**INFORMATION AND NETWORK SECURITY COURSE**

**Classic Encryption Project**



**Member of Group:**

Dely Teja Mukti 05111540000003

Yoga Samudra 05111540000080

M. Ilham Febriyanto 05111540000083

**Lecturer:**

Henning Titi Ciptaningtyas, S.Kom, M.Kom

**DEPARTMENT OF INFORMATICS**

**FACULTY OF INFORMATION TECHNOLOGY**

**INSTITUT TEKNOLOGI SEPULUH NOPEMBER**

**2018**

## 1. CAESAR CIPHER

## DEFINITION

Caesar Cipher is the simplest method of message encryption found by Julius Caesar. It make use of substitution cipher which involves replacing each letter of the alphabet with the letter standing three places further down the alphabet. For example, shifting the alphabet 5 letter next to it, so letter A become F, B become G, C become H, and etc.

As unreadable as the resulting ciphertext may appear, the Caesar Cipher is one of the weakest forms of encryption one can employ for the following reasons:

* The key space is very small. Using a [brute force attack](https://learncryptography.com/attack-vectors/brute-force-attack) method, one could easily try all (25) possible combinations to decrypt the message without initially knowing the key.
* The structure of the original plaintext remains intact. This makes the encryption method vulnerable to [frequency analysis](https://learncryptography.com/attack-vectors/frequency-analysis) - by looking at h ow often certain characters or sequences of characters appear, one can discover patterns and potentially discover the key without having to perform a full brute force search.

## DESCRIPTION

In this project, we use Javascript to make the encryption and decryption code and combine it with web programming. The input type is text form (read as string) in html and the output type is also in text form.

## SOURCE CODE

### encryptor.js

File Name : encryptor.js

Function : DoCaesarEncrypt

Input : text (string), shf (integer)

Output : text (string)

Source code

ABC = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

x = "";

for (i=0; i<text.length; i++)

{

char = text.charAt(i).toUpperCase();

pos = ABC.indexOf(char);

if (pos>=0) x += ABC.charAt((pos+shf)%26);

else x += char;

}

return x;

File Name : encryptor.js

Function : DoCaesarDecrypt

Input : text (string), shf (integer)

Output : text (string)

Source code

return DoCaesarEncrypt(text,-shf);

## IMPLEMENTATION

* Write step by step tutorial in details for using your application (data input or button should be clicked).
* Capture each GUI of your steps. Each GUI figure should have caption and each figure should be referred by sentence as seen on Figure 1.



Figure 1. Application Input

## EXPERIMENTAL RESULT

Write the experiment result from at least 4 inputs:

1. The first is **exactly the same** as the example in the E book
2. The next is up to you

You can modify this table if necessary.

## 2. MONOALPHABETIC CIPHER

## DEFINITION

Monoalphabetic Cipher or usually called as Substitution Cipher is an encryption method which can be demonstrated by writing out the alphabet in some order to represent the substitution. The cipher alphabet may be shifted or reversed (creating the [Caesar](https://en.wikipedia.org/wiki/Caesar_cipher) and [Atbash](https://en.wikipedia.org/wiki/Atbash) ciphers, respectively) or scrambled in a more complex fashion, in which case it is called a **mixed alphabet** or **deranged alphabet**. Traditionally, mixed alphabets may be created by first writing out a keyword, removing repeated letters in it, then writing all the remaining letters in the alphabet in the usual order.

## DESCRIPTION

In this project, we use Javascript to make the encryption and decryption code and combine it with web programming. The input type is text form (read as string) in html and the output type is also in text form.

## SOURCE CODE

### encryptor.js

File Name : encryptor.js

Function : Encrypt

Input : text (string), key (string)

Output : text (string)

Source code

plaintext = text.toLowerCase();

ekey = key.toLowerCase().replace(/[^a-z]/g,"");

if(plaintext.length < 1){

alert("please enter some plaintext (letters and numbers only)"); return;

}

if(ekey.length != 26){

alert("key must be 26 characters in length");

return;

}

ciphertext = ""; var re = /[a-z]/;

for(i=0; i<plaintext.length; i++){

if(re.test(plaintext.charAt(i))) ciphertext += ekey.charAt(plaintext.charCodeAt(i)-97);

else ciphertext += plaintext.charAt(i);

}

return ciphertext.toUpperCase();

File Name : encryptor.js

Function : Decrypt

Input : text (string), key (string)

Output : text (string)

Source code

ciphertext = text.toLowerCase();

ekey = key.toLowerCase().replace(/[^a-z]/g, "");

if(ciphertext.length < 1){ alert("please enter some ciphertext (letters only)"); return; }

if(ekey.length != 26){ alert("key must be 26 characters in length"); return; }

plaintext = ""; var re = /[a-z]/;

for(i=0; i<ciphertext.length; i++){

if(re.test(ciphertext.charAt(i))) plaintext += String.fromCharCode(ekey.indexOf(ciphertext.charAt(i))+97);

else plaintext += ciphertext.charAt(i);

}

return plaintext.toUpperCase();

## 3. PLAYFAIR CIPHER

## DEFINITION

PlayFair Cipher is the best-known multiple-letter encryption cipher which treats digrams in the plaintext as single units and translates these units into ciphertext digrams. The PlayFair algorithm is based on the use of a 5 \* 5 matrix of letters constructed using a keyword that will act as the key for encrypting your plaintext. Each of the 25 letters must be unique and one letter of the alphabet (usually Q) is omitted from the table (as there are 25 spots and 26 letters in the alphabet).

## DESCRIPTION

In this project, we use javascript to make the encryption and decryption code and combine it with web programming. The input type is text form (read as string) in html and the output type is also in text form.

## SOURCE CODE

### PlayfairCipher.js

File Name : encryptor.js

Function : setKey

Input : key (string)

Output : grid (two dimensional array)

Source code

if (key) {

// create grid from key

alphabet = ['abcdefghiklmnopqrstuvwxyz'];

sanitizedKey = key.toLowerCase().replace(/j/g, 'i').replace(/[^a-z]/g, '');

keyGrid = [...new Set(`${sanitizedKey}${alphabet}`)];

grid = [];

for (let i = 0; i < keyGrid.length; i += 5) {

grid.push(keyGrid.slice(i, i + 5));

}

} else {

grid = [

['a', 'b', 'c', 'd', 'e'],

['f', 'g', 'h', 'i', 'k'],

['l', 'm', 'n', 'o', 'p'],

['q', 'r', 's', 't', 'u'],

['v', 'w', 'x', 'y', 'z']

];

}

File Name : encryptor.js

Function : preProcess

Input : input (string), decrypt (boolean)

Output : ciphertext (string)

Source code

// split into duples, fixing double-letters (hello => he lx lo) and padding

text = input.toLowerCase().replace(/[^a-z]/g, '').replace(/j/g, 'i').split('').filter(x => x !== ' ');

duples = [];

for (let i = 0; i < text.length; i += 2) {

currentDuple = text.slice(i, i + 2);

if (!decrypt && currentDuple.length !== 2) {

currentDuple.push('x');

duples.push(currentDuple);

}else if (!decrypt && currentDuple[0] === currentDuple[1]) {

text.splice(i + 1, 0, 'x');

duples.push(text.slice(i, i + 2));

}else {

duples.push(currentDuple);

}

}

// find row and column for each letter in duple

const coordinates = [];

duples.forEach((duple) => {

coordinates.push(duple.map((letter) => {

let col;

const row = grid.findIndex(row => {

const rowIdx = row.findIndex(x => x === letter);

if (rowIdx >= 0) {

col = rowIdx;

return true;

}

return false;

});

return [row, col];

}));

});

return coordinates;

File Name : encryptor.js

Function : doPlayfair

Input : input (string), decrypt (boolean)

Output : ciphertext (string)

Source code

if (!grid) return 'First set the key!';

if (input && decrypt && input.length % 2 !== 0) input += 'x';

const coordinates = preProcess(input, decrypt);

// set modifiers to respond appropriately based on decrypt switch

const modifier = decrypt ? -1 : 1;

const wall = decrypt ? 0 : 4;

const phase = decrypt ? 4 : -4;

const processedLocs = [];

coordinates.forEach((loc) => {

// loc: [ [ firstLetterR, firstLetterC ], [ secondLetterR, secondLetterC ] ]

// modified: [ [ newFirstLetterR, newFirstLetterC ], [ newSecondLetterR, newSecondLetter R ] ]

let modifiedLoc = [];

// handle letters on the same row

if (loc[0][0] === loc[1][0]) {

// increment/decrement the column

modifiedLoc[0] = loc[0][1] === wall ? [loc[0][0], wall + phase] : [loc[0][0], loc[0][1] + modifier];

modifiedLoc[1] = loc[1][1] === wall ? [loc[1][0], wall + phase] : [loc[1][0], loc[1][1] + modifier];

return processedLocs.push(modifiedLoc);

}

// handle letters in the same column

if (loc[0][1] === loc[1][1]) {

// increment/decrement the row

modifiedLoc[0] = loc[0][0] === wall ? [wall + phase, loc[0][1]] : [loc[0][0] + modifier, loc[0][1]];

modifiedLoc[1] = loc[1][0] === wall ? [wall + phase, loc[1][1]] : [loc[1][0] + modifier, loc[1][1]];

return processedLocs.push(modifiedLoc);

}

// handle different rows, different columns

modifiedLoc[0] = [loc[0][0], loc[1][1]];

modifiedLoc[1] = [loc[1][0], loc[0][1]];

processedLocs.push(modifiedLoc);

});

// translate coordinates into ciphertext

const processedText = processedLocs

.map((loc) => [grid[loc[0][0]][loc[0][1]], grid[loc[1][0]][loc[1][1]]].join(''))

.join('');

return processedText.toUpperCase();

## 4. POLYALPHABETIC CIPHER

## DEFINITION

Polyalphabetic cipher is any [cipher](https://en.wikipedia.org/wiki/Cipher) based on [substitution](https://en.wikipedia.org/wiki/Substitution_cipher), using multiple substitution alphabets. One of it’s well known example is Vigenere Cipher, which was adapted as a twist on the standard Caesar cipher to reduce the effectiveness of performing frequency analysis on the ciphertext. The cipher accomplishes this using uses a text string (for example, a word) as a key, which is then used for doing a number of alphabet shifts on the plaintext. Similar to the [Caesar Cipher](https://learncryptography.com/classical-encryption/vigenere-cipher), but instead of performing a single alphabet shift across the entire plaintext, the Vigenère cipher uses a key to determine several different shift amounts across the entirety of the message. But another popular method of this cipher is The Enigma machine which was more complex than any other method, but still use polyalphabetic cipher as it’s basic fundamental.

## DESCRIPTION

In this project, we use javascript to make the encryption and decryption code and combine it with web programming. The input type is text form (read as string) in html and the output type is also in text form.

## SOURCE CODE

### PolyalphabeticCipher.js

File Name : encryptor.js

Function : doCrypt

Input : vkey (string), vtext (string), isDecrypt (boolean)

Output : ciphertext (string)

Source code

var key = filterKey(vkey);

if (key.length == 0) {

alert("Key has no letters");

return;

}

if (isDecrypt) {

for (var i = 0; i < key.length; i++)

key[i] = (26 - key[i]) % 26;

}

var textElem = vtext;

return textElem = crypt(textElem, key);

File Name : encryptor.js

Function : crypt

Input : input (string), key (string)

Output : output (string)

Source code

var output = "";

for (var i = 0, j = 0; i < input.length; i++) {

var c = input.charCodeAt(i);

if (isUppercase(c)) {

output += String.fromCharCode((c - 65 + key[j % key.length]) % 26 + 65);

j++;

} else if (isLowercase(c)) {

output += String.fromCharCode((c - 97 + key[j % key.length]) % 26 + 97);

j++;

} else {

output += input.charAt(i);

}

}

return output;

File Name : encryptor.js

Function : filterKey

Input : key (string)

Output : result (string)

Source code

var result = [];

for (var i = 0; i < key.length; i++) {

var c = key.charCodeAt(i);

if (isLetter(c))

result.push((c - 65) % 32);

}

return result;

## 5. RAIL FENCE CIPHER

## DEFINITION

Rail Fence Cipher is one example of a transposition cipher. This encryption method use positioning as it’s basic encryption in which the plaintext is written down as a sequence of diagonals and then read off as a sequence of rows. For example, if we have 3 "rails" and a message of 'WE ARE DISCOVERED. FLEE AT ONCE', the cipherer writes out:

W . . . E . . . C . . . R . . . L . . . T . . . E

. E . R . D . S . O . E . E . F . E . A . O . C .

. . A . . . I . . . V . . . D . . . E . . . N . .

Then reads off to get the ciphertext:

WECRLTEERDSOEEFEAOCAIVDEN

## DESCRIPTION

In this project, we use javascript to make the encryption and decryption code and combine it with web programming. The input type is text form (read as string) in html and the output type is also in text form.

## SOURCE CODE

### RailFenceCipher.js

File Name : encryptor.js

Function : encode

Input : message (string)

Output : ciphertext (string)

Source code

rail1=[];

rail2=[];

rail3=[];

message = message.toUpperCase().split(' ').join('');

for( var i = 0; i < message.length; i++ ){

if(i % 4 === 0 ){

rail1.push( message[i] );

}

else if ( i % 2 === 1 ){

rail2.push( message[i] );

}

else{

rail3.push( message[i] );

}

}

rail = rail1.join('') + rail2.join('') + rail3.join('');

return rail;

File Name : encryptor.js

Function : decode

Input : message (string)

Output : plaintext (string)

Source code

ctr4 = 0;

ctr3 = 0;

ctr2 = 0;

ctr1 = 0;

rail4 = [];

rail3 = [];

rail2 = [];

rail1 = [];

rail = [];

xmessage = message.toUpperCase().split(' ').join('');

for(i = 0; i < message.length; i++){

if(i % 4 === 0 ) ctr4++;

else if ( i % 4 === 3 ) ctr3++;

else if ( i % 4 === 2 ) ctr2++;

else ctr1++;

}

while(ctr4--)

{

rail4.push(xmessage[0]);

xmessage=xmessage.substring(1);

}

rail4=rail4.join('');

ctr0 = ctr1+ctr3;

for(i=0; i<ctr0; i++){

if(i%2==1)

{

rail3.push(xmessage[0]);

xmessage=xmessage.substring(1);

}

else

{

rail1.push(xmessage[0]);

xmessage=xmessage.substring(1);

}

}

rail1=rail1.join('');

rail3=rail3.join('');

while(ctr2--)

{

rail2.push(xmessage[0]);

xmessage=xmessage.substring(1);

}

rail2=rail2.join('');

for(i = 0; i < message.length; i++){

if(i % 4 === 0 ){

rail.push(rail4[0]);

rail4=rail4.substring(1);

}

else if ( i % 4 === 1 ){

rail.push(rail1[0]);

rail1=rail1.substring(1);

}

else if ( i % 4 === 3 ){

rail.push(rail3[0]);

rail3=rail3.substring(1);

}

else{

rail.push(rail2[0]);

rail2=rail2.substring(1);

}

}

return rail.join('');

## ALGORITHM DESCRIPTION

Give short description about the algorithm such as:

1. Definition.
2. How it works.
3. Strength / Complexity.
4. Weakness.

For this section, please **resume** from the **E book** and any other relevant sources.

In the coding, you should use the static parameter **exactly the same** with the E book.

## APPLICATION DESCRIPTION

Give short description about your application

1. Programming Language.
2. Library used in the source code (if any).
3. Input and output data / file type.

You can add more details if necessary.

## SOURCE CODE

### encryptor.js

File Name : encryptor.js

Function : Explain the function of this source code file

Input : If any

Output : If any

Source code

Write the source code using **Verdana** **10** **pt**.

Give comments to explain your code.

## IMPLEMENTATION

* Write step by step tutorial in details for using your application (data input or button should be clicked).
* Capture each GUI of your steps. Each GUI figure should have caption and each figure should be referred by sentence as seen on Figure 1.



Figure 1. Application Input

## EXPERIMENTAL RESULT

Write the experiment result from at least 4 inputs:

1. The first is **exactly the same** as the example in the E book
2. The next is up to you

You can modify this table if necessary.

Table 1. Caesar Chiper Experimental Result

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Input File / Data** | **Key** | **Output File / Data** | **Execution**  **(ms)** |
|
| 1. | meet me after the toga party | C = E(3, p) = (p + 3) mod 26 | PHHW PH DIWHU WKH WRJD SDUWB | 0.732 |
| 2. | Information And Network Security Course | C = E(5, p) = (p + 5) mod 26 |  |  |
| 3. | this project is very easy | C = E(-4, p) = (p - 4) mod 26 |  |  |
| 4. | Fun web coding |  |  |  |

## REFERENCE(S)

Write reference of your library, source code, your image(s), etc using **Reference-Insert Bibliography.**