

Tugas Kecil 1 IF4020 Kriptografi

Penyusunan Kriptografi Klasik dengan GUI (Crypt GUI)



Nama: Leonard Matheus

NIM: 13519215

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG**

2021

Source Code Program C++

1. Vigenere Cipher Standard - VCS.cpp

```
1  #include "VCS.h"
2  #include <iostream>
3  #include <string>
4
5  using namespace std;
6
7  VCS::VCS(){
8      plain = "null";
9      kunci = "null";
10     cipher = "null";
11     file = "null";
12 }
13
14 void VCS::VCS_Encripsi()
15 {
16     string plain2 = getPlain();
17     //Remove karakter non-alphabetic
18     string plain = removeKarakterLain(plain2);
19     //Menyelaraskan kunci dengan panjang Plain text
20     string kunci = generateKunci(plain, this->getKunci());
21     string tampungan2 = "";
22     for (int i = 0; i < (int) plain.size(); i++) {
23         int p = charToInt(plain[i], true);
24         int k = charToInt(kunci[i], true);
25         int c = (p + k) % 26;
26         char cnya = intToChar(c, true);
27         tampungan2.push_back(cnya);
28     }
29     string temp = filterOutput(tampungan2, 5);
30     setCipher(temp);
31 }
32
33 void VCS::VCS_Encripsi_File()
34 {
35     string cipher = getCipher();
36     setFile(cipher);
37 }
38
39 void VCS::VCS_Dekripsi() {
40     string cipher2 = getCipher();
41     //Remove karakter non-alphabetic
42     string cipher = removeKarakterLain(cipher2);
43     //Menyelaraskan kunci dengan panjang Plain text
44     string kunci = generateKunci(cipher, this->getKunci());
45     string tampungan2 = "";
46     for (int i = 0; i < (int) cipher.size(); i++) {
47         int c = charToInt(cipher[i], true);
48         int k = charToInt(kunci[i], true);
49         if (c - k < 0) {
50             c += 26;
51         }
52         int p = (c - k) % 26;
53         char pnya = intToChar(p, true);
54         tampungan2.push_back(pnya);
55     }
56     setPlain(tampungan2);
57 }
58
59 void VCS::VCS_Dekripsi_File() {
60     string plain = getPlain();
61     setFile(plain);
62 }
63
64
65 string VCS::generateKunci(string _plain, string _kunci)
66 {
67     string tampungan;
68     int jumlahPlain = (int) _plain.size();
69     int jumlahKunci = (int) _kunci.size();
70     int faktor = ceil(jumlahPlain / jumlahKunci);
71     for (int i = 0; i <= faktor; i++) {
72         tampungan += kunci;
73     }
74     tampungan.resize(jumlahPlain);
75     return tampungan;
76 }
77
78 string VCS::getPlain(){return this->plain;}
79 string VCS::getKunci() { return this->kunci; }
80 string VCS::getCipher() { return this->cipher; }
81 string VCS::getFile() { return this->file; }
82
83 void VCS::setPlain(string _plain){this->plain = _plain;}
84 void VCS::setKunci(string _kunci) { this->kunci = removeKarakterLain(_kunci); }
85 void VCS::setCipher(string _cipher) { this->cipher = _cipher; }
86 void VCS::setFile(string _file) { this->file = _file; }
```

2. Full Vigenere Cipher - FCS.cpp

Pergeseran key berdasarkan urutan alfabet yang telah di hard-code sehingga tinggal dihitung modulusnya. Hardcode key dapat ditemukan di backend.cpp

```

1  #include "FVC.h"
2  #include <iostream>
3  #include <string>
4
5  using namespace std;
6
7  FVC::FVC() {
8      plain = "null";
9      kunci = "null";
10     cipher = "null";
11     file = "null";
12 }
13
14 void FVC::FVC_Encripsi()
15 {
16     string plain2 = getPlain();
17     //Remove karakter non-alphabetic
18     string plain = removeKarakterLain(plain2);
19     //Menyelaraskan kunci dengan panjang Plain text
20     string kunci = generateKunci(plain, this->getKunci());
21     string tampungan2 = "";
22     for (int i = 0; i < (int)plain.size(); i++) {
23         int p = charToInt(plain[i], false);
24         int k = charToInt(kunci[i], false);
25         int c = (p + k) % 26;
26         char cnnya = intToChar(c, false);
27         tampungan2.push_back(cnnya);
28     }
29     string temp = filterOutput(tampungan2, 5);
30     setCipher(temp);
31 }
32
33 void FVC::FVC_Encripsi_File()
34 {
35     string cipher = getCipher();
36     setFile(cipher);
37 }
38
39 void FVC::FVC_Dekripsi() {
40     string cipher2 = getCipher();
41     //Remove karakter non-alphabetic
42     string cipher = removeKarakterLain(cipher2);
43     //Menyelaraskan kunci dengan panjang Plain text
44     string kunci = generateKunci(cipher, this->getKunci());
45     string tampungan2 = "";
46     for (int i = 0; i < (int)cipher.size(); i++) {
47         int c = charToInt(cipher[i], false);
48         int k = charToInt(kunci[i], false);
49         if (c - k < 0) {
50             c += 26;
51         }
52         int p = (c - k) % 26;
53         //setCipherFile(static_cast<string>(to_string(p)));
54         char pnya = intToChar(p, false);
55         tampungan2.push_back(pnya);
56     }
57     setPlain(tampungan2);
58 }
59
60 void FVC::FVC_Dekripsi_File() {
61     string plain = getPlain();
62     setFile(plain);
63 }
64
65 string FVC::generateKunci(string _plain, string _kunci)
66 {
67     string tampungan;
68     int jumlahPlain = (int)_plain.size();
69     int jumlahKunci = (int)_kunci.size();
70     int faktor = ceil(jumlahPlain / jumlahKunci);
71     for (int i = 0; i <= faktor; i++) {
72         tampungan += kunci;
73     }
74     tampungan.resize(jumlahPlain);
75     return tampungan;
76 }
77
78 string FVC::getPlain() { return this->plain; }
79 string FVC::getKunci() { return this->kunci; }
80 string FVC::getCipher() { return this->cipher; }
81 string FVC::getFile() { return this->file; }
82
83 void FVC::setPlain(string _plain) { this->plain = _plain; }
84 void FVC::setKunci(string _kunci) { this->kunci = removeKarakterLain(_kunci); }
85 void FVC::setCipher(string _cipher) { this->cipher = _cipher; }
86 void FVC::setFile(string _file) { this->file = _file; }
87
88 switch (x) {
89     case 'A': case 'a': return 11;
90     case 'B': case 'b': return 23;
91     case 'C': case 'c': return 14;
92     case 'D': case 'd': return 6;
93     case 'E': case 'e': return 0;
94     case 'F': case 'f': return 2;
95     case 'G': case 'g': return 15;
96     case 'H': case 'h': return 4;
97     case 'I': case 'i': return 21;
98     case 'J': case 'j': return 7;
99     case 'K': case 'k': return 24;
100    case 'L': case 'l': return 12;
101    case 'M': case 'm': return 16;
102    case 'N': case 'n': return 8;
103    case 'O': case 'o': return 25;
104    case 'P': case 'p': return 5;
105    case 'Q': case 'q': return 22;
106    case 'R': case 'r': return 17;
107    case 'S': case 's': return 1;
108    case 'T': case 't': return 20;
109    case 'U': case 'u': return 9;
110    case 'V': case 'v': return 18;
111    case 'W': case 'w': return 13;
112    case 'X': case 'x': return 19;
113    case 'Y': case 'y': return 3;
114    case 'Z': case 'z': return 10;
115    default: return 0;
116 }
117
118 switch (y) {
119     case 0: return 'e';
120     case 1: return 's';
121     case 2: return 'f';
122     case 3: return 'y';
123     case 4: return 'h';
124     case 5: return 'p';
125     case 6: return 'd';
126     case 7: return 'j';
127     case 8: return 'n';
128     case 9: return 'u';
129     case 10: return 'z';
130     case 11: return 'a';
131     case 12: return 'l';
132     case 13: return 'w';
133     case 14: return 'c';
134     case 15: return 'g';
135     case 16: return 'm';
136     case 17: return 'r';
137     case 18: return 'v';
138     case 19: return 'x';
139     case 20: return 't';
140     case 21: return 'i';
141     case 22: return 'q';
142     case 23: return 'b';
143     case 24: return 'k';
144     case 25: return 'o';
145     default: return 'x';
146 }

```

3. Auto-Key Vigenere Cipher - AVC.cpp

```

#include "AVC.h"
#include <iostream>
#include <string>

using namespace std;

AVC::AVC() {
    plain = "null";
    kunci = "null";
    cipher = "null";
    file = "null";
}

void AVC::AVC_Encripsi()

```

```

{
    string plain2 = getPlain();
    //Remove karakter non-alphabetic
    string plain = removeKarakterLain(plain2);
    //Menyelaraskan kunci dengan panjang Plain text
    string kunci = generateKunci(plain, this->getKunci());
    string tampungan2 = "";
    for (int i = 0; i < (int)plain.size(); i++) {
        int p = charToInt(plain[i], true);
        int k = charToInt(kunci[i], true);
        int c = (p + k) % 26;
        char cnnya = intToChar(c, true);
        tampungan2.push_back(cnnya);
    }
    string temp = filterOutput(tampungan2, 5);
    setCipher(temp);
}

void AVC::AVC_Encripsi_File()
{
    string cipher = getCipher();
    setFile(cipher);
}

void AVC::AVC_Dekripsi() {
    string cipher2 = getCipher();
    //Remove karakter non-alphabetic
    string cipher = removeKarakterLain(cipher2);
    //Menyelaraskan kunci dengan panjang Plain text
    string kunci = generateKunci(cipher, this->getKunci());
    string tampungan2 = "";
    for (int i = 0; i < (int)cipher.size(); i++) {
        int c = charToInt(cipher[i], true);
        int k = charToInt(kunci[i], true);
        if (c - k < 0) {
            c += 26;
        }
        int p = (c - k) % 26;
        char pnya = intToChar(p, true);
        tampungan2.push_back(pnya);
    }
    setPlain(tampungan2);
}

void AVC::AVC_Dekripsi_File() {
    string plain = getPlain();
    setFile(plain);
}

string AVC::generateKunci(string _plain, string _kunci)
{
    string tampungan;
    tampungan += _kunci;
    tampungan += removeKarakterLain(_plain);
    tampungan.resize((int) _plain.size());
    return tampungan;
}

string AVC::getPlain() { return this->plain; }
string AVC::getKunci() { return this->kunci; }
string AVC::getCipher() { return this->cipher; }
string AVC::getFile() { return this->file; }

void AVC::setPlain(string _plain) { this->plain = _plain; }
void AVC::setKunci(string _kunci) { this->kunci = removeKarakterLain(_kunci); }
void AVC::setCipher(string _cipher) { this->cipher = _cipher; }
void AVC::setFile(string _file) { this->file = _file; }

```

4. Extended Vigenere Cipher - FCS.cpp

```

1  #include "EVC.h"
2  #include <iostream>
3  #include <string>
4
5  using namespace std;
6
7  EVC::EVC() {
8      plain = "null";
9      kunci = "null";
10     cipher = "null";
11     file = "null";
12 }
13
14 void EVC::EVC_Encripsi()
15 {
16     string plain2 = getPlain();
17     //Menyelaraskan kunci dengan panjang Plain text
18     string kunci = generateKunci(plain, this->getKunci());
19     string tampungan2 = "";
20     for (int i = 0; i < (int)plain.size(); i++) {
21         int p = charToIntEVC(plain[i]);
22         int k = charToIntEVC(kunci[i]);
23         int c = (p + k) % 256;
24         char cnya = intToCharEVC(c);
25         tampungan2.push_back(cnya);
26     }
27     //string temp = filterOutput(tampungan2, 5);
28     setCipher(tampungan2);
29 }
30
31 void EVC::EVC_Encripsi_File()
32 {
33     string cipher = getCipher();
34     setFile(cipher);
35 }
36
37 void EVC::EVC_Dekripsi() {
38     string cipher2 = getCipher();
39     //Menyelaraskan kunci dengan panjang Plain text
40     string kunci = generateKunci(cipher, this->getKunci());
41     string tampungan2 = "";
42     for (int i = 0; i < (int)cipher.size(); i++) {
43         int c = charToIntEVC(cipher[i]);
44         int k = charToIntEVC(kunci[i]);
45         if (c - k < 0) {
46             c += 256;
47         }
48         int p = (c - k) % 256;
49         char pnya = intToCharEVC(p);
50         tampungan2.push_back(pnya);
51     }
52     setPlain(tampungan2);
53 }
54
55 void EVC::EVC_Dekripsi_File() {
56     string plain = getPlain();
57     setFile(plain);
58 }
59
60 string EVC::generateKunci(string _plain, string _kunci)
61 {
62     string tampungan;
63     int jumlahPlain = (int)_plain.size();
64     int jumlahKunci = (int)_kunci.size();
65     int faktor = ceil(jumlahPlain / jumlahKunci);
66     for (int i = 0; i <= faktor; i++) {
67         tampungan += kunci;
68     }
69     tampungan.resize(jumlahPlain);
70     return tampungan;
71 }
72
73 string EVC::getPlain() { return this->plain; }
74 string EVC::getKunci() { return this->kunci; }
75 string EVC::getCipher() { return this->cipher; }
76 string EVC::getFile() { return this->file; }
77
78 void EVC::setPlain(string _plain) { this->plain = _plain; }
79 void EVC::setKunci(string _kunci) { this->kunci = _kunci; }
80 void EVC::setCipher(string _cipher) { this->cipher = _cipher; }
81 void EVC::setFile(string _file) { this->file = _file; }
82

```

5. Playfair Cipher - PFCip.cpp

```

#include "PFCip.h"
#include <iostream>
#include <string>
#include <algorithm>
#include <map>

using namespace std;

PFCip::PFCip() {
    plain = "null";
    kunci = "null";
    cipher = "null";
    file = "null";
}

int PFCip::getIndex(vector<char> v, int K)
{

```

```

        auto it = find(v.begin(), v.end(), K);
        if (it != v.end())
        {
            int index = it - v.begin();
            return index;
        }
        else {
            return -1;
        }
    }
}

pair<int, int> PFCip::letakHuruf(vector<vector<char>> m, char c) {
    pair<int, int> index;
    for (unsigned int i = 0; i < m.size(); i++) {
        if (getIndex(m.at(i), c) != -1) {
            index.second = getIndex(m.at(i), c);
            index.first = i;
            return index;
        }
    }
    //Kalau characternya tidak ada
    index.second = -1;
    index.first = -1;
    return index;
}

void PFCip::PFCip_Enkripsi()
{
    string _kunci = this->getKunci();
    vector<char> v, v2;
    vector<vector<char>> v3;
    std::copy(_kunci.begin(), _kunci.end(), std::back_inserter(v));
    v2 = unique_elements(v);
    v3 = matriksKey(v2);
    string plain2 = getPlain();
    //Remove karakter non-alphabetic
    string plain = removeKarakterLain(plain2);
    for (unsigned int j = 0; j < plain.size(); j++) {
        if (plain[j] == 'J' || plain[j] == 'j')
        {
            plain.erase(j, 1);
            j--;
        }
    }
    //setting plain text nya
    unsigned int i = 0;
    while (i < plain.size()) {
        if (plain[i] == plain[i + 1]) {
            plain.insert((i + 1), 1, 'X');
        }
        i += 2;
    }
    if (plain.size() % 2 == 1) {
        plain.push_back('X');
    }
    //Proses enkripsi
    string cipher;
    unsigned int k = 0;
    while (k < plain.size()) {
        pair<int, int> huruf1, huruf2;
        huruf1 = letakHuruf(v3, plain[k]);
        huruf2 = letakHuruf(v3, plain[k + 1]);
        //Barisnya sama
        if (huruf1.first == huruf2.first) {
            huruf1.second += 1;
            huruf1.second %= 5;
            huruf2.second += 1;
            huruf2.second %= 5;
        }
    }
}

```

```

        cipher.append(1, v3.at(huruf1.first).at(huruf1.second));
        cipher.append(1, v3.at(huruf2.first).at(huruf2.second));
    }
    //Kolomnya sama
    else if (huruf1.second == huruf2.second) {
        huruf1.first += 1;
        huruf1.first %= 5;
        huruf2.first += 1;
        huruf2.first %= 5;
        cipher.append(1, v3.at(huruf1.first).at(huruf1.second));
        cipher.append(1, v3.at(huruf2.first).at(huruf2.second));
    }
    else {
        cipher.append(1, v3.at(huruf1.first).at(huruf2.second));
        cipher.append(1, v3.at(huruf2.first).at(huruf1.second));
    }
    k += 2;
}
string temp = filterOutput(cipher, 5);
setCipher(temp);
}

void PFCip::PFCip_Enkripsi_File()
{
    string cipher = getCipher();
    setFile(cipher);
}

void PFCip::PFCip_Dekripsi() {
    string _kunci = this->getKunci();
    vector<char> v, v2;
    vector<vector<char>> v3;
    std::copy(_kunci.begin(), _kunci.end(), std::back_inserter(v));
    v2 = unique_elements(v);
    v3 = matriksKey(v2);
    string cipher = getCipher();
    //Remove karakter non-alphabetic
    cipher = removeKarakterLain(cipher);
    for (unsigned int j = 0; j < cipher.size(); j++) {
        if (cipher[j] == 'J' || cipher[j] == 'j')
        {
            cipher.erase(j, 1);
            j--;
        }
    }
    //Proses dekripsi
    string plain;
    unsigned int k = 0;
    while (k < cipher.size()) {
        pair<int, int> huruf1, huruf2;
        huruf1 = letakHuruf(v3, cipher[k]);
        huruf2 = letakHuruf(v3, cipher[k + 1]);
        //Barisnya sama
        if (huruf1.first == huruf2.first) {
            if ((huruf1.second) - 1 < 0) {
                huruf1.second += 5;
            }
            huruf1.second -= 1;
            if ((huruf2.second) - 1 < 0) {
                huruf2.second += 5;
            }
            huruf2.second -= 1;
            plain.append(1, v3.at(huruf1.first).at(huruf1.second));
            plain.append(1, v3.at(huruf2.first).at(huruf2.second));
        }
        //Kolomnya sama
        else if (huruf1.second == huruf2.second) {
            if ((huruf1.first) - 1 < 0) {

```

```

        huruf1.first += 5;
    }
    huruf1.first -= 1;
    if ((huruf2.first) - 1 < 0) {
        huruf2.first += 5;
    }
    huruf2.first -= 1;
    plain.append(1, v3.at(huruf1.first).at(huruf1.second));
    plain.append(1, v3.at(huruf2.first).at(huruf2.second));
}
else {
    plain.append(1, v3.at(huruf1.first).at(huruf2.second));
    plain.append(1, v3.at(huruf2.first).at(huruf1.second));
}
k += 2;
}
//setting plain text nya
for (unsigned int i = 0; i < plain.size(); i++) {
    if (plain[i] == 'X' || plain[i] == 'x')
    {
        plain.erase(i, 1);
        i--;
    }
}
setPlain(plain);
}

void PFCip::PFCip_Dekripsi_File() {
    string plain = getPlain();
    setFile(plain);
}

vector<char> PFCip::unique_elements(vector<char>& vec)
{
    vector<char> vec2;
    map<char, int> m;
    for (auto p = vec.begin(); p != vec.end(); ++p) {
        if (*p != 'J' && *p >= 'A' && *p <= 'Z') {
            m[*p]++;
        }
        if (m[*p] > 0 && m[*p] < 2) {
            vec2.push_back(*p);
        }
    }
    for (char i = 'A'; i <= 'Z'; i++) {
        if (i != 'J' && m[i] == 0) {
            vec2.push_back(i);
        }
    }
    return vec2;
}

vector<vector<char>> PFCip::matriksKey(vector<char> kunci) {
    vector<vector<char>> m;
    for (int i = 0; i < 25; i++) {
        if (i % 5 == 0) {
            m.push_back(vector<char>());
        }
        m.at(round(i / 5)).push_back(kunci.at(i));
    }
    return m;
}

vector<vector<char>> PFCip::generateKunci(string _kunci)
{
    vector<char> v, v2;
    vector<vector<char>> v3;
    std::copy(_kunci.begin(), _kunci.end(), std::back_inserter(v));

```



```

        v2 = unique_elements(v);
        return matriksKey(v2);
    }

    string PFCip::getPlain() { return this->plain; }
    string PFCip::getKunci() { return this->kunci; }
    string PFCip::getCipher() { return this->cipher; }
    string PFCip::getFile() { return this->file; }

    void PFCip::setPlain(string _plain) { this->plain = _plain; }
    void PFCip::setKunci(string _kunci) { this->kunci = removeKarakterLain(_kunci); }
    void PFCip::setCipher(string _cipher) { this->cipher = _cipher; }
    void PFCip::setFile(string _file) { this->file = _file; }

```

6. Affine Cipher - AFC.cpp

```

1  #include "AFC.h"
2  #include <iostream>
3  #include <string>
4
5  using namespace std;
6
7  AFC::AFC() {
8      plain = "null";
9      kunci_m = "null";
10     kunci_b = "null";
11     cipher = "null";
12     file = "null";
13 }
14
15 void AFC::AFC_Encripsi()
16 {
17     string plain2 = getPlain();
18     //Remove karakter non-alphabetic
19     string plain = removeKarakterLain(plain2);
20     //Menyelaraskan kunci dengan panjang Plain text
21     int m = stoi(this->getKunciM());
22     int b = stoi(this->getKunciB());
23     string tampungan2 = "";
24     for (int i = 0; i < (int)plain.size(); i++) {
25         int p = charToInt(plain[i], true);
26         int c = ((m * p) + b) % 26;
27         char cnya = intToChar(c, true);
28         tampungan2.push_back(cnya);
29     }
30     string temp = filterOutput(tampungan2, 5);
31     setCipher(temp);
32 }
33
34 void AFC::AFC_Encripsi_File()
35 {
36     string cipher = getCipher();
37     setFile(cipher);
38 }
39
40 void AFC::AFC_Dekripsi() {
41     int k, pengali;
42     bool found = false;
43     string cipher2 = getCipher();
44     //Remove karakter non-alphabetic
45     string cipher = removeKarakterLain(cipher2);
46     //Menyelaraskan kunci dengan panjang Plain text
47     int m = stoi(this->getKunciM());
48     int b = stoi(this->getKunciB());
49     //Menyelaraskan kunci dengan panjang Plain text
50     pengali = 1;
51     while (!found) {
52         if (((pengali * 26) + 1) % 26 == 1) && (((pengali * 26) + 1) % m == 0)) {
53             k = ((pengali * 26) + 1) / m;
54             found = true;
55         }
56         pengali++;
57     }
58     string tampungan2 = "";
59     for (int i = 0; i < (int)cipher.size(); i++) {
60         int c = charToInt(cipher[i], true);
61         int p = ((c - b) * (k));
62         if (p < 0) {
63             int faktor = ((-p) / 26) + 1;
64             p += faktor * 26;
65         }
66         p %= 26;
67         char pnya = intToChar(p, true);
68         tampungan2.push_back(pnya);
69     }
70     setPlain(tampungan2);
71 }
72
73 void AFC::AFC_Dekripsi_File() {
74     string plain = getPlain();
75     setFile(plain);
76 }

```

```

78     string AFC::getPlain() { return this->plain; }
79     string AFC::getKunciB() { return this->kunci_b; }
80     string AFC::getKunciM() { return this->kunci_m; }
81     string AFC::getCipher() { return this->cipher; }
82     string AFC::getFile() { return this->file; }
83
84     void AFC::setPlain(string _plain) { this->plain = _plain; }
85     void AFC::setKunciB(string _kunci) { this->kunci_b = _kunci; }
86     void AFC::setKunciM(string _kunci) { this->kunci_m = _kunci; }
87     void AFC::setCipher(string _cipher) { this->cipher = _cipher; }
88     void AFC::setFile(string _file) { this->file = _file; }

```

BONUS

7. Hill Cipher - HC.cpp

```
1  #include "HCip.h"
2  #include <iostream>
3
4  using namespace std;
5
6  HCip::HCip() {
7      plain = "null";
8      kunci = "null";
9      cipher = "null";
10     file = "null";
11 }
12
13 vector<int> HCip::kaliMatEnkripsi(vector<vector<int>> kunci, vector<int> plain) {
14     vector<int> hasil(kunci.size(), 0);
15     for (unsigned int i = 0; i < kunci.size(); i++) {
16         for (int u = 0; u < kunci.size(); u++) {
17             hasil[i] += kunci[i][u] * plain[u];
18         }
19         hasil[i] %= 26;
20     }
21     return hasil;
22 }
23
24 void HCip::HCip_Enkripsi()
25 {
26     string _kunci = this->getKunci();
27     vector<char> v, v2;
28     vector<vector<int>> k3;
29     k3 = generateKunci(_kunci);
30     string plain2 = getPlain();
31     //Remove karakter non-alphabetic
32     string plain = removeKarakterLain(plain2);
33     //Proses enkripsi
34     string cipher;
35     unsigned int k = 0;
36     int faktor = floor(plain.size() / 3);
37     int sisa = plain.size() % 3;
38     for (int i = 0; i < sisa; i++) {
39         plain.push_back('x');
40     }
41     vector<int> temp_plain;
42     vector<int> result;
43     while (k < plain.size()) {
44         temp_plain.push_back(charToInt(plain[k], true));
45         if ((k % 3 == 2) && (temp_plain.size() == 3)) {
46             result = kaliMatEnkripsi(k3, temp_plain);
47             for (int index = 0; index < 3; index++) {
48                 cipher.push_back(intToChar(result[index], true));
49             }
50             temp_plain.clear();
51             result.clear();
52         }
53         k++;
54     }
55     for (int i = 0; i < sisa; i++) {
56         plain.pop_back();
57     }
58     string temp = filterOutput(cipher, 5);
59     setCipher(temp);
60 }
61
62 void HCip::HCip_Dekripsi() {
63     string _kunci = this->getKunci();
64     vector<char> v, v2;
65     vector<vector<int>> k3, k2, k1, i1, i2, i3;
66     k3 = generateKunci(_kunci);
67     i3 = generateInvers(k3);
68     string cipher = getCipher();
69     //Remove karakter non-alphabetic
70     cipher = removeKarakterLain(cipher);
71     //Proses dekripsi
72     string plain;
73     unsigned int k = 0;
74     int sisa = cipher.size() % 3;
75     vector<int> temp_cipher;
76     vector<int> result;
77     while (k < cipher.size()) {
78         temp_cipher.push_back(charToInt(cipher[k], true));
79         if ((k % 3 == 2) && (temp_cipher.size() == 3)) {
80             result = kaliMatEnkripsi(i3, temp_cipher);
81             for (int index = 0; index < 3; index++) {
82                 plain.push_back(intToChar(result[index], true));
83             }
84             temp_cipher.clear();
85             result.clear();
86         }
87         k++;
88     }
89     for (int i = plain.size()-1; i > 0; i--) {
90         if (plain[i] == 'x') {
91             plain.pop_back();
92         }
93     }
94     setPlain(plain);
95 }
96
97 void HCip::HCip_Dekripsi_File() {
98     string plain = getPlain();
99     setFile(plain);
100 }
101
102 vector<vector<int>> HCip::generateKunci(string _kunci)
103 {
104     return vector<vector<int>>{
105         {17, 17, 5},
106         {21, 18, 21},
107         {2, 2, 19}
108     };
109 }
110
111 vector<vector<int>> HCip::generateInvers(vector<vector<int>> matriks) {
112     return vector<vector<int>>{
113         {4,9,15},
114         {15,17,6},
115         {24,0,17}
116     };
117 }
118
119 string HCip::getPlain() { return this->plain; }
120 string HCip::getKunci() { return this->kunci; }
121 string HCip::getCipher() { return this->cipher; }
122 string HCip::getFile() { return this->file; }
123
124 void HCip::setPlain(string _plain) { this->plain = _plain; }
125 void HCip::setKunci(string _kunci) { this->kunci = removeKarakterLain(_kunci); }
126 void HCip::setCipher(string _cipher) { this->cipher = _cipher; }
127 void HCip::setFile(string _file) { this->file = _file; }
```

8. Enigma Cipher - ECip.cpp

```
195 char ECip::ptoc(char karakter) {
196     for (auto itr = m1.find(karakter); itr != m1.end(); itr++) {
197         for (auto itr2 = m2.find(itr->second); itr2 != m2.end(); itr2++) {
198             for (auto itr3 = m3.find(itr2->second); itr3 != m3.end(); itr3++) {
199                 for (auto itr4 = m4.find(itr3->second); itr4 != m4.end(); itr4++) {
200                     for (auto itr5 = m5.find(itr4->second); itr5 != m5.end(); itr5++) {
201                         for (auto itr6 = m6.find(itr5->second); itr6 != m6.end(); itr6++) {
202                             for (auto itr7 = m7.find(itr6->second); itr7 != m7.end(); itr7++) {
203                                 return itr7->second;
204                             }
205                         }
206                     }
207                 }
208             }
209         }
210     }
211 }
```

```
213 char ECip::ctop(char karakter) {
214     for (auto& it7 : m7) {
215         if (it7.second == karakter) {
216             for (auto& it6 : m6) {
217                 if (it6.second == it7.first) {
218                     for (auto& it5 : m5) {
219                         if (it5.second == it6.first) {
220                             for (auto& it4 : m4) {
221                                 if (it4.second == it5.first) {
222                                     for (auto& it3 : m3) {
223                                         if (it3.second == it4.first) {
224                                             for (auto& it2 : m2) {
225                                                 if (it2.second == it3.first) {
226                                                     for (auto& it1 : m1) {
227                                                         if (it1.second == it2.first) {
228                                                             return it1.first;
229                                                         }
230                                                     }
231                                                 }
232                                             }
233                                         }
234                                     }
235                                 }
236                             }
237                         }
238                     }
239                 }
240             }
241         }
242     }
243 }
```

```
void ECip::ECip_Encripsi()
{
    string plain2 = getPlain();
    //Remove karakter non-alphabetic
    string plain = removeKarakterLain(plain2);
    //Proses enkripsi
    string cipher;
    for (unsigned int i = 0; i < plain.size(); i++) {
        cipher.push_back(ptoc(plain[i]));
    }
}
```

```

        string temp = filterOutput(cipher, 5);
        setCipher(temp);
    }

    void ECip::ECip_Enkripsi_File()
    {
        string cipher = getCipher();
        setFile(cipher);
    }

    void ECip::ECip_Dekripsi() {
        string _kunci = this->getKunci();
        string cipher = getCipher();
        //Remove karakter non-alphabetic
        cipher = removeKarakterLain(cipher);
        //Proses dekripsi
        string plain;
        for (unsigned int i = 0; i < cipher.size(); i++) {
            plain.push_back(ctop(cipher[i]));
        }
        setPlain(plain);
    }

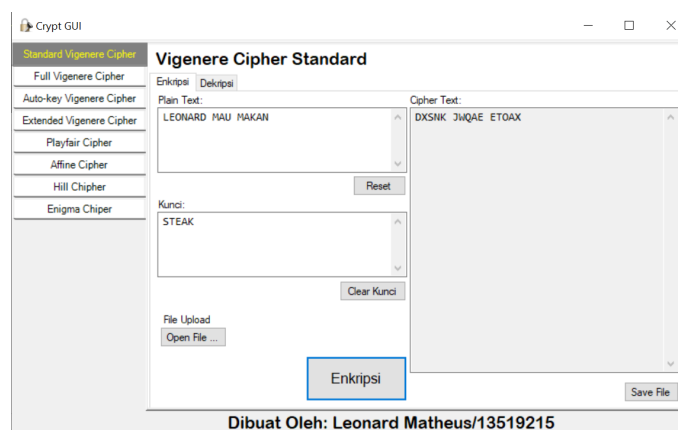
    void ECip::ECip_Dekripsi_File() {
        string plain = getPlain();
        setFile(plain);
    }

    string ECip::getPlain() { return this->plain; }
    string ECip::getKunci() { return this->kunci; }
    string ECip::getCipher() { return this->cipher; }
    string ECip::getFile() { return this->file; }

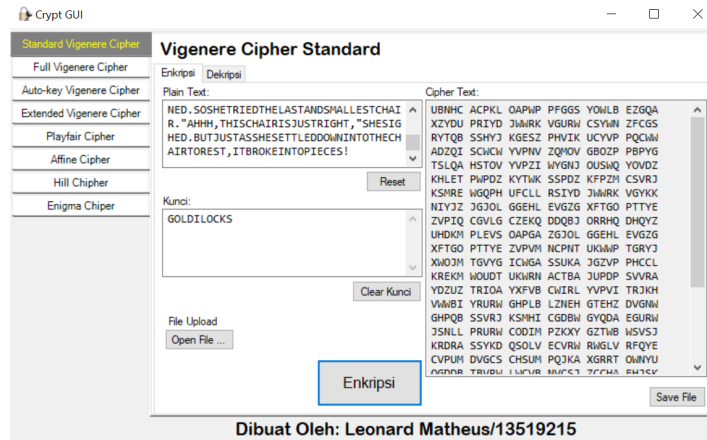
    void ECip::setPlain(string _plain) { this->plain = _plain; }
    void ECip::setKunci(string _kunci) { this->kunci = removeKarakterLain(_kunci); }
    void ECip::setCipher(string _cipher) { this->cipher = _cipher; }
    void ECip::setFile(string _file) { this->file = _file; }

```

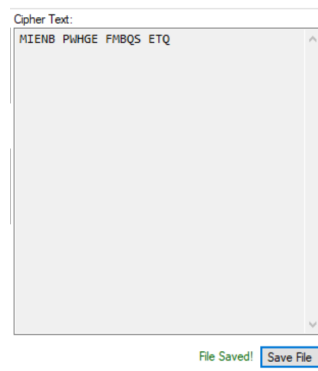
Tampilan Antarmuka Aplikasi



Tampilan Utama

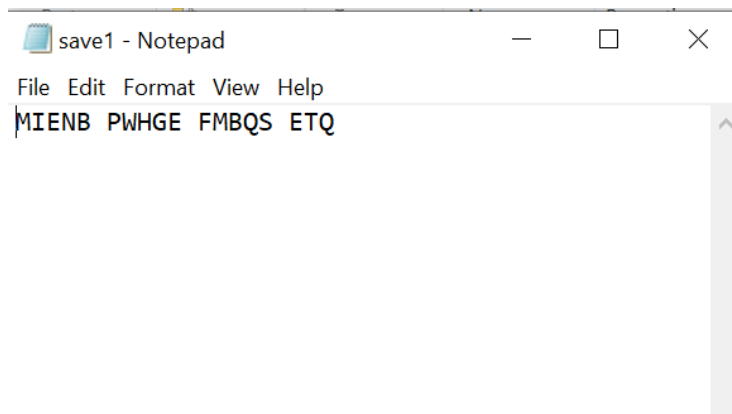


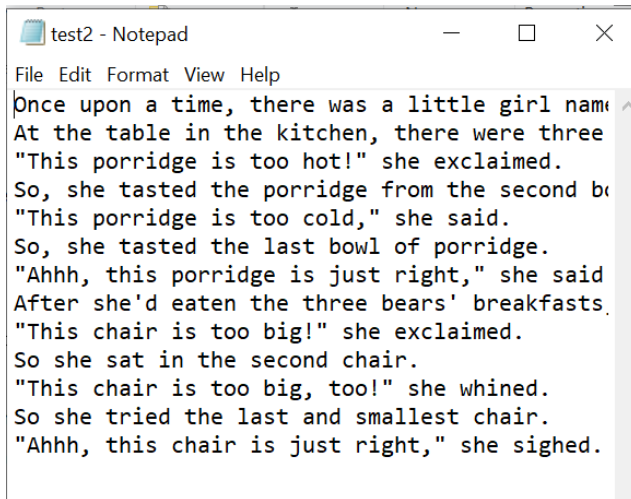
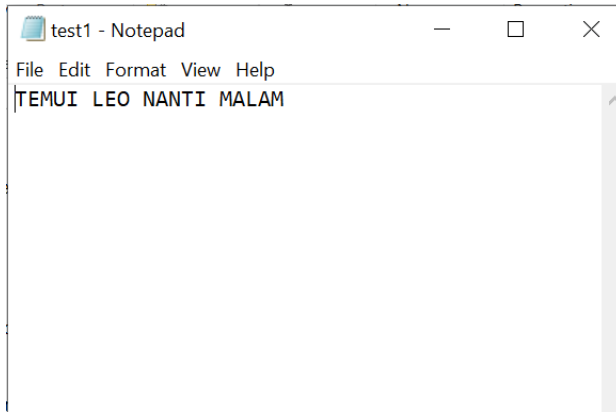
Handle Text Cipher yang banyak



Fitur Open file dan save file

Contoh File





No.	Spek	Berhasil (✓)	Kurang Berhasil (✓)	Keterangan
1	Vigenere Cipher standard	✓		Text bisa enkripsi-dekripsi dan kembali ke asal
2	Full Vigenere Cipher	✓		Text bisa enkripsi-dekripsi dan kembali ke asal
3	Auto-Key Vigenere Cipher	✓		Text bisa enkripsi-dekripsi dan kembali ke asal
4	Extended Vigenere Cipher	✓		Text bisa enkripsi-dekripsi

				dan kembali ke asal
5	Playfair cipher	✓		Text bisa enkripsi-dekripsi dan kembali ke asal
6	Affine Cipher	✓		Text bisa enkripsi-dekripsi dan kembali ke asal
7	Hill Cipher	✓		Text bisa enkripsi-dekripsi dan kembali ke asal
8	Enigma cipher	✓		Text bisa enkripsi-dekripsi dan kembali ke asal

LETAK DRIVE UNTUK DIUJI COBA

https://github.com/leomatt547/Crypt_GUI