

LAPORAN
STUDI KASUS
MATEMATIKA DISKRIT
“ Program Enkripsi dan Dekripsi Kuantum Caesar Cipher
(QCC) v1.0 ”



Disusun Oleh:

Nama : Reyhanssan Islamey
NIM : 2200018411
Kelas : A
Slot : Selasa 12.00

LABORATORIUM KOMPUTER INFORMATIKA
PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS AHMAD DAHLAN
YOGYAKARTA
2023

A. DESKRIPSI STUDI KASUS

Sebagai salah satu jenis cipher yang paling sederhana dalam dunia kriptografi, Caesar Cipher telah digunakan sejak lama untuk mengamankan pesan-pesan rahasia. Cara kerja Caesar Cipher adalah dengan menggeser setiap huruf dalam plaintext sejauh n karakter, di mana n adalah jumlah pergeseran yang ditentukan. Sebagai contoh, huruf A akan diganti dengan huruf B apabila jumlah pergeseran n adalah 1. Namun, Caesar Cipher memiliki kelemahan yang besar, yaitu mudah untuk didekripsi dengan melakukan brute force pada setiap kemungkinan pergeseran.

Dalam upaya untuk meningkatkan keamanan Caesar Cipher, terdapat suatu metode baru yang menggunakan Quantum Computing sebagai alat untuk melakukan enkripsi dan dekripsi pesan. Metode tersebut menggunakan konsep superposisi dan entanglement dari partikel sub-atomik untuk membuat pesan tersebut menjadi aman dari serangan brute force. Dengan menggunakan teknologi Quantum Computing, Caesar Cipher dapat diimplementasikan dengan lebih aman dan efektif dibandingkan dengan metode enkripsi klasik lainnya.

Maka dari itu, dalam studi kasus ini, saya akan mempelajari cara implementasi Caesar Cipher pada Quantum Computing menggunakan bahasa pemrograman Python dan library Qiskit. Dalam studi kasus ini, saya akan menggunakan materi yang telah dipelajari pada pertemuan sebelumnya dan menerapkan konsep Quantum Computing pada kasus yang lebih mendalam.

Dalam kehidupan sehari-hari, keamanan pesan dan informasi penting sangatlah krusial. Saat ini, saya sering kali mengirim pesan-pesan rahasia melalui email, pesan teks, atau bahkan media sosial. Namun, saya seringkali khawatir bahwa pesan tersebut akan dibaca oleh orang yang tidak berhak atau bahkan oleh pengguna jaringan yang tidak bertanggung jawab. Oleh karena itu, Caesar Cipher dengan Quantum Computing menjadi sangat penting dalam menjaga keamanan dan kerahasiaan data dalam berbagai bidang, seperti keuangan, militer, dan bisnis.

Dalam bidang keuangan, Caesar Cipher dengan Quantum Computing dapat membantu melindungi data keuangan dan informasi penting dari pencurian dan serangan cyber. Dalam bidang militer, teknologi ini dapat melindungi pesan-pesan rahasia dan strategi militer dari musuh. Dalam bisnis, Caesar Cipher dengan Quantum Computing dapat membantu mengamankan data perusahaan dan informasi penting dari pesaing.

Dengan demikian, implementasi Caesar Cipher pada Quantum Computing memiliki banyak manfaat dan kegunaan dalam kehidupan sehari-hari. Penggunaan teknologi ini dapat membantu menjaga keamanan dan kerahasiaan data dalam berbagai bidang, dan menjadi solusi untuk masalah sehari-hari yang seringkali dihadapi oleh banyak orang.

Pada Caesar Cipher dengan Quantum Computing, setiap huruf dalam plaintext dienkripsi dengan menggeser sejauh n karakter. Namun, pada Quantum Computing, pergeseran dilakukan dengan menggunakan operator rotasi Z . Operator rotasi Z ini membawa sebuah qubit ke arah sumbu Z pada Bloch sphere, dan menggeser fase qubit sejauh θ .

Rumus rotasi Z adalah sebagai berikut:

$$R_z(\theta) = \exp(-i\theta Z/2)$$

Di mana θ adalah sudut rotasi, dan Z adalah operator Pauli Z . Dalam Caesar Cipher dengan Quantum Computing, kita menghitung nilai θ berdasarkan jumlah pergeseran yang ditentukan pada plaintext.

Misalnya, jika $n = 1$, maka $\theta = 2\pi/26$.

Setelah itu, kita mengaplikasikan operator rotasi Z pada setiap qubit dari plaintext. Kemudian, kita mengaplikasikan operator Hadamard pada setiap qubit untuk membuatnya berada dalam keadaan superposisi. Dalam keadaan superposisi ini, setiap

qubit memiliki kemungkinan untuk memiliki nilai 0 atau 1 dengan probabilitas yang sama.

Setelah itu, kita mengaplikasikan operator rotasi Z lagi untuk melakukan dekripsi pada ciphertext. Dalam dekripsi, kita menggunakan nilai negatif dari θ untuk menggeser balik setiap qubit. Kemudian, kita mengaplikasikan operator Hadamard lagi untuk mendapatkan nilai plaintext yang asli.

Dalam Caesar Cipher dengan Quantum Computing, keamanan pesan terjamin karena kunci enkripsi hanya dapat dipecahkan dengan Quantum Computer yang memiliki jumlah qubit yang cukup besar. Selain itu, serangan brute force menjadi tidak efektif karena setiap kemungkinan pergeseran akan menghasilkan hasil yang berbeda-beda pada setiap qubit.

Dalam studi kasus ini, kita akan mempelajari cara mengimplementasikan algoritma Caesar Cipher pada Quantum Computer menggunakan bahasa pemrograman Python dan Qiskit Library. Kita akan mempelajari konsep dasar Quantum Computing dan cara menerapkannya pada algoritma Caesar Cipher dengan menggunakan operator rotasi Z dan Hadamard. Dengan cara ini, kita dapat meningkatkan keamanan pesan-pesan rahasia dan informasi penting di kehidupan sehari-hari.

Saya sendiri sangat tertarik dengan kriptografi dan teknologi Quantum Computing, dan memilih untuk mengambil studi kasus ini karena ingin mempelajari lebih dalam mengenai cara mengamankan data dan informasi penting. Saya yakin bahwa dengan mempelajari studi kasus ini, saya dapat berkontribusi dalam menciptakan solusi yang lebih aman dan efektif untuk masalah keamanan data di kehidupan sehari-hari.

B. KESULITAN MASALAH

Saya mengalami beberapa kesulitan dalam studi kasus ini. Salah satu kesulitan utama adalah pada pemahaman konsep dasar Quantum Computing. Konsep dasar seperti Superposisi dan Entanglement memerlukan pemahaman yang mendalam dalam fisika kuantum, dan perlu banyak waktu untuk mempelajarinya dengan benar.

Selain itu, saya juga mengalami kesulitan dalam menentukan key dari qubit kuantum agar sulit ditebak oleh orang lain. Dalam Caesar Cipher klasik, key yang mudah ditebak dapat mengurangi keamanan dari enkripsi. Saya menggunakan pendekatan yang sama dengan Caesar Cipher klasik, yaitu dengan menghitung frekuensi kemunculan dari setiap karakter pada plaintext. Namun, pada Quantum Computing, saya menggunakan nilai kemunculan tersebut sebagai peluang kemunculan pada setiap qubit. Menghitung peluang kemunculan ini memerlukan penghitungan matematis yang teliti dan memakan waktu.

Kesulitan lain yang saya alami adalah pada implementasi algoritma Caesar Cipher pada Quantum Computer menggunakan bahasa pemrograman Python dan Qiskit Library. Pada awalnya, saya mengalami kesulitan dalam membuat sirkuit Quantum yang tepat dan memahami cara kerjanya. Saya juga mengalami kesulitan dalam melakukan pengukuran hasil dari sirkuit Quantum yang telah dibuat. Saat menjalankan program, saya mengalami beberapa error dan bug yang membuat saya harus memperbaiki kode secara terus-menerus. Beberapa bagian dari program juga memerlukan waktu yang cukup lama untuk dijalankan, karena penggunaan Quantum Computer memerlukan waktu yang lebih lama dibandingkan dengan komputer klasik.

Hal ini juga yang menjadi salah satu kesulitan dalam implementasi algoritma Caesar Cipher pada Quantum Computer menggunakan bahasa pemrograman Python dan Qiskit Library. Saya mengalami kesulitan dalam menjalankan program karena program sering mengalami force close atau crash saat teks yang dimasukkan terlalu panjang. Bahkan, saat saya menjalankan program di Google Colab untuk mengurangi beban komputasi pada laptop saya, sesi dari cloud tersebut juga tidak mampu menangani kompleksitas program dan langsung crash.

Dalam mengatasi kesulitan ini, saya melakukan beberapa optimasi pada program, seperti mengurangi jumlah pergeseran yang digunakan dan mengurangi jumlah plaintext yang dienkripsi. Saya juga melakukan beberapa pengujian untuk menentukan batas maksimum karakter yang dapat dienkripsi tanpa menyebabkan program crash.

C. MANFAAT

Program Quantum Computing dengan Caesar Cipher dapat memberikan manfaat yang besar dalam kehidupan sehari-hari. Dalam era digital yang semakin maju ini, keamanan data dan informasi sangatlah penting. Dengan program ini, pesan-pesan rahasia dan informasi penting dapat dijamin kerahasiaannya dari serangan hacker dan pencurian data.

Latar belakang dari program ini adalah bahwa Caesar Cipher telah digunakan selama berabad-abad sebagai metode enkripsi paling sederhana. Namun, metode ini rentan terhadap serangan brute force dan memiliki kelemahan besar dalam menjaga kerahasiaan data. Oleh karena itu, program ini dikembangkan untuk meningkatkan tingkat keamanan pada enkripsi pesan menggunakan Caesar Cipher dengan Quantum Computing, sehingga pesan yang dikirimkan dapat dijamin kerahasiaannya.

Alasan pembuatan program ini adalah untuk meningkatkan keamanan data dan informasi di berbagai sektor, seperti keuangan, militer, dan bisnis. Program ini juga dapat membantu dalam menjaga kerahasiaan data pribadi dan informasi penting di kehidupan sehari-hari, seperti informasi perbankan, kesehatan, dan lain sebagainya.

Tujuan dari program ini adalah agar pesan-pesan yang dikirimkan dapat dienkripsi dengan aman menggunakan Caesar Cipher dengan Quantum Computing sehingga dapat dijamin kerahasiaannya. Selain itu, diharapkan program ini dapat meningkatkan keamanan data dan informasi di berbagai sektor dan membantu dalam menjaga kerahasiaan data pribadi dan informasi penting di kehidupan sehari-hari.

Harapan kedepannya adalah agar program ini dapat digunakan secara luas oleh masyarakat dan dapat memberikan manfaat yang besar bagi keamanan data dan informasi di berbagai sektor. Selain itu, diharapkan program ini dapat terus dikembangkan dan ditingkatkan

kualitasnya agar dapat menangani masalah keamanan data yang semakin kompleks di masa depan.

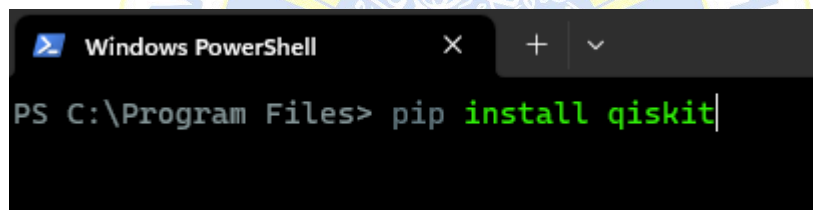
D. HASIL DAN PEMBAHASAN

Pertama, pastikan bahwa Anda sudah meng-install Qiskit, PyQt5, dan PySide2 pada komputer Anda. Qiskit adalah library Python yang digunakan untuk membangun program Quantum Computing. PyQt5 dan PySide2 adalah library Python yang digunakan untuk membuat antarmuka grafis pada program. Pastikan juga bahwa Anda sudah memahami dasar-dasar Quantum Computing dan pemrograman Python, agar dapat memahami fungsi-fungsi yang digunakan dalam program.

Berikut cara intstallasinya :

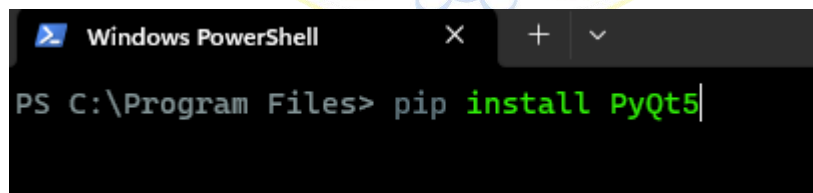
Pertama buka terminal pada computer lalu ketikan command berikut :

Qiskit Library



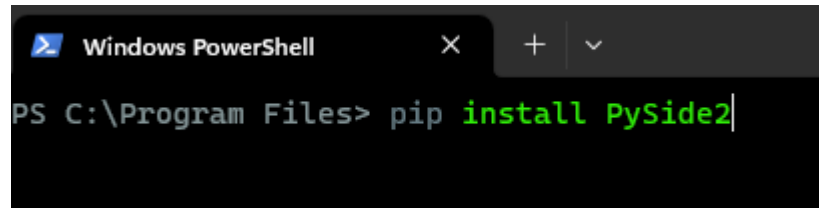
```
Windows PowerShell
PS C:\Program Files> pip install qiskit
```

PyQt5 library



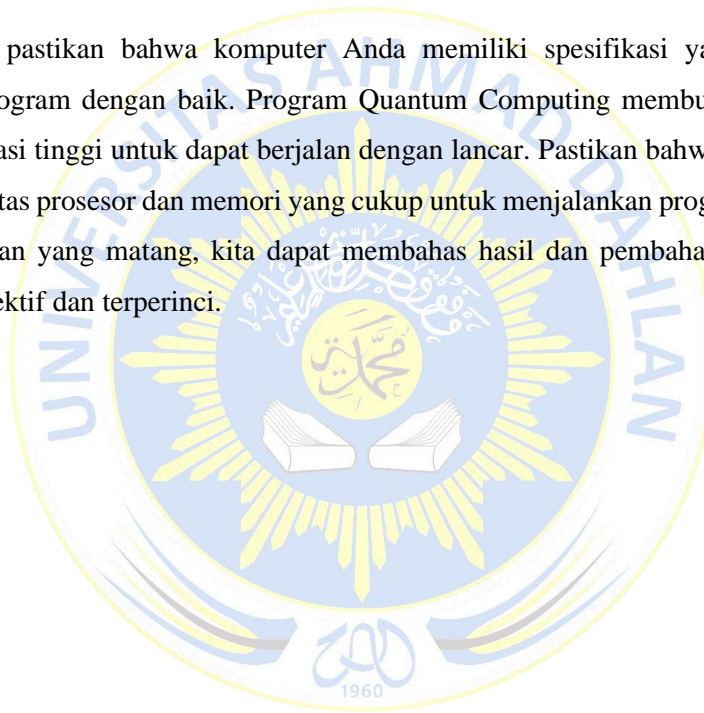
```
Windows PowerShell
PS C:\Program Files> pip install PyQt5
```

PySide2 library

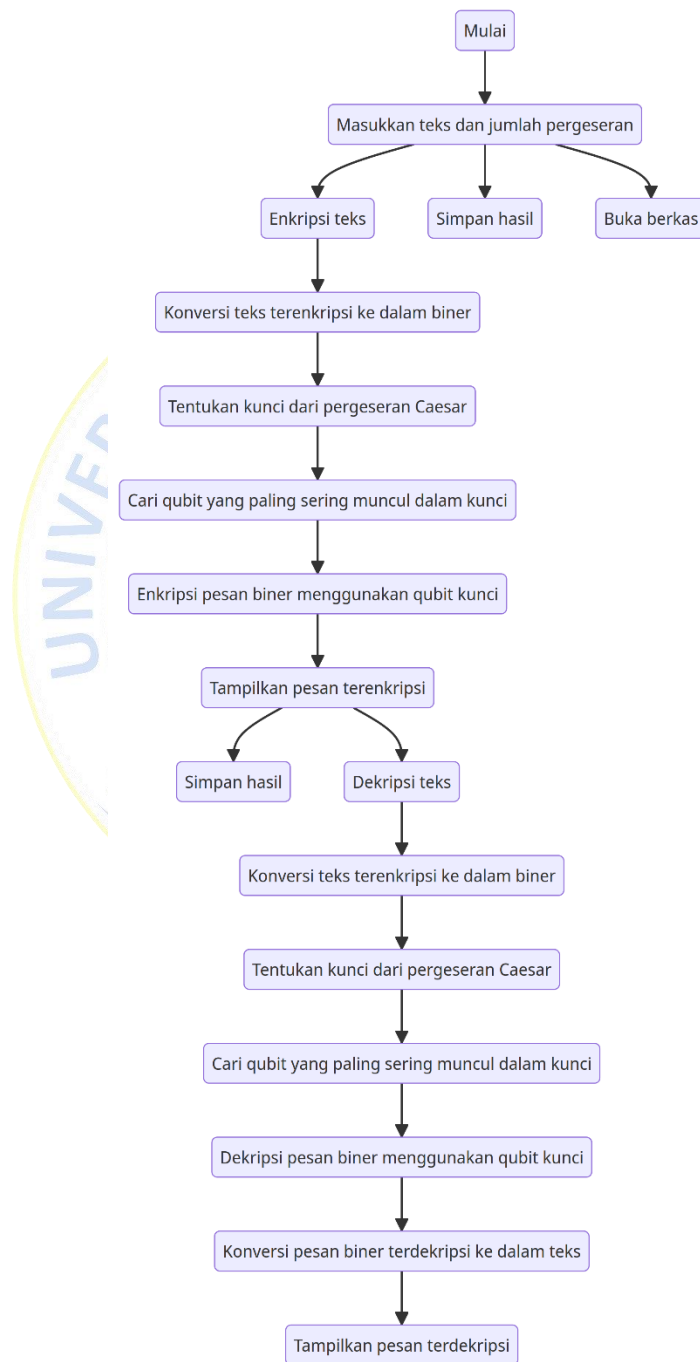


```
Windows PowerShell
PS C:\Program Files> pip install PySide2
```

Kedua, pastikan bahwa komputer Anda memiliki spesifikasi yang cukup untuk menjalankan program dengan baik. Program Quantum Computing membutuhkan komputer dengan spesifikasi tinggi untuk dapat berjalan dengan lancar. Pastikan bahwa komputer Anda memiliki kapasitas prosesor dan memori yang cukup untuk menjalankan program dengan baik. Dengan persiapan yang matang, kita dapat membahas hasil dan pembahasan dari program dengan lebih efektif dan terperinci.



Berikut kerangka kerja dari program agar dapat membantu pemahaman terkait dengan program Quantum Computing yang saya buat :



1. CODE DOCUMENTATION

```

1  #!/usr/bin/env python
2  from qiskit import QuantumCircuit, transpile, assemble, Aer, execute
3  from qiskit.visualization import plot_histogram
4  from qiskit.circuit import QuantumCircuit, Qubit, QubitLayout, QubitIndex, QubitIndexList, QubitIndexSet
5  from qiskit.circuit import QubitIndexList
6
7  def caesar_encrypt(text, shift):
8      encrypted_text = ""
9      for char in text:
10         if char.isalpha():
11             encrypted_char = chr((ord(char) - ord('a') + shift) % 26 + ord('a'))
12         else:
13             encrypted_char = chr((ord(char) - ord('A') + shift) % 26 + ord('A'))
14         encrypted_text += encrypted_char
15     return encrypted_text
16
17 def text_to_binary(text):
18     binary_string = ""
19     for char in text:
20         binary_char = format(ord(char), '08b')
21         binary_string += binary_char
22     return binary_string
23
24 def binary_to_text(binary):
25     text = ""
26     for i in range(0, len(binary), 8):
27         binary_char = binary[i:i+8]
28         char = chr(int(binary_char, 2))
29         text += char
30     return text
31
32 def get_key_from_cesar(shift):
33     alphabet = "abcdefghijklmnopqrstuvwxyz"
34     shifted_alphabet = alphabet[shift:] + alphabet[:shift]
35     key = shifted_alphabet
36     for char in shifted_alphabet:
37         if shifted_alphabet.count(char) > alphabet.count(char):
38             key = char
39     key_binary = format(ord(key), '08b')
40     return key_binary
41
42 def most_frequent_bit(key):
43     count_0 = 0
44     count_1 = 0
45     for bit in key:
46         if bit == '0':
47             count_0 += 1
48         elif bit == '1':
49             count_1 += 1
50     if count_0 > count_1:
51         return '0'
52     else:
53         return '1'
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

101  def encrypt_message(message, key):
102      circuit = QuantumCircuit(len(message), len(message))
103      for i, bit in enumerate(message):
104         if bit == '0':
105             circuit.x(i)
106         if bit == '1':
107             circuit.z(i)
108         circuit.measure(circuit.qubits[i], range(len(message)))
109         simulator = Aer.get_backend('qasm_simulator')
110         job = execute(circuit, simulator, shots=1)
111         result = job.result()
112         counts = result.get_counts(circuit)
113         encrypted_message = list(counts.keys())[0]
114         return encrypted_message
115
116 def decrypt_message(encrypted_message, key_qubit):
117     circuit = QuantumCircuit(len(encrypted_message), len(encrypted_message))
118     for i, bit in enumerate(key_qubit):
119         if bit == '0':
120             circuit.x(i)
121         if bit == '1':
122             circuit.z(i)
123         circuit.measure(circuit.qubits[i], range(len(encrypted_message)))
124         simulator = Aer.get_backend('qasm_simulator')
125         job = execute(circuit, simulator, shots=1)
126         result = job.result()
127         counts = result.get_counts(circuit)
128         decrypted_message = list(counts.keys())[0]
129         return decrypted_message
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200

```

```

112 self.lineEditShift = QLineEdit()
113 vbox.addWidget(self.lineEditShift)
114
115 self.btnEncrypt = QPushButton('Encrypt', self)
116 self.btnEncrypt.setIcon(QIcon('encrypt.png'))
117 self.btnEncrypt.setStyleSheet('background-color: #ACAF58; color: black;')
118 self.btnEncrypt.clicked.connect(self.encrypt)
119 vbox.addWidget(self.btnEncrypt)
120
121 self.btnDecrypt = QPushButton('Decrypt', self)
122 self.btnDecrypt.setIcon(QIcon('decrypt.png'))
123 self.btnDecrypt.setStyleSheet('background-color: #2196F3; color: black;')
124 self.btnDecrypt.clicked.connect(self.decrypt)
125 vbox.addWidget(self.btnDecrypt)
126
127 self.btnSave = QPushButton('Save', self)
128 self.btnSave.setIcon(QIcon('save.png'))
129 self.btnSave.setStyleSheet('background-color: #4CAF50; color: black;')
130 self.btnSave.clicked.connect(self.save)
131 vbox.addWidget(self.btnSave)
132
133 self.btnOpen = QPushButton('Open', self)
134 self.btnOpen.setIcon(QIcon('open.png'))
135 self.btnOpen.setStyleSheet('background-color: #FF9800; color: black;')
136 self.btnOpen.clicked.connect(self.open)
137 vbox.addWidget(self.btnOpen)
138
139 self.labelResult = QLabel('Result')
140 vbox.addWidget(self.labelResult)
141
142 self.textEditResult = QTextEdit()
143 vbox.addWidget(self.textEditResult)
144
145 self.labelCreator = QLabel('Created by Reyhanssan Islamey - 2200018411')
146 # self.labelCreator.setStyleSheet('text-align: center;')
147 vbox.addWidget(self.labelCreator)
148
149 self.setGeometry(100, 100, 300, 300)
150 self.show()
151
152 def encrypt(self):
153     text = self.textEdit.toPlainText()
154     shift = int(self.lineEditShift.text())
155     encrypted_text = caesar_encrypt(text, shift)
156     binary = text_to_binary(encrypted_text)
157     key = get_key_from_cesar(shift)
158     key_qubit = most_frequent_bit(key)
159     encrypted_message = encrypt_message(binary, key)
160     self.textEditResult.setText(encrypted_message)
161
162 def decrypt(self):
163     encrypted_message = self.textEdit.toPlainText()
164     shift = int(self.lineEditShift.text())
165     key = get_key_from_cesar(shift)
166     key_qubit = most_frequent_bit(key)
167     decrypted_message = decrypt_message(encrypted_message, key_qubit)
168     text = binary_to_text(decrypted_message)
169     self.textEditResult.setText(text)
170
171 def save(self):
172     options = QFileDialog.Options()
173     options |= QFileDialog.DontUseRelativeDialog
174     filename, _ = QFileDialog.getSaveFileName(self, 'Save File', "", 'All Files (*);;Text Files (*.txt)', options=options)
175     if filename:
176         with open(filename, 'w') as f:
177             f.write(self.textEditResult.toPlainText())
178
179 def open(self):
180     options = QFileDialog.Options()
181     options |= QFileDialog.DontUseRelativeDialog
182     filename, _ = QFileDialog.getOpenFileName(self, 'Open File', "", 'All Files (*);;Text Files (*.txt)', options=options)
183     if filename:
184         with open(filename, 'r') as f:
185             self.textEditResult.setText(f.read())
186
187 if __name__ == '__main__':
188     import sys
189     app = QApplication(sys.argv)
190     ex = MyApp()
191     sys.exit(app.exec_())

```

```

171 def save(self):
172     options = QFileDialog.Options()
173     options |= QFileDialog.DontUseRelativeDialog
174     filename, _ = QFileDialog.getSaveFileName(self, 'Save File', "", 'All Files (*);;Text Files (*.txt)', options=options)
175     if filename:
176         with open(filename, 'w') as f:
177             f.write(self.textEditResult.toPlainText())
178
179 def open(self):
180     options = QFileDialog.Options()
181     options |= QFileDialog.DontUseRelativeDialog
182     filename, _ = QFileDialog.getOpenFileName(self, 'Open File', "", 'All Files (*);;Text Files (*.txt)', options=options)
183     if filename:
184         with open(filename, 'r') as f:
185             self.textEditResult.setText(f.read())
186
187 if __name__ == '__main__':
188     import sys
189     app = QApplication(sys.argv)
190     ex = MyApp()
191     sys.exit(app.exec_())

```

2. PENJELASAN

i. Library

Sesuai dengan instalasi library sebelumnya, berikut penjelasannya :



```
1 from qiskit import QuantumCircuit, transpile, assemble, Aer, execute
2 from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton, QTextEdit, QLineEdit, QLabel, QFileDialog
3 from PyQt5.QtGui import QIcon
4 from PyQt5.QtCore import Qt
```

1. Qiskit:

Qiskit adalah library Python yang digunakan untuk membangun program Quantum Computing. Library ini menyediakan fungsi-fungsi untuk membuat sirkuit kuantum, menjalankan simulasi kuantum, dan mengakses mesin kuantum fisik. Dalam program ini, Qiskit digunakan untuk membuat sirkuit kuantum dan menjalankan simulasi kuantum.

2. PyQt5:

PyQt5 adalah library Python yang digunakan untuk membuat antarmuka grafis pada program. Library ini menyediakan berbagai widget yang dapat digunakan untuk membuat tampilan program, seperti tombol, teks, dan kotak input. PyQt5 juga menyediakan fungsi-fungsi untuk memproses interaksi pengguna dengan program. Dalam program ini, PyQt5 digunakan untuk membuat antarmuka grafis pada program.

3. QTextEdit, QLineEdit, QLabel, QPushButton, QVBoxLayout, QFileDialog:

Kelima widget tersebut adalah bagian dari PyQt5. QTextEdit digunakan untuk membuat kotak teks pada program, QLineEdit digunakan untuk membuat kotak input, QLabel digunakan untuk membuat label pada program, QPushButton digunakan untuk membuat tombol pada program, QVBoxLayout digunakan untuk membuat layout vertikal pada program, dan QFileDialog digunakan untuk membuat jendela dialog untuk memilih file pada program.

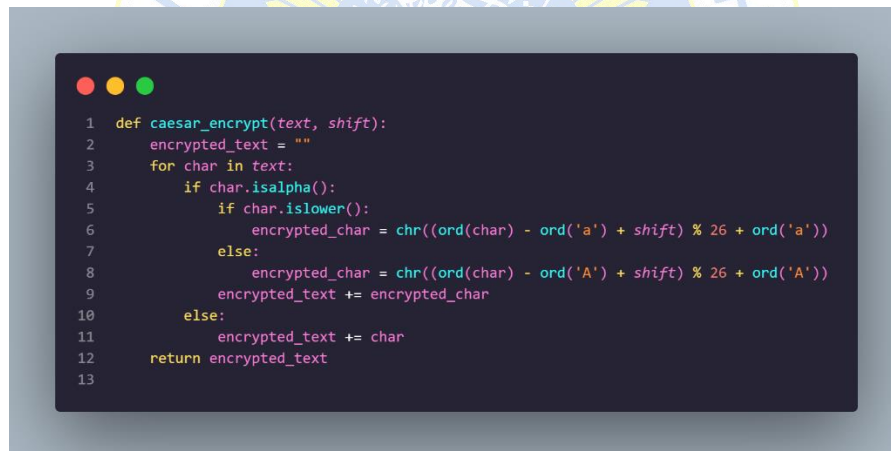
4. Aer:

Aer adalah library Python yang digunakan untuk menjalankan simulasi kuantum dengan cepat. Library ini menyediakan berbagai simulator kuantum yang dapat digunakan untuk menjalankan simulasi kuantum pada komputer. Dalam program ini, Aer digunakan untuk menjalankan simulasi kuantum pada komputer.

5. execute:

Fungsi execute adalah bagian dari Qiskit. Fungsi ini digunakan untuk menjalankan sirkuit kuantum pada simulator atau mesin kuantum. Dalam program ini, fungsi execute digunakan untuk menjalankan sirkuit kuantum pada simulasi kuantum dengan menggunakan Aer.

ii. Enkripsi Caesar



```
1 def caesar_encrypt(text, shift):
2     encrypted_text = ""
3     for char in text:
4         if char.isalpha():
5             if char.islower():
6                 encrypted_char = chr((ord(char) - ord('a') + shift) % 26 + ord('a'))
7             else:
8                 encrypted_char = chr((ord(char) - ord('A') + shift) % 26 + ord('A'))
9             encrypted_text += encrypted_char
10        else:
11            encrypted_text += char
12    return encrypted_text
13
```

Dalam implementasi program di atas, terdapat fungsi caesar_encrypt yang digunakan untuk melakukan enkripsi teks menggunakan metode Caesar cipher. Caesar cipher adalah metode enkripsi sederhana yang menggeser setiap huruf dalam teks sejauh nilai shift yang ditentukan.

Fungsi caesar_encrypt memiliki dua parameter, yaitu text yang merupakan teks yang akan dienkripsi, dan shift yang merupakan jumlah pergeseran yang akan diterapkan pada setiap

huruf. Selama proses enkripsi, variabel `encrypted_text` digunakan untuk menyimpan teks yang telah dienkripsi.

Pada setiap iterasi, kondisi `if` digunakan untuk memeriksa apakah karakter tersebut merupakan huruf. Jika karakter tersebut merupakan huruf kecil, karakter tersebut akan dienkripsi dengan menggeser nilai ASCII-nya menggunakan rumus $(\text{ord}(\text{char}) - \text{ord}('a') + \text{shift}) \% 26 + \text{ord}('a')$. Sedangkan jika karakter tersebut merupakan huruf besar, proses enkripsi yang sama dilakukan dengan menggunakan rumus $(\text{ord}(\text{char}) - \text{ord}('A') + \text{shift}) \% 26 + \text{ord}('A')$.

Jika karakter tersebut bukan huruf, maka karakter tersebut langsung ditambahkan ke dalam `encrypted_text` tanpa melakukan enkripsi. Setelah selesai mengiterasi semua karakter dalam teks, `encrypted_text` yang berisi teks yang telah dienkripsi dikembalikan sebagai hasil dari fungsi. Dengan menggunakan program ini, pengguna dapat dengan mudah melakukan enkripsi teks dengan metode Caesar cipher.

Berikut penjelasan per-line dari codenya :

- *`def caesar_encrypt(text, shift):`*

Ini adalah definisi fungsi `caesar_encrypt`, yang memiliki dua parameter yaitu `text` (teks yang akan dienkripsi) dan `shift` (jumlah pergeseran karakter pada teks).

- *`encrypted_text = ""`*

Ini adalah variabel `encrypted_text` yang berisi string kosong. Variabel ini akan digunakan untuk menyimpan hasil enkripsi teks.

- *`for char in text:`*

Ini adalah loop `for` yang digunakan untuk mengambil karakter satu per satu dari variabel `text`.

- *`if char.isalpha():`*

Ini adalah statement `if` yang digunakan untuk memeriksa apakah karakter yang diambil pada variabel `text` merupakan huruf.

- *`if char.islower():`*

Ini adalah statement if bersarang yang digunakan untuk memeriksa apakah karakter yang diambil merupakan huruf kecil.

- $encrypted_char = chr((ord(char) - ord('a') + shift) \% 26 + ord('a'))$

Ini adalah variabel `encrypted_char` yang berisi karakter yang telah dienkripsi. Proses enkripsi dilakukan dengan mengubah nilai ASCII dari karakter awal dengan menambahkan shift dan kemudian mengubah kembali ke karakter baru dengan menggunakan fungsi `chr`.

- *else:*

Ini adalah statement `else` yang akan dieksekusi jika karakter yang diambil bukan merupakan huruf kecil.

- $encrypted_char = chr((ord(char) - ord('A') + shift) \% 26 + ord('A'))$

Ini adalah variabel `encrypted_char` yang berisi karakter yang telah dienkripsi. Proses enkripsi dilakukan dengan mengubah nilai ASCII dari karakter awal dengan menambahkan shift dan kemudian mengubah kembali ke karakter baru dengan menggunakan fungsi `chr`.

- $encrypted_text += encrypted_char$

Ini adalah statement yang digunakan untuk menambahkan karakter yang telah dienkripsi ke dalam variabel `encrypted_text`.

- *else:*

Ini adalah statement `else` yang akan dieksekusi jika karakter yang diambil bukan merupakan huruf.

- $return encrypted_text$

Ini adalah statement yang digunakan untuk mengembalikan hasil enkripsi teks yang telah dienkripsi.

Simulasi hasil dari fungsi `caesar_encrypt('hello world', 3)`:

```
text = 'hello world'
shift = 3
encrypted_text = ""
for char in text:
    if char.isalpha():
        if char.islower():
            encrypted_char = chr((ord(char) - ord('a') + shift) % 26 + ord('a'))
        else:
            encrypted_char = chr((ord(char) - ord('A') + shift) % 26 + ord('A'))
        encrypted_text += encrypted_char
    else:
        encrypted_text += char
# encrypted_text = 'khoor zruog'
return encrypted_text
# Output: 'khoor zruog'
```

Dalam simulasi ini, kita memanggil fungsi `caesar_encrypt` dengan parameter `text = 'hello world'` dan `shift = 3`. Setelah dijalankan, fungsi tersebut menghasilkan output `'khoor zruog'` yang merupakan pesan yang telah dienkripsi dengan Caesar Cipher menggunakan pergeseran sebanyak 3 karakter.

iii. Text to Binary

```
1 def text_to_binary(text):
2     binary_string = ""
3     for char in text:
4         binary_char = format(ord(char), '08b')
5         binary_string += binary_char
6     return binary_string
```

Fungsi `text_to_binary` digunakan untuk mengonversi teks menjadi representasi biner dengan panjang 8-bit menggunakan ASCII. Dalam implementasi fungsi ini, terdapat satu parameter yaitu `text` yang akan diubah menjadi representasi biner.

Pada setiap iterasi, fungsi `ord()` digunakan untuk mengembalikan nilai ASCII dari karakter dalam teks. Kemudian, fungsi `format()` digunakan untuk mengonversi nilai ASCII

tersebut menjadi representasi biner dengan panjang 8-bit dan diisi dengan nol di depan apabila diperlukan. Representasi biner dari setiap karakter kemudian ditambahkan ke dalam variabel `binary_string`.

Setelah selesai mengiterasi semua karakter dalam teks, `binary_string` yang berisi representasi biner dari teks dikembalikan sebagai hasil dari fungsi. Dengan menggunakan fungsi `text_to_binary` ini, pengguna dapat dengan mudah mengonversi teks menjadi representasi biner yang dapat digunakan dalam operasi enkripsi atau dekripsi lainnya.

Berikut penjelasan per-line dari codenya :

- `def text_to_binary(text):`

Kode ini memulai definisi fungsi `text_to_binary` dengan satu parameter yaitu `text`, yang akan diubah menjadi representasi biner.

- `binary_string = ""`

Ini adalah variabel `binary_string` yang berisi string kosong. Variabel ini akan digunakan untuk menyimpan hasil konversi teks ke dalam representasi biner.

- `for char in text:`

Ini adalah loop `for` yang digunakan untuk mengambil karakter satu per satu dari variabel `text`.

- `binary_char = format(ord(char), '08b')`

Ini adalah variabel `binary_char` yang berisi karakter yang telah diubah menjadi representasi biner. Proses konversi dilakukan dengan mengubah nilai ASCII dari karakter awal menjadi representasi biner dengan menggunakan fungsi `format`.

- `binary_string += binary_char`

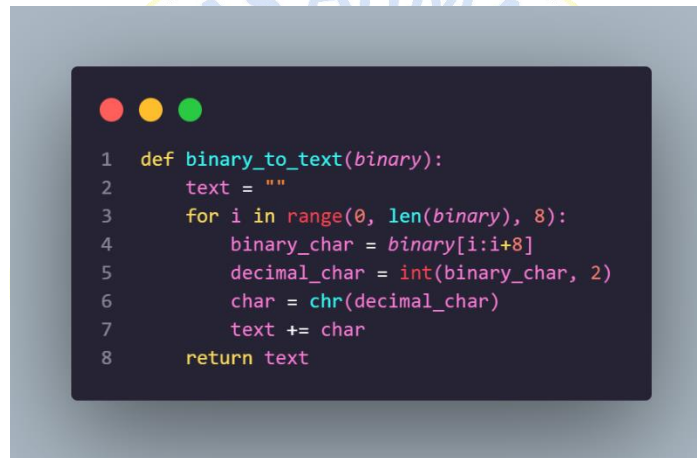
Ini adalah statement yang digunakan untuk menambahkan karakter yang telah diubah menjadi representasi biner ke dalam variabel `binary_string`.

- *return binary_string*

Ini adalah statement yang digunakan untuk mengembalikan hasil konversi teks ke dalam representasi biner yang telah disimpan di dalam variabel `binary_string`.

Apabila fungsi `text_to_binary("Hello, World!")` dijalankan, maka output yang dihasilkan adalah "01001000 01100101 01101100 01101100 01101111 00101100 00100000 01010111 01101111 01110010 01101100 01100100 00100001".

iv. Binary to Text



```
1 def binary_to_text(binary):
2     text = ""
3     for i in range(0, len(binary), 8):
4         binary_char = binary[i:i+8]
5         decimal_char = int(binary_char, 2)
6         char = chr(decimal_char)
7         text += char
8     return text
```

Fungsi `binary_to_text` digunakan untuk mengonversi representasi biner menjadi teks menggunakan nilai ASCII yang sesuai. Dalam implementasi fungsi ini, terdapat satu parameter yaitu `binary` yang merupakan representasi biner yang akan dikonversi menjadi teks.

Pada setiap iterasi, loop `for` digunakan untuk mengambil setiap blok 8 karakter dari representasi biner dan mengonversinya kembali ke dalam bentuk teks. Variabel `binary_char` diambil dari `binary` dengan menggunakan slicing `[i:i+8]`. Ini digunakan untuk mengambil 8 karakter biner yang sesuai dengan setiap karakter teks.

Kemudian, fungsi `int(binary_char, 2)` digunakan untuk mengonversi `binary_char` menjadi bilangan desimal dengan menggunakan basis 2 (biner). Setelah itu, fungsi `chr(decimal_char)` digunakan untuk mengonversi bilangan desimal kembali menjadi karakter

menggunakan nilai ASCII yang sesuai. Karakter yang dihasilkan kemudian ditambahkan ke dalam variabel `text`.

Setelah selesai mengiterasi semua blok 8 karakter dalam representasi biner, `text` yang berisi teks yang telah dikonversi dikembalikan sebagai hasil dari fungsi. Dengan menggunakan fungsi `binary_to_text` ini, pengguna dapat dengan mudah mengonversi representasi biner kembali menjadi teks yang dapat dibaca.

Berikut penjelasan per-linanya :

- `def binary_to_text(binary):`

Kode ini memulai definisi fungsi `binary_to_text` dengan satu parameter yaitu `binary`, yang akan diubah kembali ke dalam bentuk teks.

- `text = ""`

Ini adalah variabel `text` yang berisi string kosong. Variabel ini akan digunakan untuk menyimpan hasil konversi representasi biner ke dalam teks.

- `for i in range(0, len(binary), 8):`

Ini adalah loop `for` yang digunakan untuk mengambil setiap 8 karakter dari representasi biner dan mengonversinya kembali ke dalam bentuk teks.

- `binary_char = binary[i:i+8]`

Ini adalah variabel `binary_char` yang berisi 8 karakter dari representasi biner yang diambil pada setiap iterasi loop `for`.

- `decimal_char = int(binary_char, 2)`

Ini adalah variabel `decimal_char` yang berisi nilai desimal dari representasi biner yang telah diambil pada langkah sebelumnya.

- `char = chr(decimal_char)`

Ini adalah variabel `char` yang berisi karakter yang dihasilkan dari nilai desimal yang telah diambil pada langkah sebelumnya.

- `text += char`

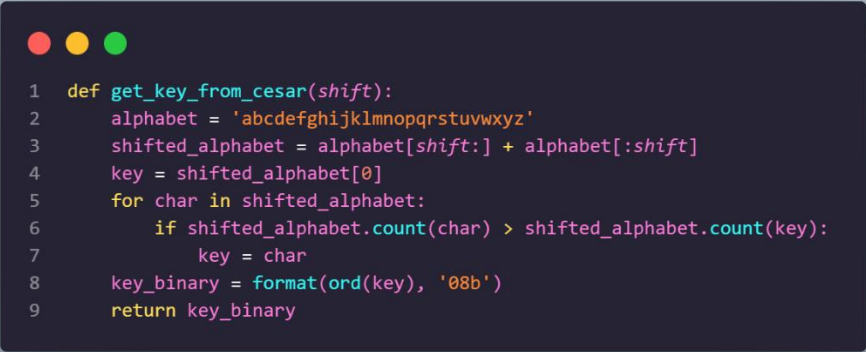
Ini adalah statement yang digunakan untuk menambahkan karakter yang telah dihasilkan ke dalam variabel `text`.

- `return text`

Ini adalah statement yang digunakan untuk mengembalikan hasil konversi representasi biner ke dalam teks yang telah disimpan di dalam variabel `text`.

Dengan menggunakan fungsi `binary_to_text` ini, pengguna dapat dengan mudah mengubah representasi biner ke dalam bentuk teks. Apabila fungsi `binary_to_text("01001000 01100101 01101100 01101100 01101111 00101100 00100000 01010111 01101111 01110010 01101100 01100100 00100001")` dijalankan, maka output yang dihasilkan adalah "Hello, World!".

v. Mendapatkan kunci dari Caesar



```

1 def get_key_from_cesar(shift):
2     alphabet = 'abcdefghijklmnopqrstuvwxyz'
3     shifted_alphabet = alphabet[shift:] + alphabet[:shift]
4     key = shifted_alphabet[0]
5     for char in shifted_alphabet:
6         if shifted_alphabet.count(char) > shifted_alphabet.count(key):
7             key = char
8     key_binary = format(ord(key), '08b')
9     return key_binary

```

Fungsi `get_key_from_cesar` digunakan untuk mendapatkan kunci dari metode enkripsi Caesar cipher dengan menggunakan `shift` yang telah ditentukan. Dalam implementasi fungsi ini, terdapat satu parameter yaitu `shift` yang merupakan jumlah pergeseran karakter yang digunakan dalam enkripsi Caesar cipher.

Pada awal implementasi fungsi, string alphabet yang berisi 26 karakter alfabet digunakan. Kemudian, `shifted_alphabet` diinisialisasi dengan memindahkan karakter alfabet sebanyak shift ke kanan. Dalam hal ini, karakter-karakter di belakang shift akan dipindahkan ke depan.

Setelah menginisialisasi `shifted_alphabet`, variabel `key` diinisialisasi dengan karakter pertama dari `shifted_alphabet`. Kemudian, loop for digunakan untuk mengiterasi setiap karakter dalam `shifted_alphabet`.

Pada setiap iterasi, statement if digunakan untuk membandingkan jumlah kemunculan karakter saat ini dengan jumlah kemunculan karakter pada variabel `key`. Jika jumlah kemunculan karakter saat ini lebih besar dari jumlah kemunculan karakter pada variabel `key`, maka variabel `key` akan diubah menjadi karakter saat ini.

Setelah selesai mengiterasi semua karakter dalam `shifted_alphabet`, variabel `key` yang berisi karakter paling sering muncul dikonversi menjadi representasi biner dengan menggunakan fungsi `format()`. Representasi biner dari karakter `key` kemudian dikembalikan sebagai hasil dari fungsi.

Berikut penjelasan per-lininya :

- `def get_key_from_cesar(shift):`

Fungsi `get_key_from_cesar` memiliki satu parameter yaitu `shift`, yang merupakan jumlah karakter yang digeser dalam Caesar cipher.

- `alphabet = 'abcdefghijklmnopqrstuvwxyz'`

Inisialisasi variabel `alphabet` sebagai string yang berisi 26 karakter alfabet

- `shifted_alphabet = alphabet[shift:] + alphabet[:shift]`

Menggeser karakter alfabet sebanyak shift ke kanan dengan menggunakan slicing pada variabel `alphabet` dan menyimpannya pada variabel `shifted_alphabet`.

- `key = shifted_alphabet[0]`

Inisialisasi variabel `key` dengan karakter pertama dalam `shifted_alphabet` sebagai kunci awal.

- *for char in shifted_alphabet:*

if shifted_alphabet.count(char) > shifted_alphabet.count(key):

key = char

Loop for digunakan untuk mengiterasi setiap karakter dalam shifted_alphabet. Pada setiap iterasi, kondisi if digunakan untuk memeriksa apakah jumlah kemunculan karakter saat ini lebih besar daripada jumlah kemunculan karakter kunci saat ini. Jika jumlah kemunculan karakter saat ini lebih besar, maka karakter saat ini akan menjadi kunci baru.

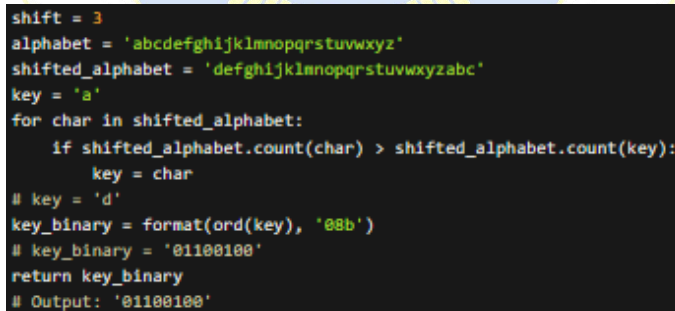
- *key_binary = format(ord(key), '08b')*

Konversi karakter kunci menjadi representasi biner dengan panjang 8-bit menggunakan fungsi format() dan menyimpannya pada variabel key_binary.

- *return key_binary*

Mengembalikan kunci dalam bentuk representasi biner sebagai hasil dari fungsi.

Simulasi hasil dari fungsi get_key_from_cesar(3):

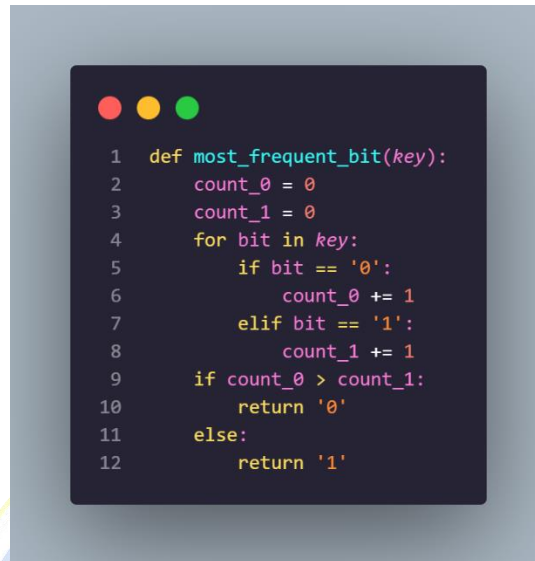


```

shift = 3
alphabet = 'abcdefghijklmnopqrstuvwxyz'
shifted_alphabet = 'defghijklmnopqrstuvwxyzabc'
key = 'a'
for char in shifted_alphabet:
    if shifted_alphabet.count(char) > shifted_alphabet.count(key):
        key = char
# key = 'd'
key_binary = format(ord(key), '08b')
# key_binary = '01100100'
return key_binary
# Output: '01100100'

```

Dalam simulasi ini, kita memanggil fungsi get_key_from_cesar dengan parameter shift = 3. Setelah dijalankan, fungsi tersebut menghasilkan output '01100100' yang merupakan kunci dalam representasi biner untuk Caesar cipher dengan pergeseran sebanyak 3 karakter.

vi. Menentukan kunci utama

```
1 def most_frequent_bit(key):
2     count_0 = 0
3     count_1 = 0
4     for bit in key:
5         if bit == '0':
6             count_0 += 1
7         elif bit == '1':
8             count_1 += 1
9     if count_0 > count_1:
10        return '0'
11    else:
12        return '1'
```

Fungsi `most_frequent_bit` digunakan untuk menentukan bit paling sering muncul dalam kunci representasi biner. Fungsi ini memiliki satu parameter yaitu `key` yang merupakan kunci representasi biner.

Dalam implementasi fungsi ini, terdapat dua variabel yaitu `count_0` dan `count_1` yang digunakan untuk menghitung jumlah kemunculan bit 0 dan bit 1 dalam kunci.

Loop `for` digunakan untuk mengiterasi setiap bit dalam kunci. Pada setiap iterasi, kondisi `if` digunakan untuk memeriksa apakah bit saat ini adalah 0 atau 1. Jika bit saat ini adalah 0, maka variabel `count_0` akan ditambah 1, dan jika bit saat ini adalah 1, maka variabel `count_1` akan ditambah 1.

Setelah selesai mengiterasi semua bit dalam kunci, kondisi `if` digunakan untuk memeriksa apakah jumlah kemunculan bit 0 lebih besar daripada jumlah kemunculan bit 1. Jika jumlah kemunculan bit 0 lebih besar, maka fungsi akan mengembalikan '0', dan jika jumlah kemunculan bit 1 lebih besar atau sama dengan jumlah kemunculan bit 0, maka fungsi akan mengembalikan '1'.

Berikut penjelasan per-linya :

- *def most_frequent_bit(key):*

Fungsi *most_frequent_bit* digunakan untuk menentukan bit paling sering muncul dalam kunci representasi biner. Fungsi ini memiliki satu parameter yaitu *key* yang merupakan kunci representasi biner.

- *count_0 = 0*

count_1 = 0

Inisialisasi dua variabel yaitu *count_0* dan *count_1* yang digunakan untuk menghitung jumlah kemunculan bit 0 dan bit 1 dalam kunci.

- *for bit in key:*

if bit == '0':

count_0 += 1

elif bit == '1':

count_1 += 1

Loop *for* digunakan untuk mengiterasi setiap bit dalam kunci. Pada setiap iterasi, kondisi *if* digunakan untuk memeriksa apakah bit saat ini adalah 0 atau 1. Jika bit saat ini adalah 0, maka variabel *count_0* akan ditambah 1, dan jika bit saat ini adalah 1, maka variabel *count_1* akan ditambah 1.

- *if count_0 > count_1:*

return '0'

else:

return '1'

Setelah selesai mengiterasi semua bit dalam kunci, kondisi *if* digunakan untuk memeriksa apakah jumlah kemunculan bit 0 lebih besar daripada jumlah kemunculan bit 1. Jika jumlah

kemunculan bit 0 lebih besar, maka fungsi akan mengembalikan '0', dan jika jumlah kemunculan bit 1 lebih besar atau sama dengan jumlah kemunculan bit 0, maka fungsi akan mengembalikan '1'.

Simulasi hasil dari fungsi `most_frequent_bit('01100100')`:

```
key = '01100100'
count_0 = 4
count_1 = 4
if count_0 > count_1:
    return '0'
else:
    return '1'
# Output: '0'
```

Dalam simulasi ini, kita memanggil fungsi `most_frequent_bit` dengan parameter `key = '01100100'`. Setelah dijalankan, fungsi tersebut menghasilkan output '0' yang merupakan bit paling sering muncul dalam kunci representasi biner tersebut.

vii. Enkripsi Kuantum

```
1 def encrypt_message(message, key):
2     circuit = QuantumCircuit(len(message), len(message))
3     for i, bit in enumerate(message):
4         if bit == '1':
5             circuit.x(i)
6     for i, bit in enumerate(key):
7         if bit == '1':
8             circuit.h(i)
9     circuit.measure(range(len(message)), range(len(message)))
10    simulator = Aer.get_backend('qasm_simulator')
11    job = execute(circuit, simulator, shots=1)
12    result = job.result()
13    counts = result.get_counts(circuit)
14    encrypted_message = list(counts.keys())[0]
15    return encrypted_message
```

Kode tersebut adalah implementasi dari quantum cryptography, yaitu ilmu yang memanfaatkan sifat-sifat mekanika kuantum untuk melakukan tugas-tugas kriptografi, seperti mengirimkan informasi rahasia dengan cara yang tidak dapat diintip oleh pihak lain. Kode tersebut menggunakan metode yang disebut quantum key distribution (QKD), yaitu metode untuk mengirimkan rangkaian acak yang rahasia, yang disebut kunci, dengan menggunakan serangkaian foton.


Kode tersebut terdiri dari beberapa bagian:

- Pertama, kode tersebut mendefinisikan fungsi `encrypt_message(message, key)`, yang menerima dua parameter: `message` dan `key`. `message` adalah pesan yang ingin dienkripsi, dan `key` adalah kunci yang digunakan untuk enkripsi. Kedua parameter tersebut harus berupa string biner, yaitu string yang hanya terdiri dari angka 0 dan 1.
- Kedua, kode tersebut membuat sebuah objek `QuantumCircuit` dengan panjang `len(message)`, yaitu jumlah bit dalam pesan. Objek ini merepresentasikan rangkaian kuantum yang digunakan untuk enkripsi. Objek ini juga memiliki jumlah qubit dan bit klasik yang sama dengan panjang pesan.
- Ketiga, kode tersebut melakukan iterasi pada setiap bit dalam pesan dan kunci. Jika bit pesan adalah 1, maka kode tersebut menerapkan operasi `x` pada qubit yang sesuai. Operasi `x` adalah operasi kuantum yang membalikkan nilai qubit dari 0 menjadi 1 atau sebaliknya. Jika bit kunci adalah 1, maka kode tersebut menerapkan operasi `h` pada qubit yang sesuai. Operasi `h` adalah operasi kuantum yang membuat qubit menjadi superposisi dari 0 dan 1, yaitu keadaan di mana qubit memiliki kemungkinan 50% untuk menjadi 0 atau 1 ketika diukur.
- Keempat, kode tersebut melakukan pengukuran pada semua qubit dan menyimpan hasilnya ke dalam bit klasik. Pengukuran adalah proses di mana qubit menentukan nilai akhirnya sebagai 0 atau 1 secara acak berdasarkan keadaan superposisinya.
- Kelima, kode tersebut menggunakan objek `Aer.get_backend('qasm_simulator')` untuk membuat sebuah simulator kuantum. Simulator ini digunakan untuk menjalankan rangkaian kuantum secara virtual dan menghasilkan hasilnya.

- Keenam, kode tersebut menggunakan fungsi `execute(circuit, simulator, shots=1)` untuk menjalankan rangkaian kuantum pada simulator dengan jumlah percobaan sebanyak 1 kali. Fungsi ini mengembalikan sebuah objek `job`, yang merepresentasikan pekerjaan simulasi yang sedang berjalan atau sudah selesai.
- Ketujuh, kode tersebut menggunakan metode `result()` pada objek `job` untuk mendapatkan hasil simulasi. Hasil ini berupa sebuah objek `result`, yang menyimpan informasi tentang hasil pengukuran dan statistiknya.
- Kedelapan, kode tersebut menggunakan metode `get_counts(circuit)` pada objek `result` untuk mendapatkan jumlah kemunculan dari setiap hasil pengukuran. Metode ini mengembalikan sebuah dictionary, di mana kunci adalah string biner yang merepresentasikan hasil pengukuran, dan nilai adalah jumlah kemunculan dari string tersebut.
- Kesembilan, kode tersebut mengambil string biner pertama dari dictionary tersebut dengan menggunakan ekspresi `list(counts.keys())[0]`. String ini adalah pesan terenkripsi yang dihasilkan dari rangkaian kuantum. Kode tersebut mengembalikan string ini sebagai nilai balik dari fungsi.

Apabila kode ini dijalankan, maka akan menghasilkan simulasi dari rangkaian kuantum yang digunakan untuk mengenkripsi pesan. Hasil dari simulasi tersebut akan menghasilkan pesan terenkripsi yang dihasilkan dari rangkaian kuantum. Namun, hasil dari simulasi ini akan berbeda-beda setiap kali dijalankan karena qubit memiliki kemungkinan 50% untuk menjadi 0 atau 1 ketika diukur. Oleh karena itu, hasil dari simulasi ini tidak dapat diprediksi secara pasti.

viii. Deskripsi Kuantum



```

1  def decrypt_message(encrypted_message, key_qubit):
2      circuit = QuantumCircuit(len(encrypted_message), len(encrypted_message))
3      for i, bit in enumerate(key_qubit):
4          if bit == '1':
5              circuit.h(i)
6      for i, bit in enumerate(encrypted_message):
7          if bit == '1':
8              circuit.x(i)
9      circuit.measure(range(len(encrypted_message)), range(len(encrypted_message)))
10     simulator = Aer.get_backend('qasm_simulator')
11     job = execute(circuit, simulator, shots=1)
12     result = job.result()
13     counts = result.get_counts(circuit)
14     decrypted_message = list(counts.keys())[0]
15     return decrypted_message

```

Kode ini adalah fungsi `decrypt_message(encrypted_message, key_qubit)`, yang menerima dua parameter: `encrypted_message` dan `key_qubit`. `encrypted_message` adalah pesan yang telah dienkripsi dengan menggunakan fungsi `encrypt_message(message, key)`, dan `key_qubit` adalah kunci yang digunakan untuk enkripsi. Kedua parameter ini harus berupa string biner, yaitu string yang hanya terdiri dari angka 0 dan 1.

Kode ini melakukan proses yang hampir sama dengan fungsi `encrypt_message(message, key)`, tetapi dengan urutan yang terbalik. Kode ini bertujuan untuk mengembalikan pesan asli dari pesan terenkripsi dengan menggunakan kunci yang sama.

Kode ini terdiri dari beberapa bagian:

- Pertama, kode ini membuat sebuah objek `QuantumCircuit` dengan panjang `len(encrypted_message)`, yaitu jumlah bit dalam pesan terenkripsi. Objek ini merepresentasikan rangkaian kuantum yang digunakan untuk dekripsi. Objek ini juga memiliki jumlah qubit dan bit klasik yang sama dengan panjang pesan terenkripsi.
- Kedua, kode ini melakukan iterasi pada setiap bit dalam kunci dan pesan terenkripsi. Jika bit kunci adalah 1, maka kode ini menerapkan operasi `h` pada qubit yang sesuai.

Operasi h adalah operasi kuantum yang membuat qubit menjadi superposisi dari 0 dan 1, yaitu keadaan di mana qubit memiliki kemungkinan 50% untuk menjadi 0 atau 1 ketika diukur. Jika bit pesan terenkripsi adalah 1, maka kode ini menerapkan operasi x pada qubit yang sesuai. Operasi x adalah operasi kuantum yang membalikkan nilai qubit dari 0 menjadi 1 atau sebaliknya.

- Ketiga, kode ini melakukan pengukuran pada semua qubit dan menyimpan hasilnya ke dalam bit klasik. Pengukuran adalah proses di mana qubit menentukan nilai akhirnya sebagai 0 atau 1 secara acak berdasarkan keadaan superposisinya.
- Keempat, kode ini menggunakan objek `Aer.get_backend('qasm_simulator')` untuk membuat sebuah simulator kuantum. Simulator ini digunakan untuk menjalankan rangkaian kuantum secara virtual dan menghasilkan hasilnya.
- Kelima, kode ini menggunakan fungsi `execute(circuit, simulator, shots=1)` untuk menjalankan rangkaian kuantum pada simulator dengan jumlah percobaan sebanyak 1 kali. Fungsi ini mengembalikan sebuah objek `job`, yang merepresentasikan pekerjaan simulasi yang sedang berjalan atau sudah selesai.
- Keenam, kode ini menggunakan metode `result()` pada objek `job` untuk mendapatkan hasil simulasi. Hasil ini berupa sebuah objek `result`, yang menyimpan informasi tentang hasil pengukuran dan statistiknya.
- Ketujuh, kode ini menggunakan metode `get_counts(circuit)` pada objek `result` untuk mendapatkan jumlah kemunculan dari setiap hasil pengukuran. Metode ini mengembalikan sebuah dictionary, di mana kunci adalah string biner yang merepresentasikan hasil pengukuran, dan nilai adalah jumlah kemunculan dari string tersebut.
- Kedelapan, kode ini mengambil string biner pertama dari dictionary tersebut dengan menggunakan ekspresi `list(counts.keys())[0]`. String ini adalah pesan asli yang dihasilkan dari rangkaian kuantum. Kode ini mengembalikan string ini sebagai nilai balik dari fungsi.

Sama halnya dengan enkripsi yang telah dilakukan, apabila kode ini dijalankan akan menghasilkan simulasi dari rangkaian kuantum yang digunakan untuk mendeskripsikan pesan. Hasil dari simulasi tersebut akan menghasilkan pesan terdeskripsi yang dihasilkan dari rangkaian kuantum. Namun, hasil dari simulasi ini akan berbeda-beda setiap kali dijalankan karena qubit memiliki kemungkinan 50% untuk menjadi 0 atau 1 ketika diukur. Oleh karena itu, hasil dari simulasi ini tidak dapat diprediksi secara pasti.

Jadi besar kemungkinan terdapat beberapa symbol yang berbeda dengan pesan pengirim dan penerima. Hal ini akan saya jelaskan nanti di bagian output.

ix. Class

a. Init Jendela



Dalam kode di atas saya mendefinisikan kelas MyApp yang merupakan turunan dari QWidget. Kelas ini bertanggung jawab untuk membuat dan mengatur antarmuka pengguna aplikasi.

Metode `__init__` adalah metode konstruktor yang akan dipanggil saat objek kelas MyApp dibuat. Di dalam metode ini, saya memanggil metode `initUI` untuk menginisialisasi antarmuka pengguna.

b. Jendela windows

```

1  def initUI(self):
2      self.setStyleSheet("background-color: #F0F0F0; color: #333333;")
3      self.setWindowTitle('Program Enkripsi dan Dekripsi Kuantum Caesar Cipher (QCC) v1.0')
4      self.setWindowIcon(QIcon('decryption.png'))
5
6      vbox = QVBoxLayout()
7      self.setLayout(vbox)
8
9      self.label = QLabel('Masukkan teks:')
10     vbox.addWidget(self.label)
11
12     self.textEdit = QTextEdit()
13     vbox.addWidget(self.textEdit)
14
15     self.labelShift = QLabel('Masukkan jumlah pergeseran:')
16     vbox.addWidget(self.labelShift)
17
18     self.lineEditShift = QLineEdit()
19     vbox.addWidget(self.lineEditShift)
20
21     self.btnEncrypt = QPushButton(' Enkripsi', self)
22     self.btnEncrypt.setIcon(QIcon('encrypt.png'))
23     self.btnEncrypt.setStyleSheet("background-color: #4CAF50; color: black;")
24     self.btnEncrypt.clicked.connect(self.encrypt)
25     vbox.addWidget(self.btnEncrypt)
26
27     self.btnDecrypt = QPushButton(' Dekripsi', self)
28     self.btnDecrypt.setIcon(QIcon('decrypt.png'))
29     self.btnDecrypt.setStyleSheet("background-color: #2196F3; color: black;")
30     self.btnDecrypt.clicked.connect(self.decrypt)
31     vbox.addWidget(self.btnDecrypt)
32
33     self.btnSave = QPushButton(' Simpan', self)
34     self.btnSave.setIcon(QIcon('save.png'))
35     self.btnSave.setStyleSheet("background-color: #f44336; color: black;")
36     self.btnSave.clicked.connect(self.save)
37     vbox.addWidget(self.btnSave)
38
39     self.btnOpen = QPushButton(' Buka', self)
40     self.btnOpen.setIcon(QIcon('open.png'))
41     self.btnOpen.setStyleSheet("background-color: #FF9800; color: black;")
42     self.btnOpen.clicked.connect(self.open)
43     vbox.addWidget(self.btnOpen)
44
45     self.labelResult = QLabel('Hasil:')
46     vbox.addWidget(self.labelResult)
47
48     self.textEditResult = QTextEdit()
49     vbox.addWidget(self.textEditResult)
50
51     self.labelCreator = QLabel('Created by Reyhanssan Islamey - 2200018411')
52     self.labelCreator.setAlignment(Qt.AlignCenter)
53     vbox.addWidget(self.labelCreator)
54
55     self.setGeometry(300, 300, 300, 200)
56     self.show()

```

Code di atas adalah bagian dari program Python yang menggunakan PyQt5 untuk membuat antarmuka pengguna grafis (GUI) untuk aplikasi enkripsi dan dekripsi kuantum Caesar Cipher (QCC) v1.0. Berikut adalah penjelasan singkat dari setiap baris code:


- `def initUI(self):` Mendefinisikan sebuah fungsi yang bernama `initUI` yang menerima parameter `self`, yaitu referensi ke objek kelas yang memanggil fungsi ini.
- `self.setStyleSheet("background-color: #F0F0F0; color: #333333;")`: Mengatur gaya lembar untuk objek `self`, yaitu jendela utama aplikasi, dengan warna latar belakang abu-abu dan warna teks hitam.
- `self.setWindowTitle('Program Enkripsi dan Dekripsi Kuantum Caesar Cipher (QCC) v1.0')`: Mengatur judul jendela dengan teks yang diberikan.
- `self.setWindowIcon(QIcon('decryption.png'))`: Mengatur ikon jendela dengan gambar yang diberikan.
- `vbox = QVBoxLayout()`: Membuat sebuah objek `QVBoxLayout`, yaitu sebuah layout vertikal yang mengatur widget secara berurutan dari atas ke bawah.
- `self.setLayout(vbox)`: Mengatur layout jendela utama dengan objek `vbox` yang telah dibuat sebelumnya.
- `self.label = QLabel('Masukkan teks:')`: Membuat sebuah objek `QLabel`, yaitu sebuah widget yang menampilkan teks, dengan teks yang diberikan.
- `vbox.addWidget(self.label)`: Menambahkan widget `label` ke layout `vbox`.
- `self.textEdit = QTextEdit()`: Membuat sebuah objek `QTextEdit`, yaitu sebuah widget yang memungkinkan pengguna untuk mengedit teks multi-baris.
- `vbox.addWidget(self.textEdit)`: Menambahkan widget `textEdit` ke layout `vbox`.
- `self.labelShift = QLabel('Masukkan jumlah pergeseran:')`: Membuat sebuah objek `QLabel` dengan teks yang diberikan.
- `vbox.addWidget(self.labelShift)`: Menambahkan widget `labelShift` ke layout `vbox`.

- `self.lineEditShift = QLineEdit()`: Membuat sebuah objek `QLineEdit`, yaitu sebuah widget yang memungkinkan pengguna untuk mengedit teks satu baris.
- `vbox.addWidget(self.lineEditShift)`: Menambahkan widget `lineEditShift` ke layout `vbox`.
- `self.btnEncrypt = QPushButton(' Enkripsi', self)`: Membuat sebuah objek `QPushButton`, yaitu sebuah widget yang menampilkan tombol dengan teks dan ikon, dengan teks dan ikon yang diberikan. Parameter `self` menunjukkan bahwa tombol ini adalah anak dari jendela utama.
- `self.btnEncrypt.setIcon(QIcon('encrypt.png'))`: Mengatur ikon tombol dengan gambar yang diberikan.
- `self.btnEncrypt.setStyleSheet("background-color: #4CAF50; color: black;")`: Mengatur gaya lembar untuk tombol dengan warna latar belakang hijau dan warna teks hitam.
- `self.btnEncrypt.clicked.connect(self.encrypt)`: Menghubungkan sinyal `clicked` dari tombol, yaitu sinyal yang dipancarkan ketika tombol diklik, dengan slot `encrypt` dari objek `self`, yaitu fungsi yang akan dieksekusi ketika sinyal dipancarkan. Fungsi `encrypt` ini harus didefinisikan di tempat lain dalam program untuk melakukan proses enkripsi teks.
- `vbox.addWidget(self.btnEncrypt)`: Menambahkan widget `btnEncrypt` ke layout `vbox`.
- `self.btnDecrypt = QPushButton(' Dekripsi', self)`: Membuat sebuah objek `QPushButton` dengan teks dan ikon yang diberikan. Parameter `self` menunjukkan bahwa tombol ini adalah anak dari jendela utama.
- `self.btnDecrypt.setIcon(QIcon('decrypt.png'))`: Mengatur ikon tombol dengan gambar yang diberikan.
- `self.btnDecrypt.setStyleSheet("background-color: #2196F3; color: black;")`: Mengatur gaya lembar untuk tombol dengan warna latar belakang biru dan warna teks hitam.

- `self.btnDecrypt.clicked.connect(self.decrypt)`: Menghubungkan sinyal clicked dari tombol dengan slot decrypt dari objek self, yaitu fungsi yang akan dieksekusi ketika sinyal dipancarkan. Fungsi decrypt ini harus didefinisikan di tempat lain dalam program untuk melakukan proses dekripsi teks.
- `vbox.addWidget(self.btnDecrypt)`: Menambahkan widget btnDecrypt ke layout vbox.
- `self.btnSave = QPushButton(' Simpan', self)`: Membuat sebuah objek QPushButton dengan teks dan ikon yang diberikan. Parameter self menunjukkan bahwa tombol ini adalah anak dari jendela utama.
- `self.btnSave.setIcon(QIcon('save.png'))`: Mengatur ikon tombol dengan gambar yang diberikan.
- `self.btnSave.setStyleSheet("background-color: #f44336; color: black;")`: Mengatur gaya lembar untuk tombol dengan warna latar belakang merah dan warna teks hitam.
- `self.btnSave.clicked.connect(self.save)`: Menghubungkan sinyal clicked dari tombol dengan slot save dari objek self, yaitu fungsi yang akan dieksekusi ketika sinyal dipancarkan. Fungsi save ini harus didefinisikan di tempat lain dalam program untuk menyimpan teks hasil ke dalam file.
- `vbox.addWidget(self.btnSave)`: Menambahkan widget btnSave ke layout vbox.
- `self.btnOpen = QPushButton(' Buka', self)`: Membuat sebuah objek QPushButton dengan teks dan ikon yang diberikan. Parameter self menunjukkan bahwa tombol ini adalah anak dari jendela utama.
- `self.btnOpen.setIcon(QIcon('open.png'))`: Mengatur ikon tombol dengan gambar yang diberikan.
- `self.btnOpen.setStyleSheet("background-color: #FF9800; color: black;")`: Mengatur gaya lembar untuk tombol dengan warna latar belakang oranye dan warna teks hitam.
- `self.btnOpen.clicked.connect(self.open)`: Menghubungkan sinyal clicked dari tombol dengan slot open dari objek self, yaitu fungsi yang akan dieksekusi ketika sinyal

dipancarkan. Fungsi open ini harus didefinisikan di tempat lain dalam program untuk membuka file yang berisi teks hasil dan menampilkannya di widget textEditResult.

- `vbox.addWidget(self.btnOpen);` Menambahkan widget btnOpen ke layout vbox.
- `self.labelResult = QLabel('Hasil:');` Membuat sebuah objek QLabel dengan teks yang diberikan.
- `vbox.addWidget(self.labelResult);` Menambahkan widget labelResult ke layout vbox.
- `self.textEditResult = QTextEdit();` Membuat sebuah objek QTextEdit, yaitu sebuah widget yang memungkinkan pengguna untuk mengedit teks multi-baris. Widget ini akan menampilkan teks hasil enkripsi atau dekripsi yang dilakukan oleh program.
- `vbox.addWidget(self.textEditResult);` Menambahkan widget textEditResult ke layout vbox.
- `self.labelCreator = QLabel('Created by Reyhanssan Islamey - 2200018411');` Membuat sebuah objek QLabel dengan teks yang diberikan. Teks ini menunjukkan nama pembuat program dan nomor induk mahasiswa (NIM).
- `self.labelCreator.setAlignment(Qt.AlignCenter);` Mengatur penjumlahan teks pada label menjadi tengah.
- `vbox.addWidget(self.labelCreator);` Menambahkan widget labelCreator ke layout vbox.
- `self.setGeometry(300, 300, 300, 200);` Mengatur geometri jendela utama dengan parameter x, y, lebar, dan tinggi yang diberikan. Parameter x dan y menentukan posisi jendela relatif terhadap layar, sedangkan parameter lebar dan tinggi menentukan ukuran jendela dalam piksel.
- `self.show();` Menampilkan jendela utama ke layar.

x. Memanggil enkripsi

```
1 def encrypt(self):
2     text = self.textEdit.toPlainText()
3     shift = int(self.lineEditShift.text())
4     encrypted_text = caesar_encrypt(text, shift)
5     binary = text_to_binary(encrypted_text)
6     key = get_key_from_cesar(shift)
7     key_qubit = most_frequent_bit(key)
8     encrypted_message = encrypt_message(binary, key)
9     self.textEditResult.setText(encrypted_message)
```

Metode `encrypt()` pada kode di atas adalah sebuah fungsi yang digunakan untuk mengenkripsi teks yang telah dimasukkan oleh pengguna dengan menggunakan algoritma Caesar Cipher yang telah dimodifikasi dengan teknologi kuantum. Fungsi ini terdiri dari beberapa langkah yang dilakukan secara berurutan untuk memastikan keamanan pesan yang dikirimkan. Pertama, teks yang telah dimasukkan oleh pengguna diambil menggunakan metode `toPlainText()`, kemudian jumlah pergeseran yang telah dimasukkan diambil menggunakan metode `text()` dan diubah menjadi integer menggunakan fungsi `int()`.

Selanjutnya, teks yang telah dimasukkan oleh pengguna dienkripsi menggunakan fungsi `caesar_encrypt()` dengan menggunakan jumlah pergeseran yang telah dimasukkan. Hasil enkripsi kemudian diubah menjadi bilangan biner menggunakan fungsi `text_to_binary()`, dan kunci yang digunakan untuk enkripsi diambil menggunakan fungsi `get_key_from_cesar()`. Setelah itu, kunci qubit diambil menggunakan fungsi `most_frequent_bit()` dan pesan yang telah diubah menjadi bilangan biner dienkripsi menggunakan fungsi `encrypt_message()` dengan menggunakan kedua kunci tersebut. Hasil enkripsi kemudian ditampilkan dalam teks edit untuk ditampilkan kepada pengguna.

Berikut adalah penjelasan singkat dari setiap baris teks:

- `text = self.textEdit.toPlainText()`: Mendapatkan teks yang dimasukkan oleh pengguna di widget `textEdit` dan menyimpannya dalam variabel `text`.

- `shift = int(self.lineEditShift.text())`: Mendapatkan jumlah pergeseran yang dimasukkan oleh pengguna di widget `lineEditShift` dan mengubahnya menjadi bilangan bulat, lalu menyimpannya dalam variabel `shift`.
- `encrypted_text = caesar_encrypt(text, shift)`: Memanggil fungsi `caesar_encrypt` yang didefinisikan di tempat lain dalam program untuk melakukan enkripsi teks menggunakan metode Caesar Cipher dengan jumlah pergeseran yang diberikan, lalu menyimpan hasilnya dalam variabel `encrypted_text`.
- `binary = text_to_binary(encrypted_text)`: Memanggil fungsi `text_to_binary` yang didefinisikan di tempat lain dalam program untuk mengubah teks terenkripsi menjadi kode biner, lalu menyimpan hasilnya dalam variabel `binary`.
- `key = get_key_from_cesar(shift)`: Memanggil fungsi `get_key_from_cesar` yang didefinisikan di tempat lain dalam program untuk mendapatkan kunci enkripsi dari jumlah pergeseran yang diberikan, lalu menyimpan hasilnya dalam variabel `key`.
- `key_qubit = most_frequent_bit(key)`: Memanggil fungsi `most_frequent_bit` yang didefinisikan di tempat lain dalam program untuk mendapatkan bit yang paling sering muncul dalam kunci enkripsi, lalu menyimpan hasilnya dalam variabel `key_qubit`. Bit ini akan digunakan sebagai qubit kunci untuk melakukan enkripsi kuantum.
- `encrypted_message = encrypt_message(binary, key)`: Memanggil fungsi `encrypt_message` yang didefinisikan di tempat lain dalam program untuk melakukan enkripsi kuantum pada kode biner menggunakan qubit kunci, lalu menyimpan hasilnya dalam variabel `encrypted_message`. Fungsi ini menggunakan modul Qiskit untuk membuat dan menjalankan sirkuit kuantum.
- `self.textEditResult.setText(encrypted_message)`: Menampilkan pesan terenkripsi di widget `textEditResult`.

xi. Memanggil Deskripsi

```
1 def decrypt(self):  
2     encrypted_message = self.textEdit.toPlainText()  
3     shift = int(self.lineEditShift.text())  
4     key = get_key_from_cesar(shift)  
5     key_qubit = most_frequent_bit(key)  
6     decrypted_message = decrypt_message(encrypted_message, key_qubit)  
7     text = binary_to_text(decrypted_message)  
8     decrypted_text = caesar_encrypt(text, -shift)  
9     self.textEditResult.setText(decrypted_text)
```

Metode `decrypt()` pada kode di atas adalah sebuah fungsi yang digunakan untuk mendekripsi pesan yang telah dienkripsi menggunakan algoritma Caesar Cipher yang telah dimodifikasi dengan teknologi kuantum. Fungsi ini terdiri dari beberapa langkah yang dilakukan secara berurutan untuk memastikan pesan yang telah dienkripsi dapat didekripsi dengan benar. Pertama, fungsi ini mengambil pesan yang telah dienkripsi oleh pengguna dari teks edit menggunakan metode `toPlainText()`. Selanjutnya, jumlah pergeseran yang telah digunakan untuk mengenkripsi pesan diambil dari line edit menggunakan metode `text()` dan diubah menjadi integer menggunakan fungsi `int()`.

Setelah mendapatkan pesan yang telah dienkripsi dan jumlah pergeseran yang digunakan, fungsi `decrypt()` menggunakan fungsi `get_key_from_cesar()` untuk mendapatkan kunci enkripsi yang digunakan dalam pengenkripsian pesan. Selanjutnya, fungsi ini menggunakan fungsi `most_frequent_bit()` untuk mendapatkan kunci qubit yang digunakan dalam pengenkripsian kuantum. Kemudian, pesan yang telah dienkripsi di-dekripsi menggunakan fungsi `decrypt_message()` dengan menggunakan kunci qubit yang telah didapatkan. Hasil dekripsi kemudian diubah kembali menjadi teks menggunakan fungsi `binary_to_text()`.

Terakhir, setelah mendapatkan teks yang telah di-dekripsi, fungsi `decrypt()` menggunakan fungsi `caesar_encrypt()` untuk mendekripsi teks tersebut dengan menggunakan jumlah pergeseran yang telah digunakan untuk mengenkripsi pesan dengan tanda negatif. Hasil dekripsi kemudian ditampilkan dalam teks edit menggunakan metode `setText()`.

Berikut adalah penjelasan singkat dari setiap baris teks:

- `encrypted_message = self.textEdit.toPlainText()`: Mendapatkan pesan terenkripsi yang dimasukkan oleh pengguna di widget `textEdit` dan menyimpannya dalam variabel `encrypted_message`.
- `shift = int(self.lineEditShift.text())`: Mendapatkan jumlah pergeseran yang dimasukkan oleh pengguna di widget `lineEditShift` dan mengubahnya menjadi bilangan bulat, lalu menyimpannya dalam variabel `shift`.
- `key = get_key_from_cesar(shift)`: Memanggil fungsi `get_key_from_cesar` yang didefinisikan di tempat lain dalam program untuk mendapatkan kunci dekripsi dari jumlah pergeseran yang diberikan, lalu menyimpan hasilnya dalam variabel `key`.
- `key_qubit = most_frequent_bit(key)`: Memanggil fungsi `most_frequent_bit` yang didefinisikan di tempat lain dalam program untuk mendapatkan bit yang paling sering muncul dalam kunci dekripsi, lalu menyimpan hasilnya dalam variabel `key_qubit`. Bit ini akan digunakan sebagai qubit kunci untuk melakukan dekripsi kuantum.
- `decrypted_message = decrypt_message(encrypted_message, key_qubit)`: Memanggil fungsi `decrypt_message` yang didefinisikan di tempat lain dalam program untuk melakukan dekripsi kuantum pada pesan terenkripsi menggunakan qubit kunci, lalu menyimpan hasilnya dalam variabel `decrypted_message`. Fungsi ini menggunakan modul `Qiskit` untuk membuat dan menjalankan sirkuit kuantum.
- `text = binary_to_text(decrypted_message)`: Memanggil fungsi `binary_to_text` yang didefinisikan di tempat lain dalam program untuk mengubah kode biner menjadi teks, lalu menyimpan hasilnya dalam variabel `text`.
- `decrypted_text = caesar_encrypt(text, -shift)`: Memanggil fungsi `caesar_encrypt` yang didefinisikan di tempat lain dalam program untuk melakukan dekripsi teks

menggunakan metode Caesar Cipher dengan jumlah pergeseran negatif, lalu menyimpan hasilnya dalam variabel `decrypted_text`.

- `self.textEditResult.setText(decrypted_text)`: Menampilkan teks terdekripsi di widget `textEditResult`.

xii. Save File



```

1 def save(self):
2     options = QFileDialog.Options()
3     options |= QFileDialog.DontUseNativeDialog
4     fileName, _ = QFileDialog.getSaveFileName(self, "Simpan file", "", "All Files (*);;Text Files (*.txt)", options=options)
5     if fileName:
6         with open(fileName, 'w') as f:
7             f.write(self.textEditResult.toPlainText())

```

Metode `save()` pada kode di atas adalah sebuah fungsi yang digunakan untuk menyimpan hasil enkripsi atau dekripsi pesan dalam sebuah file. Fungsi ini menggunakan `QFileDialog` untuk memunculkan dialog box yang memungkinkan pengguna untuk memilih lokasi dan nama file yang ingin disimpan. Setelah pengguna memilih lokasi dan nama file, fungsi ini membuka file tersebut menggunakan fungsi `open()` dengan mode `'w'` yang mengindikasikan bahwa file akan disimpan dalam mode write.

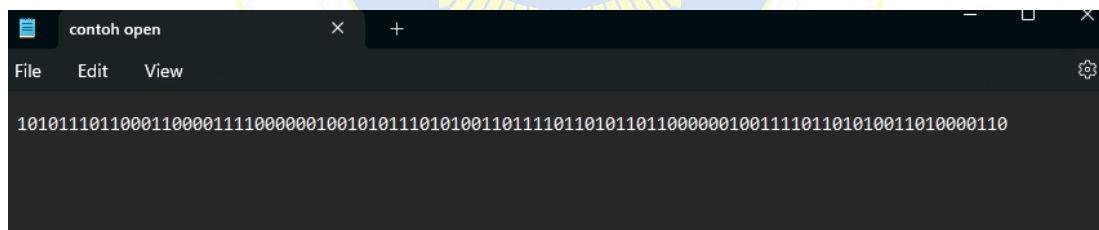
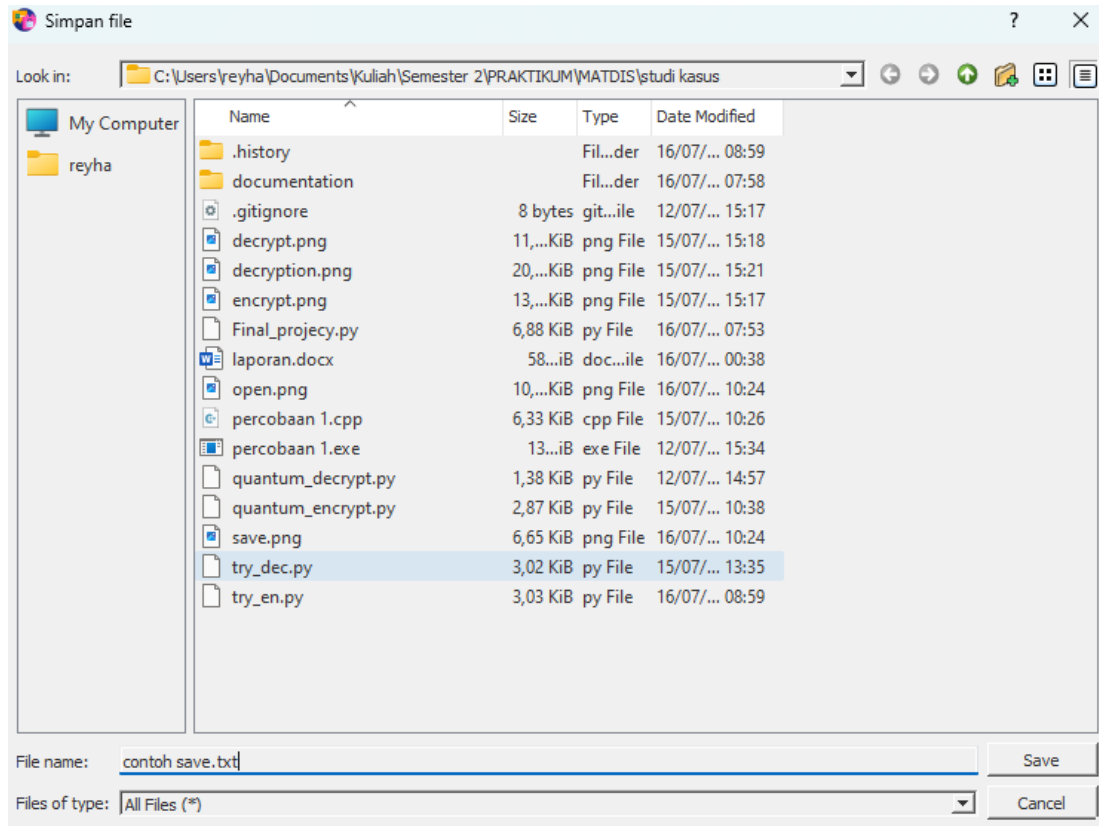
Selanjutnya, fungsi `save()` menggunakan fungsi `write()` untuk menulis teks yang telah dienkripsi atau didekripsi ke dalam file tersebut. Teks yang akan disimpan berasal dari teks edit yang terdapat pada program menggunakan metode `toPlainText()`. Dengan cara ini, pengguna dapat menyimpan hasil enkripsi atau dekripsi pesan dengan mudah dan aman dalam sebuah file.

Berikut adalah penjelasan singkat dari setiap baris teks:

- `options = QFileDialog.Options()`: Membuat sebuah objek `QFileDialog.Options`, yaitu sebuah kelas yang menyimpan berbagai pilihan untuk dialog file.

- `options |= QFileDialog.DontUseNativeDialog`: Mengatur pilihan untuk tidak menggunakan dialog file bawaan sistem operasi, melainkan menggunakan dialog file Qt sendiri.
- `fileName, _ = QFileDialog.getSaveFileName(self, "Simpan file", "", "All Files ();Text Files (.txt)", options=options)`: Memanggil fungsi statis `QFileDialog.getSaveFileName` yang menampilkan dialog file untuk memilih nama file yang akan disimpan. Fungsi ini menerima parameter berikut:
 - `self`: Menunjukkan bahwa dialog file ini adalah anak dari jendela utama.
 - `"Simpan file"`: Menunjukkan judul dialog file.
 - `""`: Menunjukkan direktori awal yang ditampilkan di dialog file. Jika kosong, maka direktori awal adalah direktori saat ini.
 - `"All Files ();Text Files (.txt)"`: Menunjukkan filter file yang dapat dipilih oleh pengguna. Filter ini menentukan jenis file yang dapat ditampilkan dan disimpan di dialog file. Dalam kasus ini, filternya adalah semua file () atau file teks (.txt).
 - `options=options`: Menunjukkan pilihan yang digunakan untuk dialog file, yaitu objek `options` yang telah dibuat sebelumnya.
- Fungsi ini mengembalikan dua nilai, yaitu nama file yang dipilih oleh pengguna dan filter file yang digunakan. Kedua nilai ini disimpan dalam variabel `fileName` dan `_`, dengan `_` menunjukkan bahwa nilai tersebut tidak digunakan.
- `if fileName`: Memeriksa apakah variabel `fileName` tidak kosong, yaitu pengguna telah memilih nama file dan tidak membatalkan dialog file.
- `with open(fileName, 'w') as f`: Membuka file dengan nama yang diberikan dalam mode tulis ('w'), dan menyimpan objek file dalam variabel `f`. Perintah `with` akan menutup file secara otomatis setelah blok kode selesai dieksekusi.
- `f.write(self.textEditResult.toPlainText())`: Menulis teks hasil yang ditampilkan di widget `textEditResult` ke dalam file.

Contoh file yang di simpan :



#note saya ganti nama untuk contoh berikutnya

xiii. Open file

```

1  def open(self):
2      options = QFileDialog.Options()
3      options |= QFileDialog.DontUseNativeDialog
4      fileName, _ = QFileDialog.getOpenFileName(self, "Buka file", "", "All Files (*);;Text files (*.txt)", options=options)
5      if fileName:
6          with open(fileName, 'r') as f:
7              self.textEditResult.setText(f.read())

```

Metode `open()` pada kode di atas adalah sebuah fungsi yang digunakan untuk membuka file yang berisi teks yang akan dienkripsi atau didekripsi. Fungsi ini menggunakan `QFileDialog` untuk memunculkan dialog box yang memungkinkan pengguna untuk memilih file yang ingin dibuka. Setelah pengguna memilih file yang ingin dibuka, fungsi ini membuka file tersebut menggunakan fungsi `open()` dengan mode `'r'` yang mengindikasikan bahwa file akan dibuka dalam mode `read`.

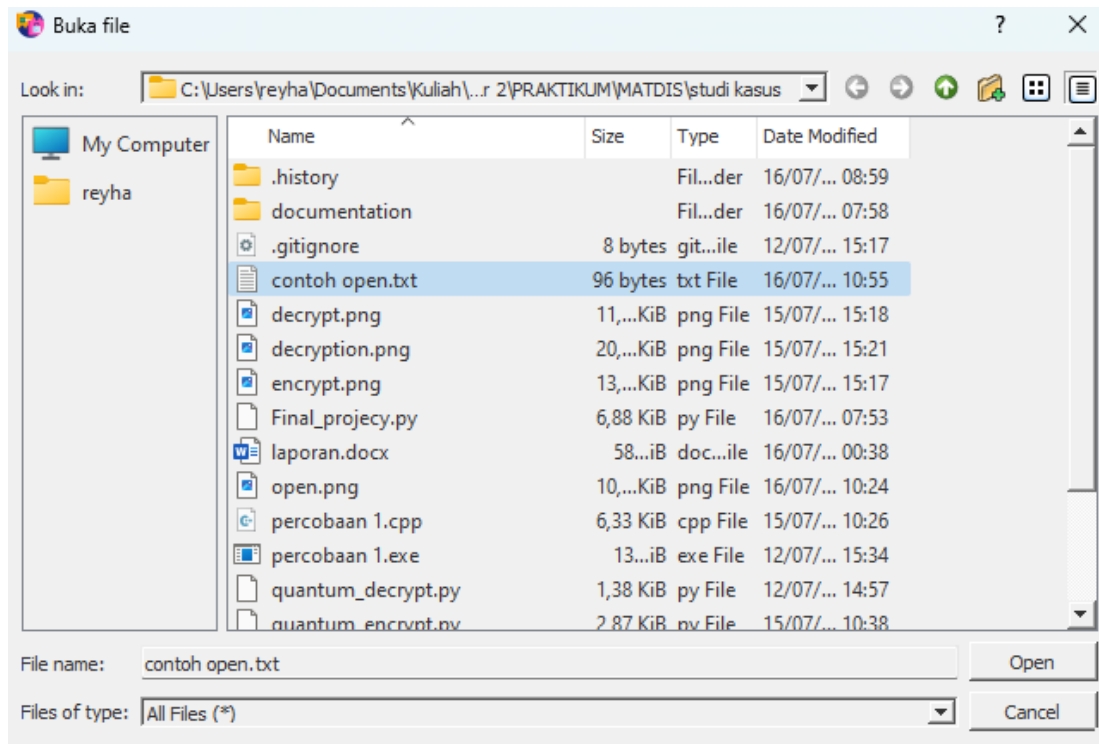
Kemudian, fungsi `open()` menggunakan fungsi `read()` untuk membaca teks yang terdapat pada file yang telah dibuka. Teks yang telah dibaca kemudian ditampilkan pada teks edit menggunakan metode `setText()`. Dengan cara ini, pengguna dapat membuka file yang berisi teks yang ingin dienkripsi atau didekripsi dengan mudah dan aman.

Berikut adalah penjelasan singkat dari setiap baris teks:

- `options = QFileDialog.Options()`: Membuat sebuah objek `QFileDialog.Options`, yaitu sebuah kelas yang menyimpan berbagai pilihan untuk dialog file.
- `options |= QFileDialog.DontUseNativeDialog`: Mengatur pilihan untuk tidak menggunakan dialog file bawaan sistem operasi, melainkan menggunakan dialog file Qt sendiri.
- `fileName, _ = QFileDialog.getOpenFileName(self, "Buka file", "", "All Files (*);;Text Files (*.txt)", options=options)`: Memanggil fungsi statis `QFileDialog.getOpenFileName` yang menampilkan dialog file untuk memilih nama file yang akan dibuka. Fungsi ini menerima parameter berikut:
 - `self`: Menunjukkan bahwa dialog file ini adalah anak dari jendela utama.


- “Buka file”: Menunjukkan judul dialog file.
- “”: Menunjukkan direktori awal yang ditampilkan di dialog file. Jika kosong, maka direktori awal adalah direktori saat ini.
- “All Files (.);;Text Files (.txt)”: Menunjukkan filter file yang dapat dipilih oleh pengguna. Filter ini menentukan jenis file yang dapat ditampilkan dan dibuka di dialog file. Dalam kasus ini, filturnya adalah semua file () atau file teks (.txt).
- options=options: Menunjukkan pilihan yang digunakan untuk dialog file, yaitu objek options yang telah dibuat sebelumnya.
- Fungsi ini mengembalikan dua nilai, yaitu nama file yang dipilih oleh pengguna dan filter file yang digunakan. Kedua nilai ini disimpan dalam variabel fileName dan _, dengan _ menunjukkan bahwa nilai tersebut tidak digunakan.
- if fileName: Memeriksa apakah variabel fileName tidak kosong, yaitu pengguna telah memilih nama file dan tidak membatalkan dialog file.
- with open(fileName, ‘r’) as f: Membuka file dengan nama yang diberikan dalam mode baca (‘r’), dan menyimpan objek file dalam variabel f. Perintah with akan menutup file secara otomatis setelah blok kode selesai dieksekusi.
- self.textEditResult.setText(f.read()): Membaca seluruh isi file dan menampilkannya di widget textEditResult.

Contoh file yang di



Hasil:

```
1010111011000110000111100000010010101110
1010011011110110101101100000010011110110
1010011010000110|
```


xiv. Main

```
1 if __name__ == '__main__':
2     import sys
3     app = QApplication(sys.argv)
4     ex = MyApp()
5     sys.exit(app.exec_())
```

Kode tersebut adalah bagian dari sebuah program Python yang digunakan untuk membangun aplikasi desktop yang dapat melakukan enkripsi dan dekripsi pesan menggunakan algoritma Caesar Cipher yang telah dimodifikasi dengan teknologi kuantum. Kode tersebut merupakan bagian dari blok utama program yang digunakan untuk menginisialisasi dan menjalankan aplikasi desktop tersebut.

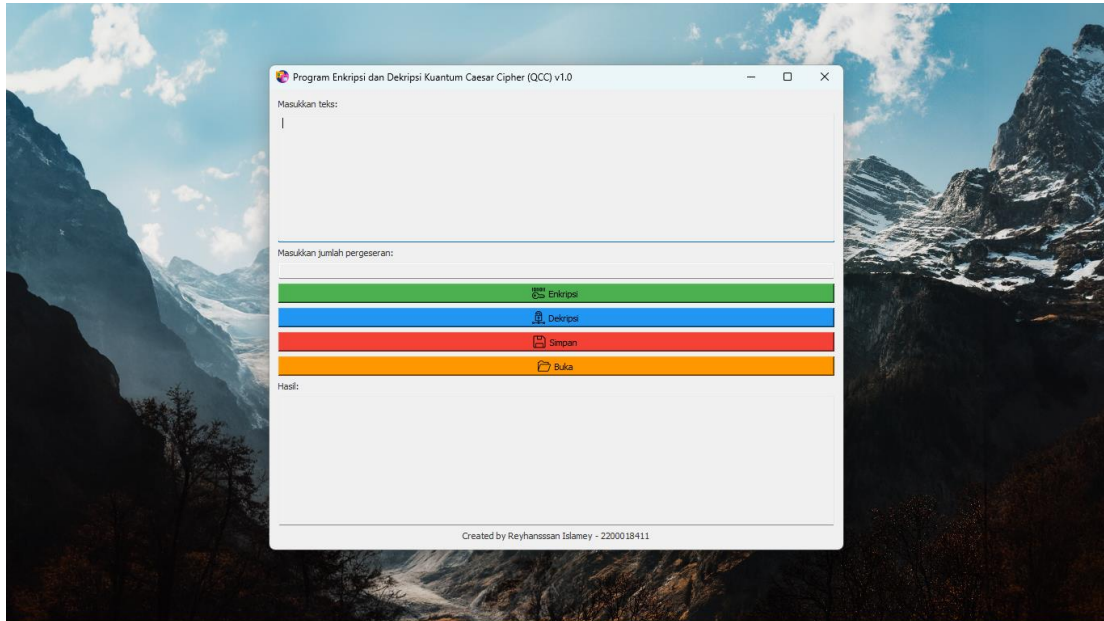
Pertama, kode tersebut menggunakan fungsi `name` untuk menentukan apakah program tersebut dijalankan sebagai program utama atau sebagai modul yang diimpor oleh program lain. Jika program tersebut dijalankan sebagai program utama, maka program akan melanjutkan ke blok selanjutnya. Jika program tersebut dijalankan sebagai modul yang diimpor oleh program lain, maka program tidak akan melanjutkan ke blok selanjutnya.

Selanjutnya, program menggunakan fungsi `QApplication` untuk menginisialisasi aplikasi desktop. Fungsi ini digunakan untuk membuat objek aplikasi yang digunakan untuk mengatur dan menjalankan aplikasi desktop. Selanjutnya, program menggunakan objek `MyApp()` untuk membuat objek utama aplikasi desktop. Objek ini digunakan untuk mengatur dan menampilkan elemen-elemen visual pada aplikasi desktop, seperti tombol, teks edit, dan lain sebagainya.

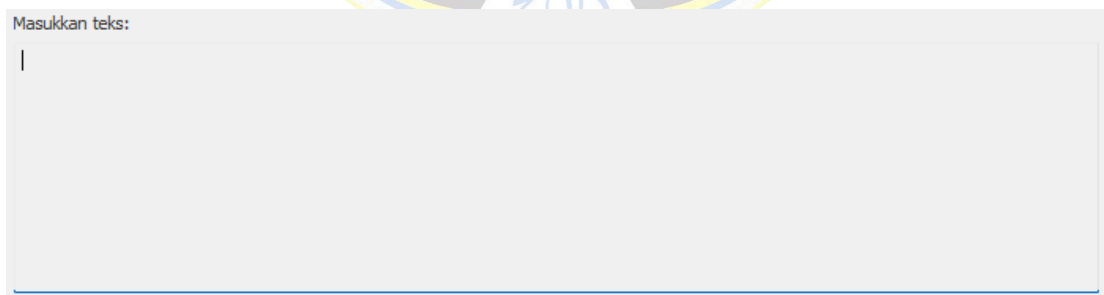
Terakhir, program menggunakan fungsi `sys.exit()` untuk menutup aplikasi desktop saat pengguna menekan tombol keluar pada jendela aplikasi. Fungsi ini memastikan bahwa aplikasi desktop ditutup dengan benar dan tidak berjalan di latar belakang. Dengan demikian, program dapat dijalankan dengan benar dan pengguna dapat melakukan enkripsi dan dekripsi pesan dengan aman dan mudah.

3. OUTPUT

Program Enkripsi dan Dekripsi Kuantum Caesar Cipher (QCC) v1.0 memiliki antarmuka pengguna yang sederhana dan mudah digunakan. Antarmuka ini terdiri dari beberapa elemen, di antaranya adalah teks edit, line edit, tombol, label, dan ikon.



Elemen pertama pada antarmuka adalah teks edit. Teks edit digunakan untuk memasukkan teks yang akan dienkripsi atau didekripsi. Teks edit pada program ini memiliki tampilan yang sederhana dan mudah digunakan.



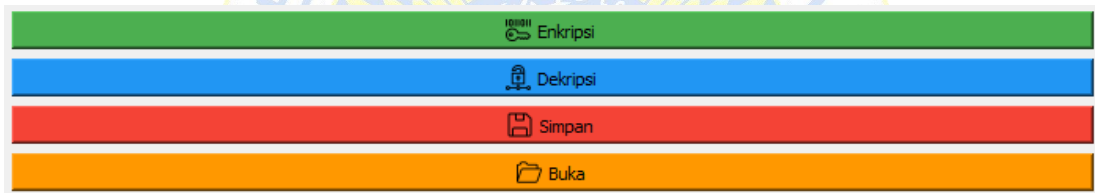
Selanjutnya, terdapat line edit yang digunakan untuk memasukkan jumlah pergeseran yang akan digunakan dalam proses enkripsi atau dekripsi. Line edit ini memiliki tampilan yang sederhana dan mudah digunakan.

Masukkan jumlah pergeseran:

Tombol pada antarmuka digunakan untuk melakukan enkripsi, dekripsi, menyimpan hasil, dan membuka file. Tombol tersebut memiliki icon yang jelas dan mudah dimengerti, serta warna yang berbeda untuk membedakan setiap tombol.

Label pada antarmuka digunakan untuk memberikan instruksi pada pengguna, seperti "Masukkan teks" dan "Masukkan jumlah pergeseran". Label juga digunakan untuk menampilkan hasil enkripsi atau dekripsi, serta untuk menampilkan informasi pembuat program.

Ikon pada antarmuka digunakan untuk memberikan tampilan yang lebih menarik dan mudah dimengerti oleh pengguna. Ikon tersebut melambangkan tindakan yang akan dilakukan oleh pengguna, seperti mengenkripsi, mendekripsi, menyimpan, dan membuka file.



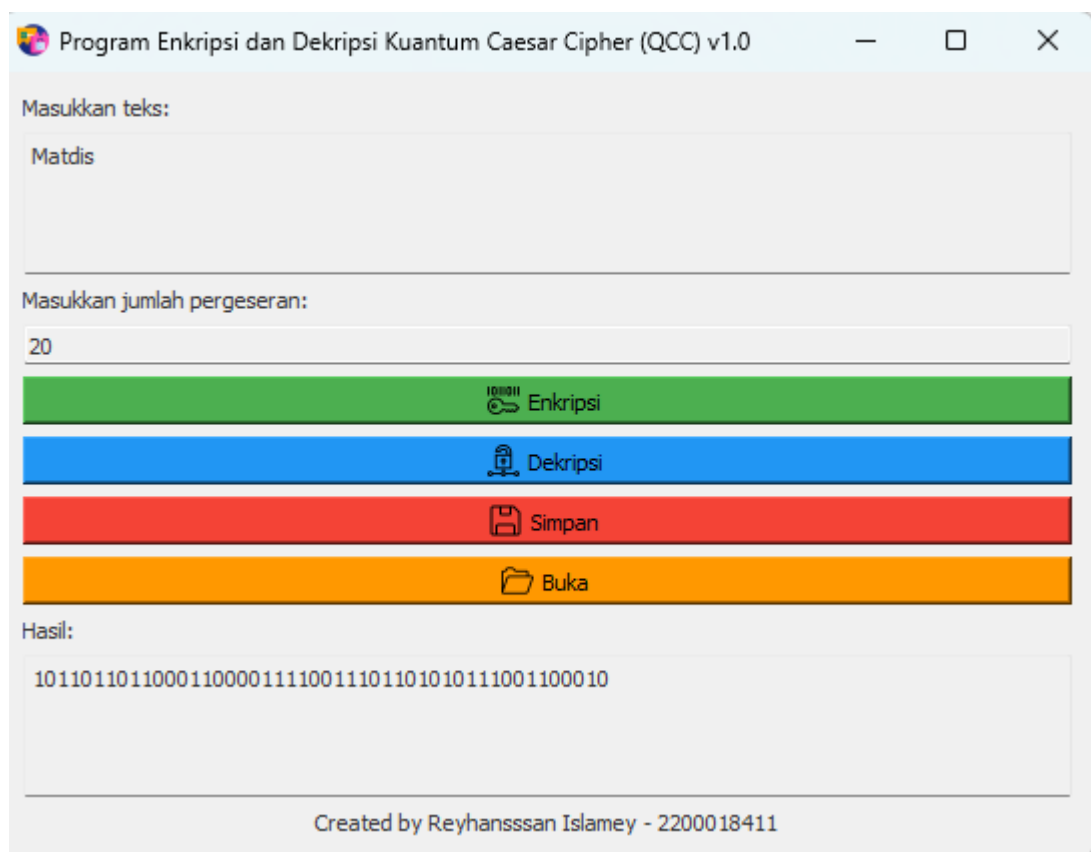
Dengan antarmuka pengguna yang sederhana dan mudah digunakan, program Enkripsi dan Dekripsi Kuantum Caesar Cipher (QCC) v1.0 dapat digunakan oleh siapa saja, baik itu pengguna yang sudah terbiasa dengan teknologi kuantum maupun pengguna yang baru mengenal teknologi ini.

Hasil:

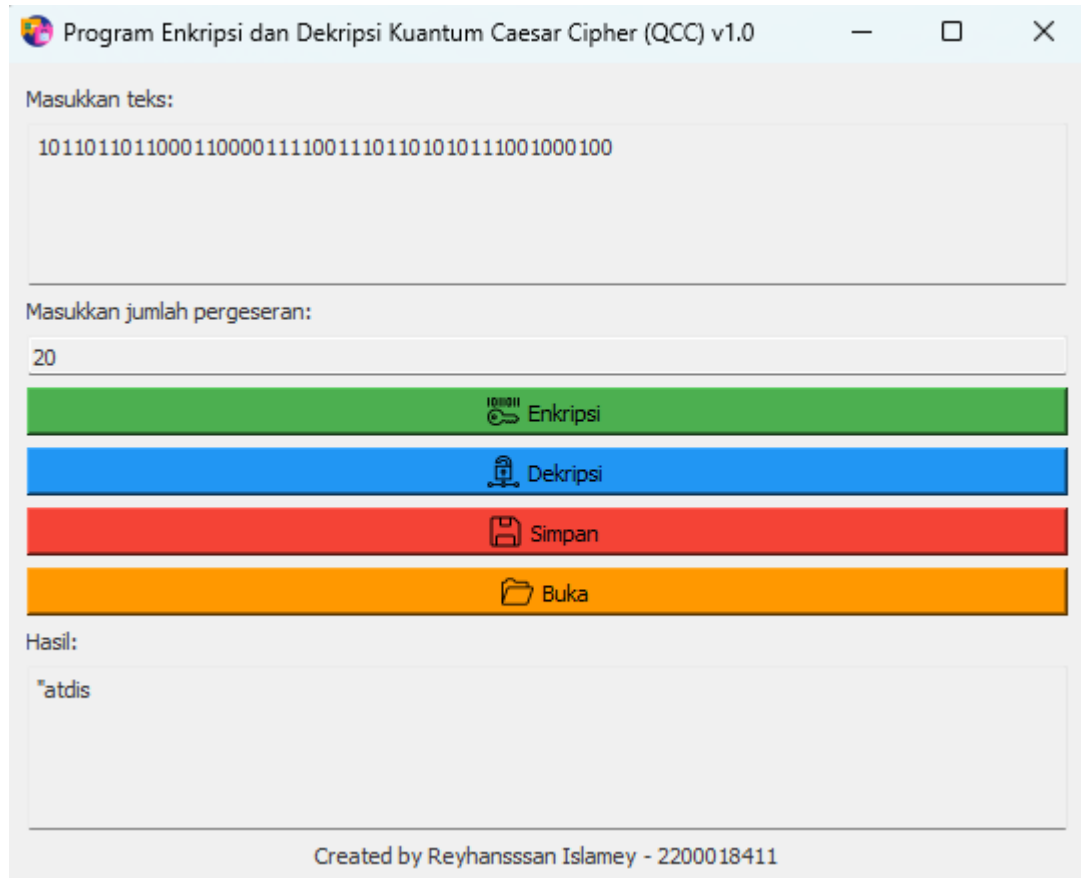
Created by Reyhanssan Islamey - 2200018411

Untuk menjalankan program ini, Anda perlu memasukkan teks yang akan dienkripsi atau didekripsi pada teks edit dan memasukkan jumlah pergeseran pada line edit. Setelah itu, Anda dapat menekan tombol "Enkripsi" atau "Dekripsi" untuk memulai proses. Hasil enkripsi atau dekripsi akan ditampilkan pada teks edit yang tersedia. Selain itu, Anda juga dapat menyimpan hasil enkripsi atau dekripsi pada file atau membuka file yang sudah tersimpan sebelumnya.

Contoh enkripsi :



Contoh deskripsi :

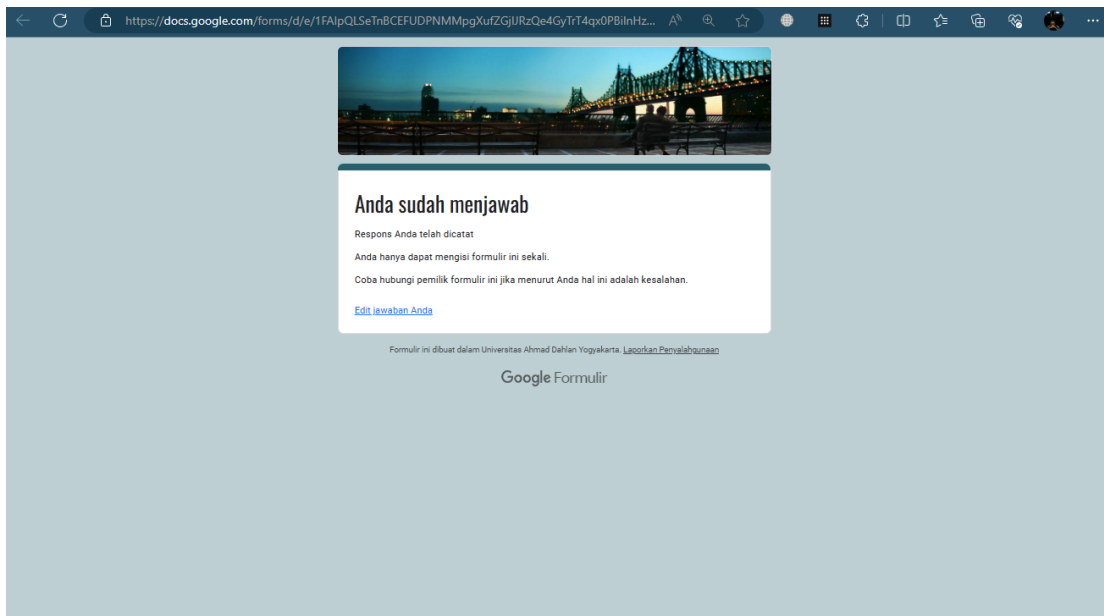


Dapat di lihat bahwa salah satu symbol hasil dari deskripsi dari pesan yang sudah terenkripsi memiliki perbedaan. Perbedaan antara pesan biasa dan pesan terenkripsi dalam enkripsi kuantum menggunakan Python disebabkan oleh sifat probabilitas qubit. Qubit dapat berada dalam superposisi keadaan, artinya dapat berada dalam kedua keadaan 0 dan 1 pada saat yang sama. Ketika sebuah qubit diukur, ia akan jatuh ke salah satu dari dua keadaan tersebut dengan probabilitas yang ditentukan oleh keadaan kuantum qubit tersebut. Hal ini berarti hasil pengukuran tidak dapat diprediksi dengan pasti, sehingga hasil dari proses enkripsi akan berbeda setiap kali program dijalankan (Andelita et al., 2021). Rumus yang digunakan dalam enkripsi kuantum tergantung pada algoritma enkripsi kuantum yang digunakan. Terdapat beberapa algoritma enkripsi kuantum, seperti quantum key distribution (QKD) dan quantum permutation pad (QPP), masing-masing dengan rumus dan metode enkripsi yang berbeda. Namun, perlu dicatat bahwa enkripsi kuantum masih dalam tahap awal

pengembangan, dan pengembangan komputer kuantum masih terus berlangsung. National Institute of Standards and Technology (NIST) saat ini sedang bekerja untuk menstandarisasi algoritma kriptografi kunci publik tahan quantum untuk mengurangi risiko yang ditimbulkan oleh komputer kuantum terhadap kriptografi kunci public (Koch et al., n.d.).

E. LAMPIRAN

1) Bukti feedback :



2) Sumber referensi dan bacaan serta inspirasi

Andelita, N., Sudiarta, W., Dian, D., & Kurniawidi, W. (2021). Penerapan Algoritma Kuantum Variational Quantum Eigensolver (VQE) untuk Menentukan Energi Keadaan Dasar Dimer Helium. In *Jurnal Fisika* (Vol. 11, Issue 2).
<https://journal.unnes.ac.id/nju/index.php/jf/index>

Koch, D., Patel, S., Wessing, L., & Alsing, P. M. (n.d.). *Fundamentals In Quantum Algorithms: A Tutorial Series Using Qiskit Continued*.

3) Link Github Project

<https://github.com/rzarey/Quantum-Encrypt-for-Matdis-Project-2.git>