

B10815044 謝鈞曜 加密

B10815057 廖聖郝 解密

加密

Caesar

先把 key 轉成整數，再將 char 根據 key 做位移

```
void Caesar(char* plaintext, char* key) {

    int shift = atoi(key);

    for (int i = 0; i < strlen(plaintext); i++) {
        if ((plaintext[i] + shift) >= 123) {
            plaintext[i] = toupper((plaintext[i] + shift) % 123 + 'a');
        }
        else if ((plaintext[i] + shift) < 97) {
            plaintext[i] = toupper((plaintext[i] + shift) + 26);
        }
        else {
            plaintext[i] = toupper((plaintext[i] + shift) % 123);
        }
    }
    cout << plaintext;
}
```

Playfair

先將 Key Table 建出

```
// Generate Key Table
int row=0, col=0;
for (int i = 0; i < strlen(key); i++) {
    if (alphabet[key[i]-'a'] == false) {
        alphabet[key[i]-'a'] = true;
        table[row][col] = key[i];
        col++;
        if (col == 5) {
            col = 0;
            row++;
        }
    }
}
for (int i = 0; i < 26; i++) {
    if (alphabet[i] == false) {
        alphabet[i] = true;
        table[row][col] = 'a' + i;
        col++;
        if (col == 5) {
            col = 0;
            row++;
        }
    }
}
```

對明文進行基本處理

```
for (int i = 0; i < input.size()-1; i++) {
    if (input[i] == input[i + 1]) {
        input.insert(input.begin() + i + 1, 'x');
    }
    i++;
}
if (input.size() % 2) {
    input.push_back('x');
}
```

根據字母相對的位置，做不同的加密，最後輸出結果

```
Pair a, b;
for (int i = 0; i < input.size(); i++) {
    for (int j = 0; j < 5; j++) {
        for (int k = 0; k < 5; k++) {
            if (input[i] == table[j][k]) {
                a.i = j;
                a.j = k;
            }
        }
    }
    for (int j = 0; j < 5; j++) {
        for (int k = 0; k < 5; k++) {
            if (input[i+1] == table[j][k]) {
                b.i = j;
                b.j = k;
            }
        }
    }
    i++;
    // Same Row
    if (a.i == b.i) {
        code.push_back(table[a.i][(a.j + 1) % 5]);
        code.push_back(table[a.i][(b.j + 1) % 5]);
    }
    // Same Col
    else if (a.j == b.j) {
        code.push_back(table[(a.i + 1) % 5][a.j]);
        code.push_back(table[(b.i + 1) % 5][b.j]);
    }
    // Both different
    else {
        code.push_back(table[a.i][b.j]);
        code.push_back(table[b.i][a.j]);
    }
}
for (int i = 0; i < code.size(); i++) {
    code[i] = toupper(code[i]);
    cout<<code[i];
}
```

Vernam

先生成 Autokey，生成完畢之後再將 key 與 plaintext 做 XOR 運算

```
void Vernam(char* plaintext, char* key) {
    vector<int>keystream;
    for (int i = 0; i < strlen(key); i++) {
        keystream.push_back(key[i] - 'a');
    }
    for (int i = 0; i < strlen(plaintext)-strlen(key); i++) {
        keystream.push_back(plaintext[i] - 'a');
    }
    for (int i = 0; i < keystream.size(); i++) {
        plaintext[i] = toupper(((plaintext[i]-'a') ^ keystream[i]) +
'a');
    }
    cout << plaintext;
}
```

RailFence

一開始先分配空間，並全部都用空白初始化，之後開始將明文填入陣列中，一開始會向下填，碰到 row 的最底層之後會開始反轉向上填，最後填完之後將陣列不是空格的元素輸出即是密文

```
void RailFence(char* plaintext, char* key) {

    int row = key[0] - '0';
    int col = strlen(plaintext);
    vector<vector<char> > chipertext(row, vector<char>(col, ' '));
    int current_row = 0;
    bool increase = true;
    for (int i = 0; i < col; i++) {
        chipertext[current_row][i] = plaintext[i];
        if (current_row == row - 1) {
            increase = false;
        }
        else if(current_row ==0){
            increase = true;
        }
        if (increase) {
            current_row++;
        }
        else {
            current_row--;
        }
    }
    for (int j = 0; j < row; j++) {
        for (int k = 0; k < col; k++) {
            if (chipertext[j][k] != ' ') {
                cout << (char)toupper(chipertext[j][k]);
            }
        }
    }
}
```

RowTransition

根據 key 的長度來決定 col、row 的大小，依據 key 的順序輸出 row，但可以

透過一定規律產生密文，就不需要配置儲存空間了

```
void RowTransition(char* plaintext, char* key) {
    int col = strlen(key);
    int row = strlen(plaintext) / col;
    if (strlen(plaintext) % col != 0) {
        row += 1;
    }
    int count = 1;
    while (count != col+1) {
        for (int i = 0; i < strlen(key); i++) {
            if (count == key[i] - '0') {
                for (int j = 0; j < row; j++) {
                    if (j * col + i >= strlen(plaintext)) {
                        continue;
                    }
                    else {
                        cout << (char)toupper(plaintext[j * col + i]);
                    }
                }
                count++;
            }
        }
    }
}
```

解密

Caesar

先將 key 值 mod 26，然後將密文中的每個值減去此值，若超出 A~Z 的範圍，

就加或減 26，使其回到 A~Z 之範圍

```
void Caesar(string ciphertext, string key) {
    int shift = stoi(key);
    shift %= 26;
    string plaintext;
    for (int i = 0; i < ciphertext.length(); i++) {
        int alpha_pos = ciphertext[i] - 'A';
        int minus_pos = alpha_pos - shift;
        if (minus_pos < 0) {
            plaintext.push_back(tolower(minus_pos + 26 + 'A'));
        }
        else if (minus_pos >= 26) {
            plaintext.push_back(tolower(minus_pos - 26 + 'A'));
        }
        else { //0~25
            plaintext.push_back(tolower(minus_pos + 'A'));
        }
    }
    cout << plaintext << endl;
}
```

Playfair

利用 bool array 紀錄每個字元有沒有出現過，並同時將重複字元去除，產生 5x5 table

```
char table[5][5];
bool encounter[26] = {0};
string no_repeat_key;
string remain_char;
int not_in_char = 0;
int J_index = 'j' - 'a';
for (int i = 0; i < key.length(); i++) {
    if (key[i] == 'j') key[i] = 'i';

    if (encounter[key[i]-'a']  == false) {
        encounter[key[i] - 'a'] = true;
        no_repeat_key.push_back(key[i]);
    }
}
for (int i = 0; i < 26; i++) {
    if (encounter[i] == false) {
        if (i != J_index) remain_char.push_back(i + 'a');
    }
}
key = no_repeat_key;
int key_index = 0;
int remain_index = 0;
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
        if (key_index < key.length()) {
            table[i][j] = key[key_index];
            key_index++;
        }
        else {
            table[i][j] = remain_char[remain_index];
            remain_index++;
        }
    }
}
```

將輸入密文分成兩個一組

```
    vector<pair<char,char>> pairs;
    for (int i = 0; i < ciphertext.length(); i += 2) {
        pairs.push_back(make_pair(ciphertext[i], ciphertext[i + 1]));
    }
```

對每個組去查表，進行解密

```
    auto find_pos = [&](char in) -> pair<int, int> {
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                if (tolower(in) == table[i][j]) {
                    return make_pair(i,j);
                }
            }
        }
    };
    string plaintext;
    for (int i = 0; i < pairs.size(); i++) {
        pair<int, int> pos1 = find_pos(pairs[i].first);
        pair<int, int> pos2 = find_pos(pairs[i].second);
        if (pos1.first == pos2.first) {
            plaintext.push_back(tolower(table[pos1.first][pos1.second !=
0 ? pos1.second - 1 : 4]));
            plaintext.push_back(tolower(table[pos2.first][pos2.second !=
0 ? pos2.second - 1 : 4]));
        }
        else if (pos1.second == pos2.second) {
            plaintext.push_back(tolower(table[pos1.first != 0 ?
pos1.first - 1 : 4][pos1.second]));
            plaintext.push_back(tolower(table[pos2.first != 0 ?
pos2.first - 1 : 4][pos2.second]));
        }
        else {
            plaintext.push_back(tolower(table[pos1.first][pos2.second]))
;
            plaintext.push_back(tolower(table[pos2.first][pos1.second]))
;
        }
    }
```

Vernam(autokey)

掃過每個密文中的字元，若 key 的長度小於密文長度，則每次解密後的字元，

須補到 key 最後面

```
void Vernam(string ciphertext, string key) {
    string plaintext;
    for (int i = 0; i < key.length(); i++) {
        plaintext.push_back(((ciphertext[i]-'A') ^ (key[i]-'a')) + 'a');
        if (ciphertext.length() > key.length()) {
            key.push_back(plaintext.back());
        }
    }
    cout << plaintext << endl;
}
```

RailFence

計算單個 fence 有多少個 element (fence_len)，利用 fence_len 與 key 值 (rail_count) 來計算密文 index 對應到的明文 index，一共分為三種情形，頂端、底部、中間進行處理

```
void RailFence(string ciphertext, string key) {
    int rail_count = stoi(key);
    int fence_len = 2 + (rail_count - 2) * 2; //V
    int index = 0;
    string plaintext;
    plaintext.resize(ciphertext.length());
    for (int i = 0; i < rail_count; i++) {
        if (i == 0) {
            for (int target_index = 0; true;  target_index+=fence_len) {
                if (target_index >= plaintext.length()) break;
                plaintext[target_index] = tolower(ciphertext[index++]);
            }
        }
        else if (i == rail_count - 1) {
            for (int target_index = i; true; target_index += fence_len)
{
                if (target_index >= plaintext.length()) break;
                plaintext[target_index] = tolower(ciphertext[index++]);
            }
        }
        else {
            for (int target_index1 = i, target_index2 = fence_len - i;
true;target_index1 += fence_len, target_index2 += fence_len) {
                if (target_index1 >= plaintext.length()) break;
                plaintext[target_index1] = tolower(ciphertext[index++]);
                if (target_index2 >= plaintext.length()) break;
                plaintext[target_index2] = tolower(ciphertext[index++]);
            }
        }
    }
    cout << plaintext << endl;
}
```

RowTransition

處理密文不足以形成完整長方形的狀況，會將每個直排的 row 數紀錄於

col_to_rows

```
    int cols = key.length();
    int max_row = ciphertext.length() / cols;
    int remain_count = ciphertext.length() % cols;
    vector<int> col_to_rows;
    vector<int> order_to_col_index;
    string plaintext;
    plaintext.resize(ciphertext.size());
    for (int i = 0; i < cols; i++) {
        if (remain_count != 0) {
            if (i >= remain_count) {
                col_to_rows.push_back(max_row);
            }
            else {
                col_to_rows.push_back(max_row + 1);
            }
        }
        else {
            col_to_rows.push_back(max_row);
        }
    }
```

將每個每個數字字元於 key 中對應的 index 紀錄於 order_to_col_index

```
    for (int i = '1'; i <= (cols + '0'); i++) {
        for (int j = 0; j < key.length(); j++) {
            if (i == key[j]) {
                order_to_col_index.push_back(j);
                break;
            }
        }
    }
```

依序將密文中的每個字元放到其對應於明文的位置，完成解密

```
    int index = 0;
    for (int i = 0; i < key.length(); i++) { //loop '1' to end
        int col = order_to_col_index[i];
        int rows = col_to_rows[col];
        for (int j = 0; j < rows;j++) {
            plaintext[j * cols + col] = tolower(ciphertext[index++]);
        }
    }
```