Implementasi Algoritma RSA, ElGamal, Paillier, ECC

Laporan Tugas 4

Diajukan Untuk Memenuhi Tugas 4 IF4020 Kriptografi

Semester I 2021/2022

Disusun oleh

Reihan Andhika Putra **(13519043)**

Karel Renaldi **(13519180)**

TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2021

BAB I

Implementasi Program

Bahasa Pemrograman : **Python**

Framework : **Flask**

1. RSA.py

```python
import utils
import sympy
import random
import typing

class RSA_Crypt():
    def __init__(self):
        self.math = utils.Math()

    def set_public_key(self, e: int, n: int):
        self.e = e
        self.n = n

    def set_private_key(self, d: int, n: int):
        self.d = d
        self.n = n

    def generate_rsa_key(self, key_size) -> typing.Tuple[int, int, int]:
        # Create two prime number, p and q.
        p = sympy.randprime(pow(2, key_size - 1) + 1, pow(2, key_size) - 1)
        q = sympy.randprime(pow(2, key_size - 1) + 1, pow(2, key_size) - 1)

        # Calculate n and totient(n).
        n = p * q
        totient_n = (p - 1) * (q - 1)

        # Create e such as coprime with totient(n).
        finish = False
        while(not(finish)):
            e = random.randrange(pow(2, key_size - 1), pow(2, key_size))
            if(self.math.isCoprime(e, totient_n)):
                finish = True

        # Calculate d.
        d = self.math.modinv(e, totient_n)

        # Set e, d, n
        self.e = e
        self.d = d
        self.n = n

        # Return e, d, n.
        return e, d, n
```

```python
def generate_rsa_key_manual(self, p, q, e):
    if(not(self.math.isPrime(p))):
        raise Exception('p is not prime')
    if(not(self.math.isPrime(q))):
        raise Exception('q is not prime')
    if(self.math.isCoprime(e, (p - 1)*(q - 1))):
        raise Exception('e not coprime with totient(n)')

    # Calculate n and totient(n).
    n = p * q
    totient_n = (p - 1) * (q - 1)

    # Calculate d.
    d = self.math.modinv(e, totient_n)

    # Set e, d, n
    self.e = e
    self.d = d
    self.n = n

    # Return e, d, n.
    return e, d, n


def encrypt(self, plain_text: str, e: int, n: int) -> str:
    max_length = (len(str(plain_text)) - 1) // 3
    messages_int =  utils.plaintextToArrInt(plain_text, max_length)

    res = []
    for message in messages_int:
        temp = pow(message, e, n)
        res.append(str(temp).rjust(len(str(n)), '0'))

    return "".join(res)


def decrypt(self, cipher_text: str, d: int, n: int) -> str:
    max_length = len(str(n))
    num_alphabet = (len(str(n))-1)//3
    messages_int = utils.ciphertextToArrInt(cipher_text, max_length)

    res = []
    for message in messages_int:
        temp = pow(message, d, n)
        res.append(str(temp).rjust(num_alphabet*3, "0"))

    plaintext = utils.ArrStrToPlaintext(res, num_alphabet)

    return plaintext
```

2. Pailler.py

```
import utils
import sympy
import random
import typing


class Paillier_Crypt():
    def __init__(self):
        self.math = utils.Math()

    def set_public_key(self, g: int, n: int):
        self.g = g
        self.n = n

    def set_private_key(self, lmd: int, miu: int):
        self.lmd = lmd
        self.miu = miu

    def generate_paillier_key(self, key_size) -> typing.Tuple[int, int, int, int]:
        # Create two prime number, p and q such as pq coprime with totient(n).
        finish = False
        while(not(finish)):
            p = sympy.randprime(pow(2, key_size - 1) + 1, pow(2, key_size) - 1)
            q = sympy.randprime(pow(2, key_size - 1) + 1, pow(2, key_size) - 1)
            if(self.math.isCoprime(p * q, (p - 1) * (q - 1))):
                finish = True

        # Calculate n and lambda.
        n = p * q
        n_square = pow(n, 2)
        lmd = self.math.lcm(p - 1, q - 1)

        # Generate random integer.
        g = random.randint(1, pow(n, 2) - 1)

        # Calculate miu.
        L = lambda x : (x - 1) // n
        miu = self.math.modinv(L(pow(g, lmd, n_square)), n)

        # Return g, n, lambda, miu.
        return g, n, lmd, miu
```

```python
def generate_paillier_key_manual(self, p, q, g) -> typing.Tuple[int, int, int, int]:
    if(self.math.isPrime(p)):
        raise Exception('p is not prime')
    if(self.math.isPrime(q)):
        raise Exception('p is not prime')
    if(not(self.math.isCoprime(p * q, (p - 1)*(q - 1)))):
        raise Exception('p * q not coprime with totient(n)')

    n = p * q
    n_square = pow(n, 2)
    lmd = self.math.lcm(p - 1, q - 1)

    if(g <= 0 or g >= n_square):
        raise Exception('g must be positive and less than n^2')

    # Calculate miu.
    L = lambda x : (x - 1) // n
    miu = self.math.modinv(L(pow(g, lmd, n_square)), n)

    # Return g, n, lambda, miu.
    return g, n, lmd, miu

def encrypt(self, plain_text: str, g: int, n: int) -> str:
    max_length = (len(str(plain_text)) - 1) // 3
    messages_int =  utils.plaintextToArrInt(plain_text, max_length)

    res = []
    for message in messages_int:
        # Find r
        while(True):
            r = random.randint(0, n - 1)
            if(self.math.isCoprime(r, n)):
                break

        n_square = pow(n, 2)
        temp = (pow(g, message, n_square) * pow(r, n, n_square)) % n_square
        res.append(str(temp).rjust(len(str(n_square)), '0'))

    return "".join(res)
```

```python
def decrypt(self, cipher_text: str, lmd: int, miu: int, n: int) -> str:
    max_length = len(str(pow(n, 2)))
    num_alphabet = (len(str(n))-1)//3
    messages_int = utils.ciphertextToArrInt(cipher_text, max_length)

    L = lambda x : (x - 1) // n

    res = []
    for c in messages_int:
        temp = (L(pow(c, lmd, pow(n, 2))) * miu) % n
        res.append(str(temp).rjust(num_alphabet*3, "0"))

    plaintext = utils.ArrStrToPlaintext(res, num_alphabet)

    return plaintext
```

3. ElGamal.py

```python
import os
import random
import utils
import sympy

from datetime import datetime
from typing import Tuple

from utils.blockText import ArrStrToPlaintext, ciphertextToArrInt, plaintextToArrInt


class ElGamalKeygen:
    """
    A class used for generating public and private key from ElGamal algorithm.
    """

    def __init__(self, public_key:Tuple[int, int, int]=(0,0,0), private_key:Tuple[int,int]=(0,0)) -> None:
        """
        Constructor for ElGamalKeygen class.
        """
        self.public_key = public_key
        self.private_key = private_key
        self.math = utils.Math()
```

```python
def generateKey(self, is_random:bool=True, key_size:int=28, p:int=0) -> None:
    """
    Generate public and private key from parameter given.
    """
    # If generate random key.
    if (is_random):
        # Get the requirement element.
        p = sympy.randprime(pow(2, key_size - 1) + 1, pow(2, key_size) - 1)

    # Validation.
    if(not is_random):
        if (not self.math.isPrime(p) or len(str(p)) <= 3):
            raise Exception("P must be a prime number and greater than 1000")

    # Get another element.
    g = random.randrange(2, p-1)
    x = random.randrange(2, p-2)
    y = pow(g, x, p)

    self.public_key = (y, g, p)
    self.private_key = (x, p)
```

```python
def readKey(self, key:str, type:str) -> None:
    """
    Read key from text input.
    """
    try:
        # Modify the key based on file extension.
        if (type == "pub"):
            self.public_key = (int(key.split(" ")[0]), int(key.split(" ")[1]), int(key.split(" ")[2]))
        else:
            self.private_key = (int(key.split(" ")[0]), int(key.split(" ")[1]))
    except :
        raise Exception("INvalid key format")
```

```python
def loadKey(self, filename:str) -> None:
    """
    Load key from .pri and .pub file.
    """
    try:
        # Read the file.
        f = open(filename, "r")
        res = f.read()

        # Modify the key based on file extension.
        if (os.path.splitext(filename)[1].lower() == ".pub"):
            self.public_key = (int(res.split(" ")[0]), int(res.split(" ")[1]), int(res.split(" ")[2]))
        elif (os.path.splitext(filename)[1].lower() == ".pri"):
            self.private_key = (int(res.split(" ")[0]), int(res.split(" ")[1]))
        else :
            raise Exception("Invalid filename extension")

    except :
        raise Exception("Failed when opening file")
```

```python
    def saveKey(self, is_public: bool = True, filename:str= str(datetime.now())) -> None:
        """
        Write key to file .pri and .pub .
        """
        # Filling filename and content.
        # Default case.
        ext = ".pub"
        content = str(self.public_key[0]) + " " + str(self.public_key[1]) + " " + str(self.public_key[2])

        # Private case.
        if (not is_public):
            ext = ".pri"
            content = str(self.private_key[0]) + " " + str(self.private_key[1])

        # Writing file to directory.
        filename += ext

        try:
            f = open(filename, "w")
            f.write(content)
        except :
            raise Exception("Failed when writing file")
```

```python
class ElGamal_Crypt():
    def encrypt(self, plain_text: str, public_key:Tuple[int, int, int]) -> Tuple[str,str]:
        """
        Encrypt the plaintext using elgamal algorithm. Public key can be generated using ElGamalKeygen
        class. Return tuple containing two ciphertext.
        """
        # Get the public key.
        y, g, p = public_key

        # Prepare for encrypting.
        max_length = (len(str(p))-1)//3
        messages_int = plaintextToArrInt(plain_text, max_length)
        print("The block is:")
        print(messages_int)

        # Encrypt using elgamal algorithm.
        complete_a =[]
        complete_b =[]
        for message in messages_int:
            k = random.randint(1, p - 2)
            a = pow(g, k, p)
            b = ((pow(y, k, p) * message) % p)
            complete_a.append(str(a).rjust(len(str(p)), "0"))
            complete_b.append(str(b).rjust(len(str(p)), "0"))

        # Combine a to one string and b to one string.
        complete_a = "".join(complete_a)
        complete_b = "".join(complete_b)
        return (complete_a, complete_b)
```

```python
    def decrypt(self, cipher_text: Tuple[str, str], private_key:Tuple[int, int]) -> str:
        """
        Decrypt the ciphertext using elgamal algorithm. Private key can be generated using ElGamalKeygen
        class. Ciphertext must be a tuple of two string (a and b). Return plaintext.
        """
        # Get the private key.
        x, p  = private_key

        # Prepare for decrypting.
        max_length = len(str(p))
        num_alphabet = (len(str(p))-1)//3
        list_a = cipher_text[0]
        list_int_a = ciphertextToArrInt(list_a, max_length)
        list_b = cipher_text[1]
        list_int_b = ciphertextToArrInt(list_b, max_length)

        # Decrypting using elgamal algorithm.
        list_ascii_plaintext = []
        for a, b in zip(list_int_a, list_int_b):
            first_equation = pow(a, p - 1 - x, p)
            plaintext_int = (b * first_equation) % p
            list_ascii_plaintext.append(str(plaintext_int).rjust(num_alphabet*3, "0"))

        # Parse list ascii plaintext to real plaintext.
        plaintext = ArrStrToPlaintext(list_ascii_plaintext, num_alphabet)
        return plaintext
```

4. ECEG.py

```python
import os
import random
import utils
import math
import sympy

from datetime import datetime
from typing import Tuple, List
from utils.blockText import ciphertextToArrTupleInt


class ECC:
    """
    A class used for generating elliptic curve cryptography parameter/key.
    Also used for basic method in ECC.

    Attributes
    ----------
    a,b,p: int
        parameter persamaan kurva eliptik, p haruslah bilangan prima.
    group: List[Tuple[int,int]]
        grup eliptik yang dihitung dari persamaan kurva eliptik
    B: Tuple[int,int]
        Titik basis (base point) (xb,yb), dipilih dari grup eliptik untuk operasi kriptografi.
    N: int
        banyaknya anggota grup yang dimiliki.
    x: int
        kunci privat, dipilih dari selang [1, p-1]
    Q: Tuple[int,int]
        kunci publik, adalah hasil kali antara x dan titik basis B: Q = x.B
    """

    def __init__(self, a:int=0, b:int=0, p:int=0, N:int=0, B:Tuple[int, int]=(0,0),
        group:List[Tuple[int, int]]=None, x:int=0, Q:Tuple[int,int]=(0,0)) -> None:

        """
        Constructor for ECEGKeygen class.
        """

        self.a = a
        self.b = b
        self.p = p
        self.group = group
        self.N = N
        self.B = B
        self.x = x
        self.Q = Q
        self.math = utils.Math()

    def fullyRandomizeAttribute(self, p_size:int = 14):
        """
        Fully randomized attribute.
        """

        self.generateEllipticCurve(is_random=True, p_size=p_size)
        self.generateGroup(is_random=True)
        self.generateBasis(is_random=True)
        self.generateKey(is_random=True)
```

```python
def generateEllipticCurve(self, a:int=0, b:int=0, p:int=0, is_random:bool=True,
    p_size:int=12)->None:
    """
    Generate Elliptic Curve basic parameter
    """
    # Randomized case.
    if (is_random):
        p = sympy.randprime(pow(2, p_size - 1) + 1, pow(2, p_size) - 1)
        a = random.randrange(2, p-1)
        b = random.randrange(1, p-1)
        while (4*(a**3) + 27*(b**2) == 0):
            a = random.randrange(2, p-1)
            b = random.randrange(1, p-1)

    # Validation.
    if (not is_random):
        if (not self.math.isPrime(p)):
            raise Exception("P must be a prime number")
        if(4*(a**3) + 27*(b**2) == 0):
            raise Exception("Invalid a and b value (4a^3 + 27b^2 != 0)")

    # Fill the value.
    self.a = a
    self.b = b
    self.p = p
```

```python
def generateGroup(self, is_random:bool=True, group:List[Tuple[int, int]]=None,  N:int=0) -> None:
    """
    Generate elliptic curve group.
    """
    # Randomized case.
    if (is_random):
        group = []
        lengroup= 0
        # Enumerate all possible x and y values.
        for x in range(self.p):
            if (lengroup >= 257):
                break
            right_side = (pow(x,3, self.p) + self.a*x + self.b) % self.p
            for y in range(self.p):
                if (lengroup >= 257):
                    break
                left_side = pow(y,2,self.p)
                # If matching then (x,y) is part of the group.
                if (right_side == left_side):
                    group.append((x,y))
                    lengroup += 1

        # Get the length.
        N = lengroup
    # Validation.
    if(N < 256):
        raise Exception("Group size must be atleast 256 point")
    # Fill the value.
    self.group = group
    self.N = N
```

```python
def generateBasis(self, is_random:bool=True, B:Tuple[int,int]=(0,0))-> None:
    """
    Generate basis for elliptic curve.
    """
    # Randomized case.
    if (is_random):
        B = self.group[random.randrange(0, len(self.group))]

    # Fill the value.
    self.B = B
```

```python
def generateKey(self, is_random:bool=True, d:int=2, Q:Tuple[int,int]=(0,0)):
    """
    Generate public key and private key.
    """
    if (is_random):
        # Generate private key
        d = random.randrange(2, self.p-1)

        # Generate public key.
        Q = self.perkalianTitik(d, self.B)
        while ((Q[0]==math.inf and Q[1]==math.inf)):
            # Generate private key
            d = random.randrange(2, self.p-1)

            # Generate public key.
            Q = self.perkalianTitik(d, self.B)

    # Validation.
    if (d <= 1 or d >= self.p):
        raise Exception("d must be between 1 and p")
    if(Q[0]==math.inf and Q[1]==math.inf):
        raise Exception("Q cant be an identity point")

    # Fill the value.
    self.d = d
    self.Q = Q
```

```python
def penjumlahanTitik(self, titik1:Tuple[int,int], titik2:Tuple[int,int]) ->Tuple[int,int]:
    """
    Melakukan penjumlahan dua titik pada elliptic curve.
    """

    # Variable naming.
    xp, yp = titik1
    xq, yq = titik2

    # Sifat elemen netral pada abelian.
    if (xp == math.inf and yp == math.inf):
        return (xq, yq)
    if (xq == math.inf and yq == math.inf):
        return (xp, yp)

    # If ditambah dengan inverse
    if (xp == xq and yp == (-1*yq) % self.p):
        return (math.inf, math.inf)

    # If equal checking (penggandaan).
    if (xp == xq and yp == yq):
        # Jika ordinat 0 maka tangen berpotongan di infinity.
        if (yp == 0):
            return (math.inf, math.inf)
        else:
            # Rumus gradien pada penggandaan.
            m = ((3*(xp**2) + self.a) * self.math.modinv3(2*yp, self.p)) % self.p

            # Rumus xr dan yr pada penggandaan.
            xr = (m**2 - 2*xp) % self.p
            yr = (m*(xp-xr)-yp) % self.p
            return (xr, yr)
    else:
        # Gradien infinity.
        if (xp - yp) == 0:
            return  (math.inf, math.inf)
        else:
            # Rumus gradien biasa.

            m = ((yp -yq) * self.math.modinv3(xp - xq, self.p)) % self.p
            # Rumus xr dan yr pada penggandaan.
            xr = (m**2 - xp - xq) % self.p
            yr = (m*(xp-xr)-yp) % self.p
            return (xr, yr)
```

```python
def perkalianTitik(self, k:int, titik:Tuple[int,int]):
    """
    Melakukan perkalian titik secara rekursif.
    """
    res = (math.inf, math.inf)
    for i in range(k):
        res = self.penjumlahanTitik(res, titik)

    return res

def negative(self, titik:Tuple[int,int]):
    """
    Melakukan negasi titik.
    """
    return (titik[0], -1* titik[1] % self.p)
```

```python
def readKey(self, key:str, type:str):
    """
    Read key from string in file.
    """

    # Read the string.
    res = key

    # Modify the key based on file extension.
    if (type =="pub"):
        # .pub
        # a b p Bx By Qx Qy
        a = int(res.split(" ")[0])
        b = int(res.split(" ")[1])
        p = int(res.split(" ")[2])
        Bx = int(res.split(" ")[3])
        By = int(res.split(" ")[4])
        B = (Bx, By)
        Qx = int(res.split(" ")[5])
        Qy = int(res.split(" ")[6])
        Q = (Qx, Qy)

        self.generateEllipticCurve(a=a, b=b, p=p, is_random=False)
        self.generateGroup(is_random=True)
        self.generateBasis(is_random=False, B = B)
        self.generateKey(is_random=False, Q = Q)

    else:
        # .pri
        # a b p Bx By d
        a = int(res.split(" ")[0])
        b = int(res.split(" ")[1])
        p = int(res.split(" ")[2])
        Bx =int(res.split(" ")[3])
        By = int(res.split(" ")[4])
        B = (Bx, By)
        d = int(res.split(" ")[5])

        self.generateEllipticCurve(a=a, b=b, p=p, is_random=False)
        self.generateGroup(is_random=True)
        self.generateBasis(is_random=False, B = B)
        self.generateKey(is_random=False, d = d)
```

```python
def loadKey(self, filename:str) -> None:
    """
    Load key from .pri and .pub file.
    """
    try:
        # Read the file.
        f = open(filename, "r")
        res = f.read()

        # Modify the key based on file extension.
        if (os.path.splitext(filename)[1].lower() == ".pub"):
            # .pub
            # a b p Bx By Qx Qy
            a = int(res.split(" ")[0])
            b = int(res.split(" ")[1])
            p = int(res.split(" ")[2])
            Bx = int(res.split(" ")[3])
            By = int(res.split(" ")[4])
            B = (Bx, By)
            Qx = int(res.split(" ")[5])
            Qy = int(res.split(" ")[6])
            Q = (Qx, Qy)

            self.generateEllipticCurve(a=a, b=b, p=p, is_random=False)
            self.generateGroup(is_random=True)
            self.generateBasis(is_random=False, B = B)
            self.generateKey(is_random=False, Q = Q)

        elif (os.path.splitext(filename)[1].lower() == ".pri"):
            # .pri
            # a b p Bx By d
            a = int(res.split(" ")[0])
            b = int(res.split(" ")[1])
            p = int(res.split(" ")[2])
            Bx = int(res.split(" ")[3])
            By = int(res.split(" ")[4])
            B = (Bx, By)
            d = int(res.split(" ")[5])

            self.generateEllipticCurve(a=a, b=b, p=p, is_random=False)
            self.generateGroup(is_random=True)
            self.generateBasis(is_random=False, B = B)
            self.generateKey(is_random=False, d = d)

        else :
            raise Exception("Invalid filename extension")

    except :
        raise Exception("Failed when opening file")
```

```python
def saveKey(self, is_public: bool = True, filename:str= str(datetime.now())) -> None:
    """
    Write key to file .pri and .pub .
    """
    # .pri
    # a b p Bx By d

    # .pub
    # a b p Bx By Qx Qy
    # Filling filename and content.

    # Default case.
    if (is_public):
        ext = ".pub"
        content = str(self.a) + " " + str(self.b) + " " + str(self.p) + " "
        content = content + str(self.B[0]) + " " + str(self.B[1]) + " "
        content = content + str(self.Q[0]) + " " + str(self.Q[1])

    # Private case.
    if (not is_public):
        ext = ".pri"
        content = str(self.a) + " " + str(self.b) + " " + str(self.p) + " "
        content = content + str(self.B[0]) + " " + str(self.B[1]) + " "
        content = content + str(self.d)

    # Writing file to directory.
    filename += ext

    try:
        f = open(filename, "w")
        f.write(content)
    except :
        raise Exception("Failed when writing file")
```

```python
    def getKeyFormatted(self, type:str):
        """
        Formatted key for file.
        """
        content = str(self.a) + " " + str(self.b) + " " + str(self.p) + " "
        content = content + str(self.B[0]) + " " + str(self.B[1]) + " "
        if (type == "pub"):
            content = content + str(self.Q[0]) + " " + str(self.Q[1])
        else:
            content = content + str(self.d)

        return content


    def printDetail(self):
        """
        Print the detail of the curve
        """
        print("Elliptic Curve parameter")
        print("a: ", self.a)
        print("b: ", self.b)
        print("p: ", self.p)
        print("N -> Group size: ", self.N)
        print("B -> Titik basis: ", end="")
        print(self.B)
        print("d -> private key: ", self.d)
        print("Q -> Public key: ", end="")
        print(self.Q)
```

```python
class ECEG:
    """
    A class used for representing ECC with elgamal algorithm.
    """
    def __init__(self, ecc:ECC=None) -> None:
        """
        Constructor for ECEG class.
        """
        # if (ecc is None):
        #   ecc = ECC()
        #   ecc.fullyRandomizeAttribute()
        self.ecc = ecc
```

```python
def encrypt(self, plain_text: str) -> Tuple[str,str]:
    """
    Encrypt the plaintext using ECEG algorithm. Return tuple containing two ciphertext.
    """

    # Encrypt using ECEG algorithm.
    complete_a =[]
    complete_b =[]

    for char in plain_text:
        # Encrypt.
        pm = self.ecc.group[ord(char)]
        k = random.randrange(2, self.ecc.p - 2)
        a = self.ecc.perkalianTitik(k, self.ecc.B)
        b = self.ecc.penjumlahanTitik(pm, self.ecc.perkalianTitik(k, self.ecc.Q))

        # Append to tuple result.
        a1 = str(a[0]).rjust(len(str(self.ecc.p)), "0")
        a2 = str(a[1]).rjust(len(str(self.ecc.p)), "0")
        complete_a.append(a1+a2)

        b1 = str(b[0]).rjust(len(str(self.ecc.p)), "0")
        b2 = str(b[1]).rjust(len(str(self.ecc.p)), "0")
        complete_b.append(b1+b2)
```

```python
def decrypt(self, cipher_text: Tuple[str, str]) -> str:
    """
    Decrypt the ciphertext using ECEG algorithm.
    Ciphertext must be a tuple of two string (a and b). Return plaintext.
    """

    # Prepare for decrypting.
    max_length = len(str(self.ecc.p))
    num_alphabet = 1
    list_a = cipher_text[0]
    list_int_a = ciphertextToArrTupleInt(list_a, max_length)
    list_b = cipher_text[1]
    list_int_b = ciphertextToArrTupleInt(list_b, max_length)

    print("a: ", end="")
    print(list_int_a)
    print("b: ", end="")
    print(list_int_b)


    # Decrypting using elgamal algorithm.
    plaintext = ""
    for a, b in zip(list_int_a, list_int_b):
        first_equation = self.ecc.perkalianTitik(self.ecc.d, a)
        pm = self.ecc.penjumlahanTitik(b, self.ecc.negative(first_equation))
        ascii = self.ecc.group.index(pm)
        plaintext += chr(ascii)

    return plaintext
```

5. BlockText.py

```python
from typing import Tuple, List


def plaintextToArrInt(plaintext:str, max_length:int) -> List[int]:
    """
    Convert plaintext to array of integer for encrypt.
    """
    equalSizedStr = [plaintext[i:i+max_length] for i in range(0, len(plaintext), max_length)]
    block = []
    for string in equalSizedStr:
        ascii_string =""
        for char in string:
            ascii_string += str(ord(char)).rjust(3, "0")
        block.append(int(ascii_string))
    return block

def ciphertextToArrInt(ciphertext:str, max_length:int) -> List[int]:
    """
    Convert ciphertext to array of integer for decrypt.
    """
    equalSizedStr = [ciphertext[i:i+max_length] for i in range(0, len(ciphertext), max_length)]
    block = [int(ciphertext) for ciphertext in equalSizedStr]
    return block
```

```python
def ArrStrToPlaintext(arr_str:List[str], num_alphabet:int)-> str:
    """
    Convert list of plaintext in ascii to string of normal plaintext.
    """
    plaintext = ""
    for combined_ascii in arr_str:
        temp = [combined_ascii[i:i+3] for i in range(0, len(combined_ascii), 3)]
        for ascii in temp:
            if (int(ascii)!= 0):
                plaintext += chr(int(ascii))
    return plaintext


def ciphertextToArrTupleInt(ciphertext:str, max_length:int) -> List[Tuple[int, int]]:
    """
    # Helper for ECEG algorithm.
    Convert ciphertext to array of tuple(int,int) for decrypt.
    """
    max_length_tuple = max_length*2
    equalSizedTuple = [ciphertext[i:i+max_length_tuple] for i in range(0, len(ciphertext), max_length_tuple)]
    equalSizedStr = [(tuple[0:max_length],tuple[max_length:max_length*2]) for tuple in equalSizedTuple]
    block = [(int(tuple[0]),int(tuple[1])) for tuple in equalSizedStr]

    return block
```

6. Math.py

```python
from typing import Tuple
import math

class Math:
    def gcd(self, x: int, y: int) -> int:
        while(y):
            x, y = y, x % y
        return x

    def lcm(self, x: int, y: int) -> int:
        return (x * y) // self.gcd(x, y)

    def egcd(self, x: int, y: int) -> Tuple[int, int, int]:
        if x == 0:
            return y, 0, 1

        gcd, x_hat, y_hat = self.egcd(y % x, x)

        x_res = y_hat - (y // x) * x_hat
        y_res = x_hat

        return gcd, x_res, y_res

    def modinv(self, x: int, y: int) -> int:
        # Work when x and y are coprime
        gcd, x, _ = self.egcd(x, y)
        if gcd != 1:
            raise Exception('Failed to compute modinv')
        return x % y
```

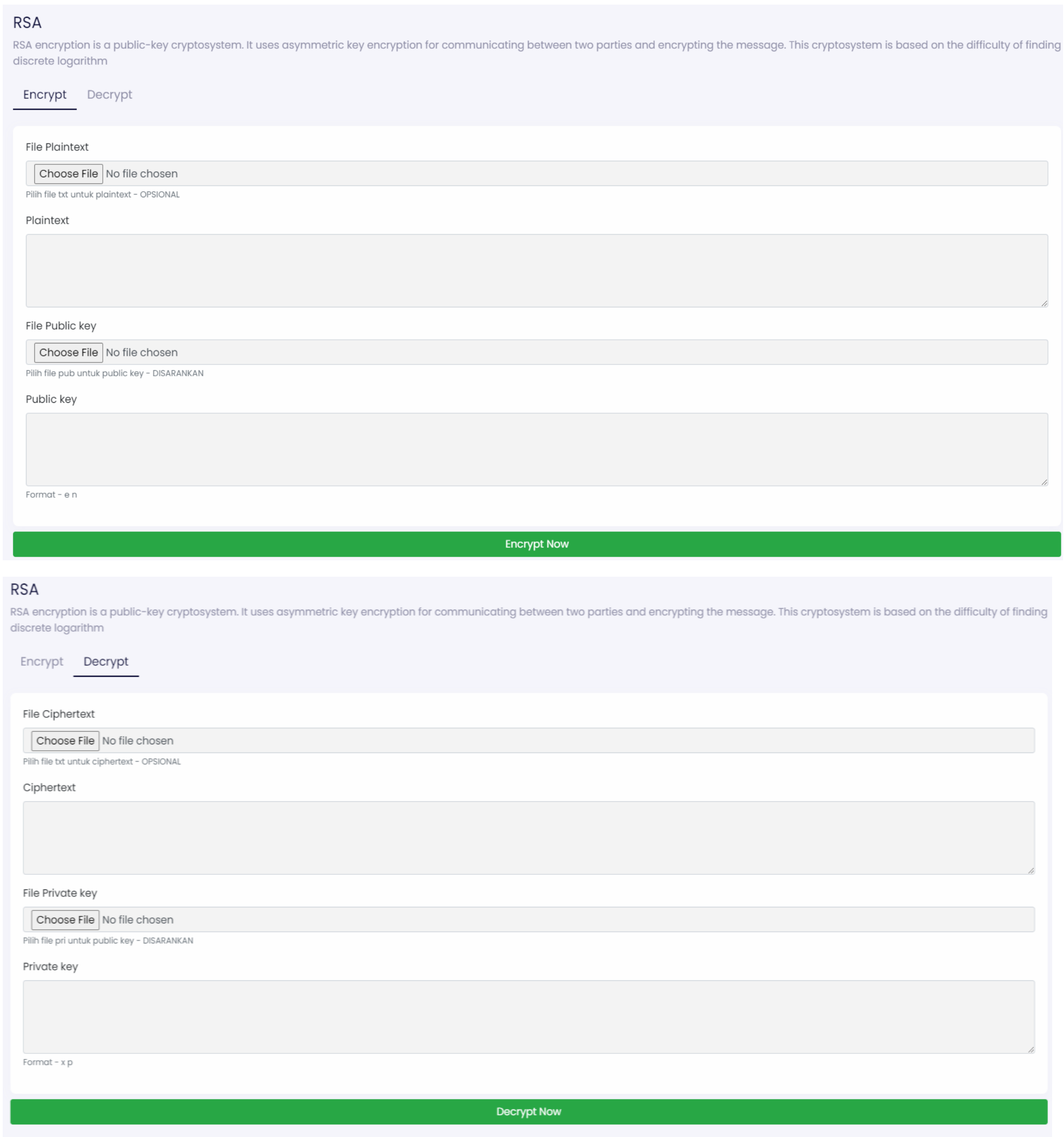
```python
    def modinv2(self, x: int, y: int) -> int:
        # Naive
        for i in range(y):
            if (x * i) % y == 1:
                return i
        raise Exception('Failed to compute modinv')

    def modinv3(self, x: int, y: int) -> int:
        # Work when x and y are coprime v2
        x = x % y
        gcd, x, _ = self.egcd(x, y)
        if gcd != 1:
            raise Exception('Failed to compute modinv')
        return x % y

    def isCoprime(self, x: int, y: int) -> bool:
        return self.gcd(x, y) == 1

    def totient(self, x: int, y: int) -> int:
        return (x - 1) * (y - 1)

    def isPrime(self, a:int) -> bool:
        for n in range(2,int(a**1/2)+1):
            if a % n == 0:
                return False
        return True
```

BAB II

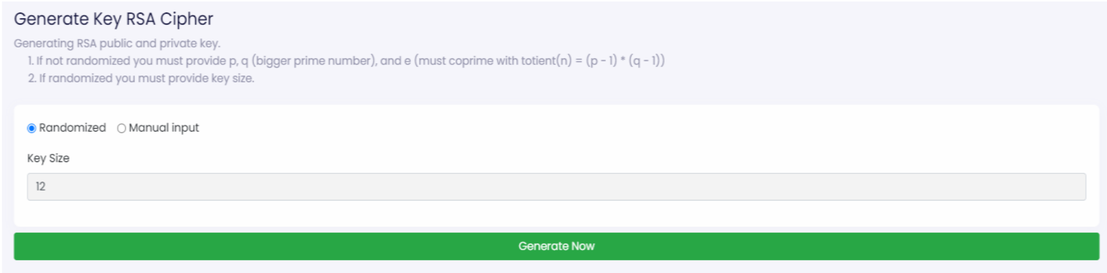
Tampilan Antarmuka Program

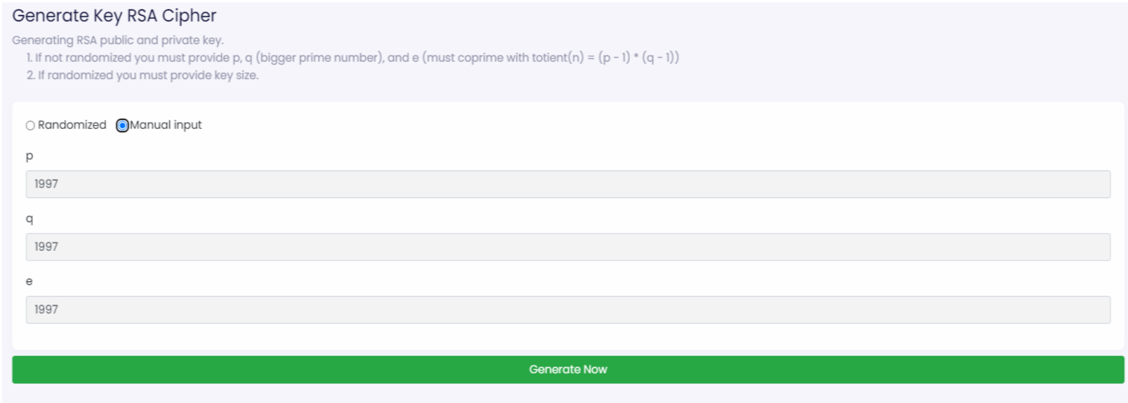
1. RSA

Encryption / Decryption



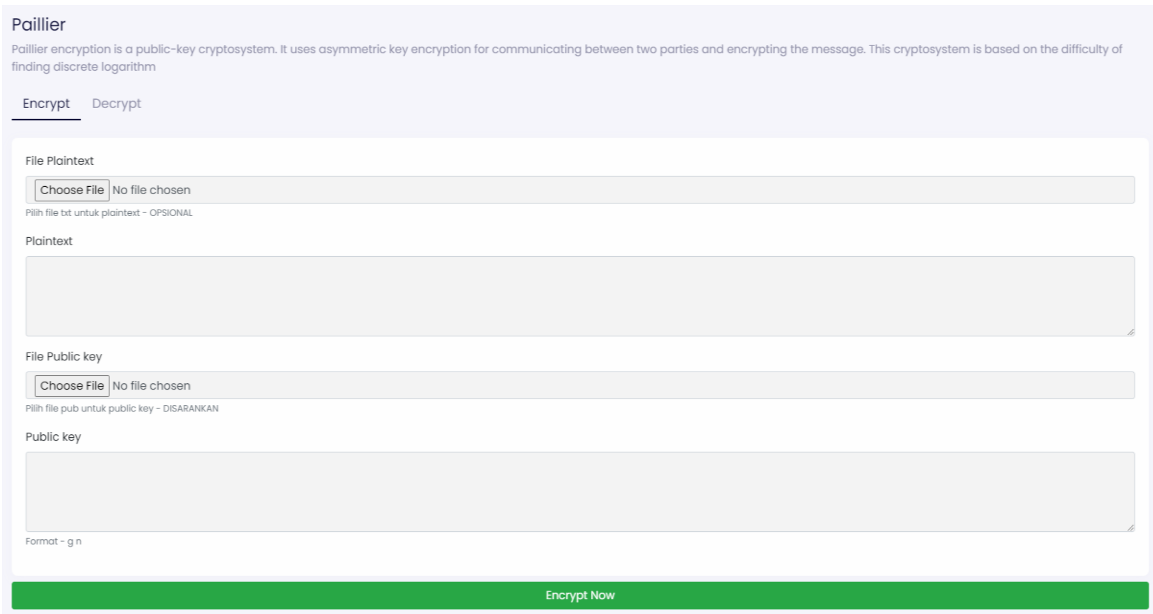
Generate Key

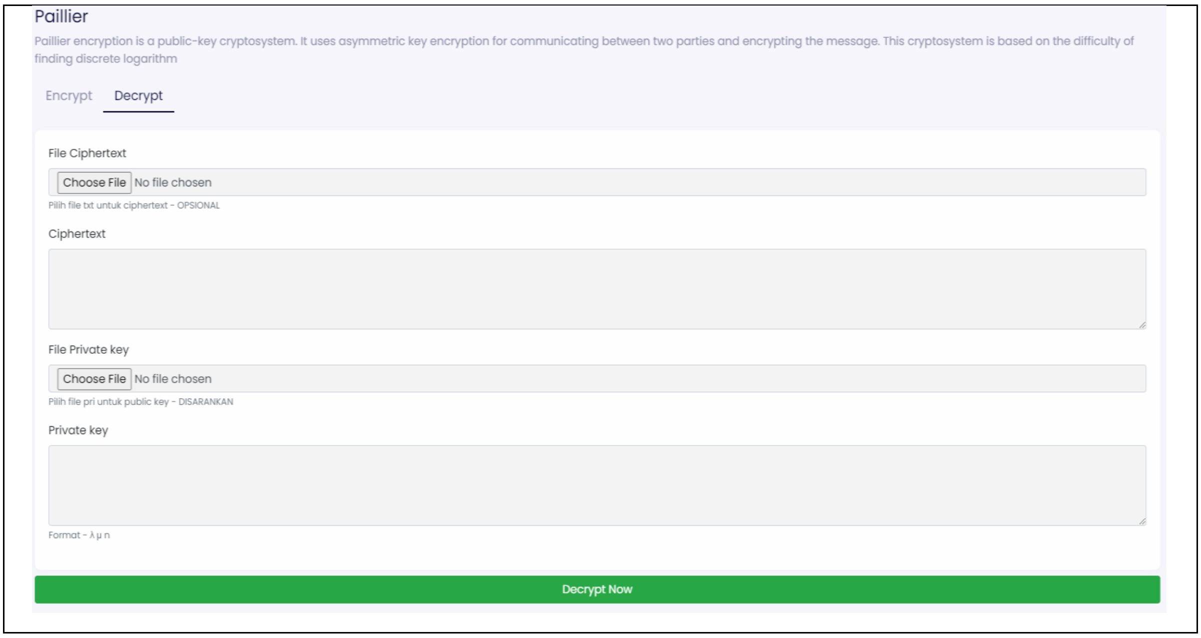




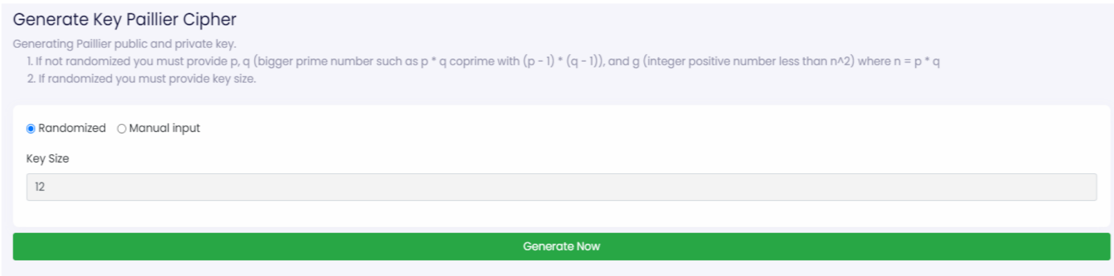
2. Paillier

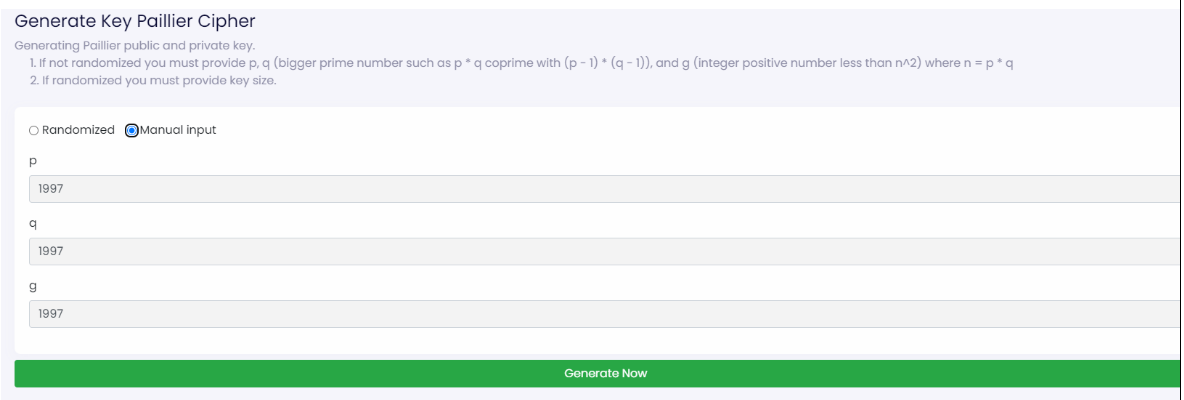
Encryption / Decryption



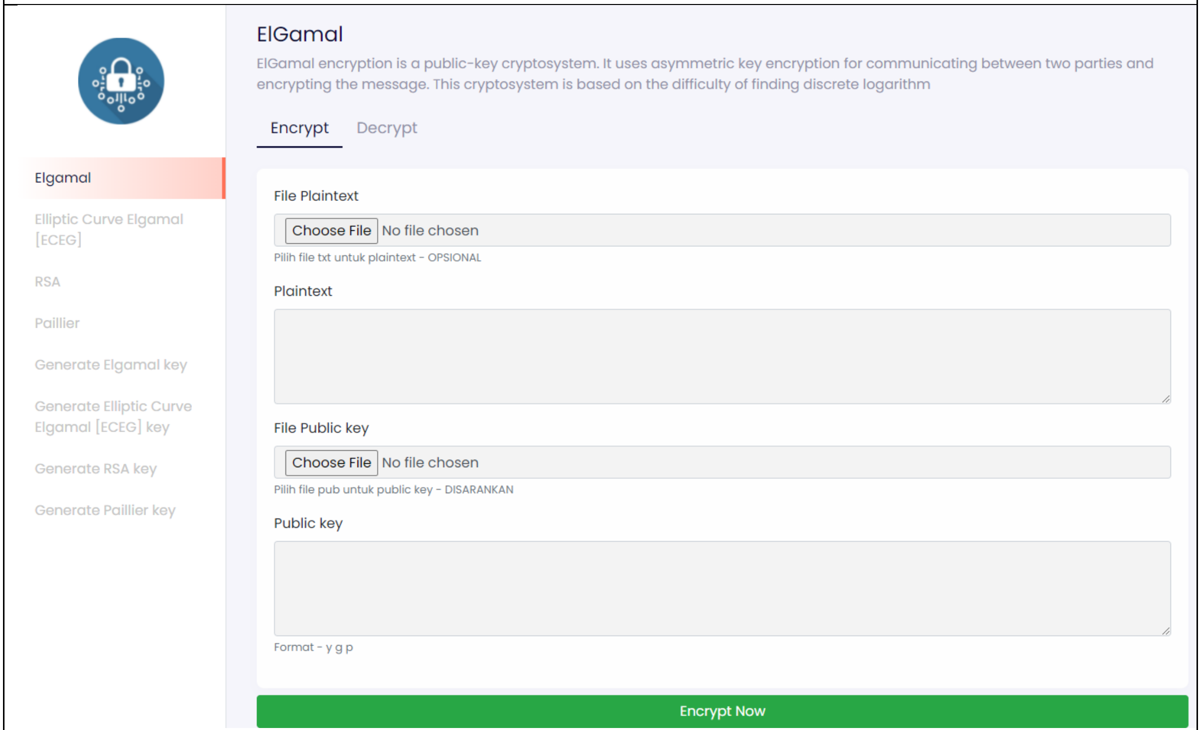
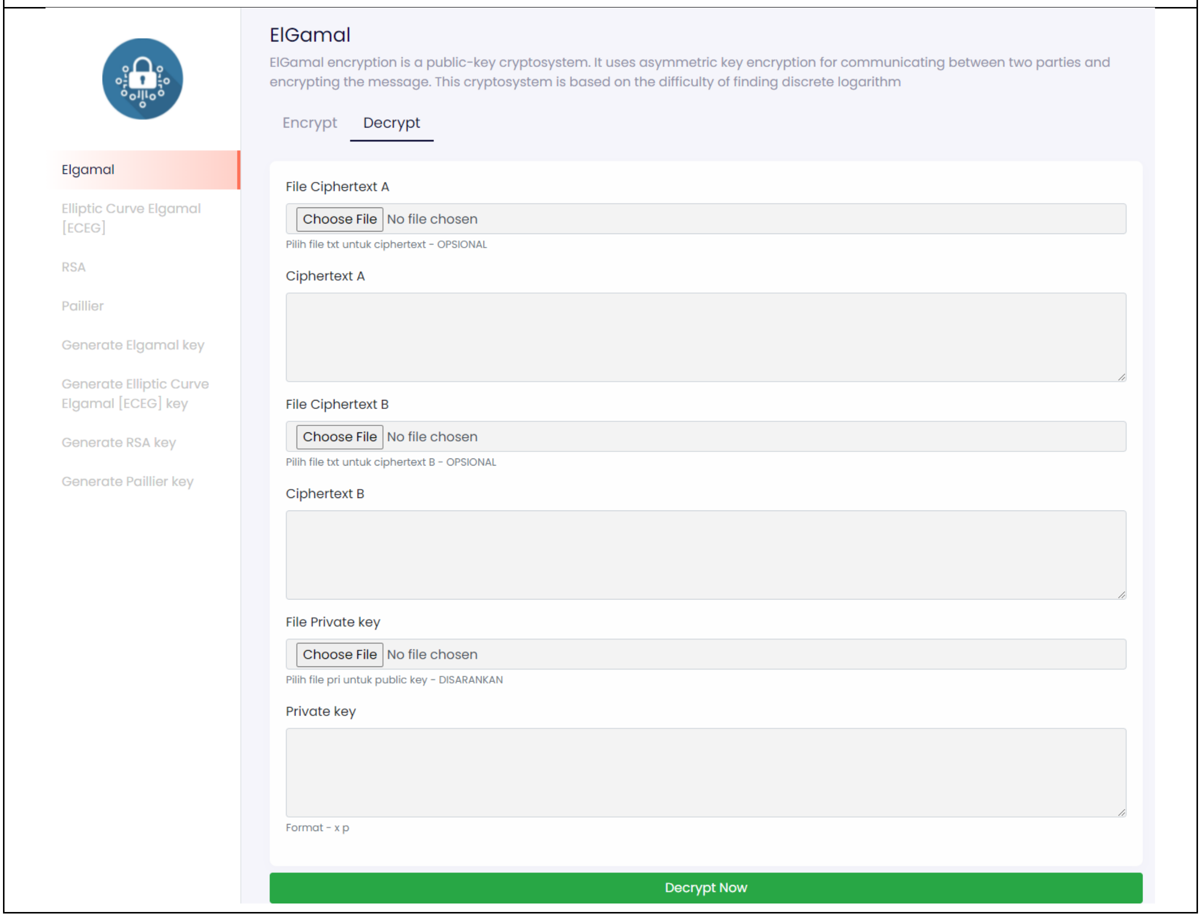


Generate Key

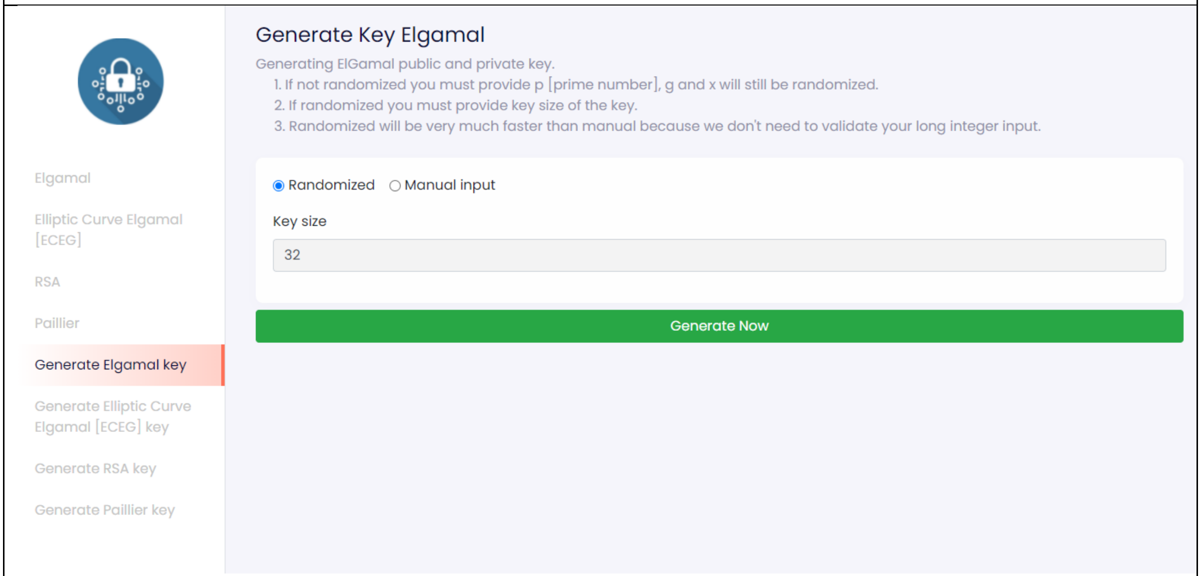


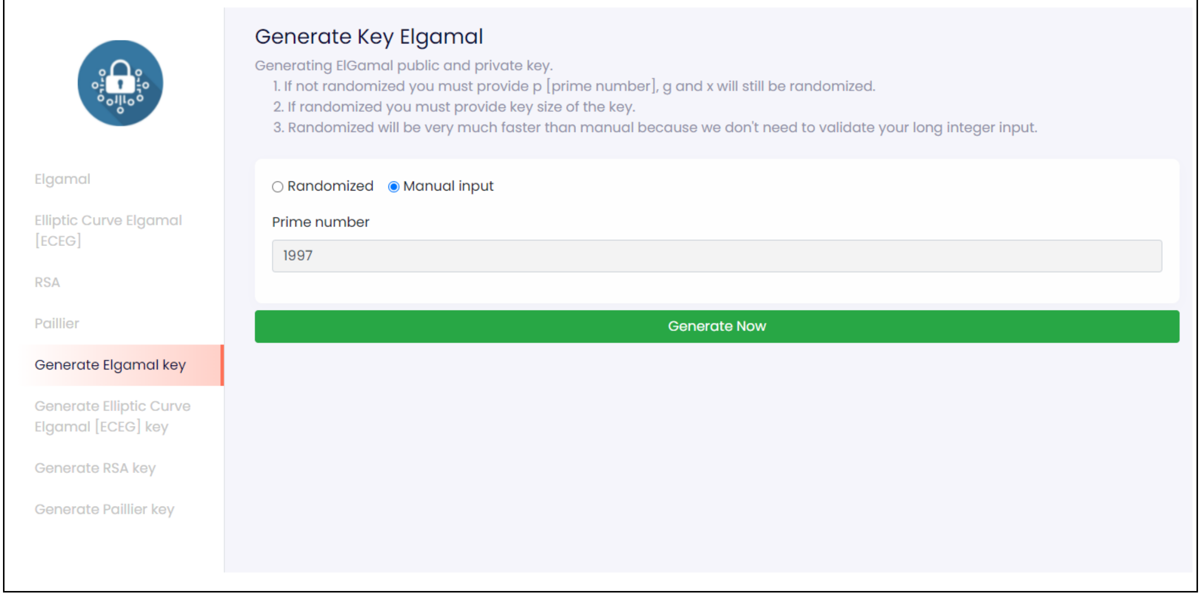


3. Elgamal

Encryption/Decryption



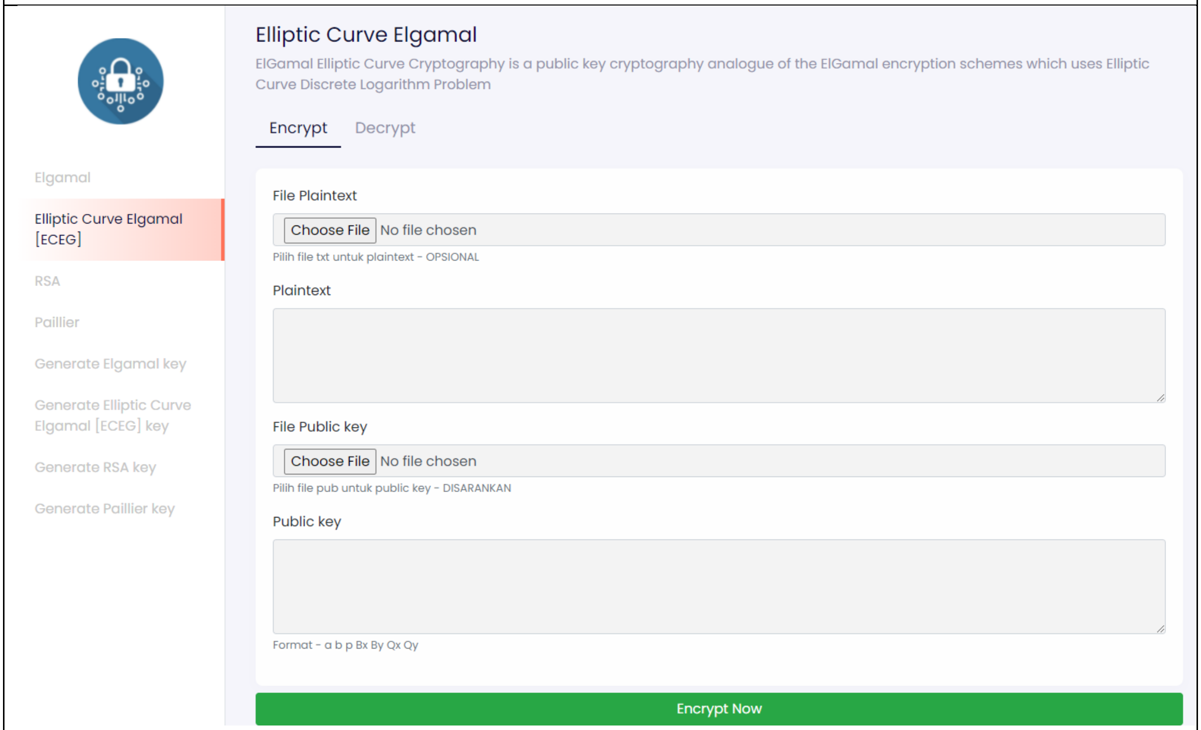
Generate Key

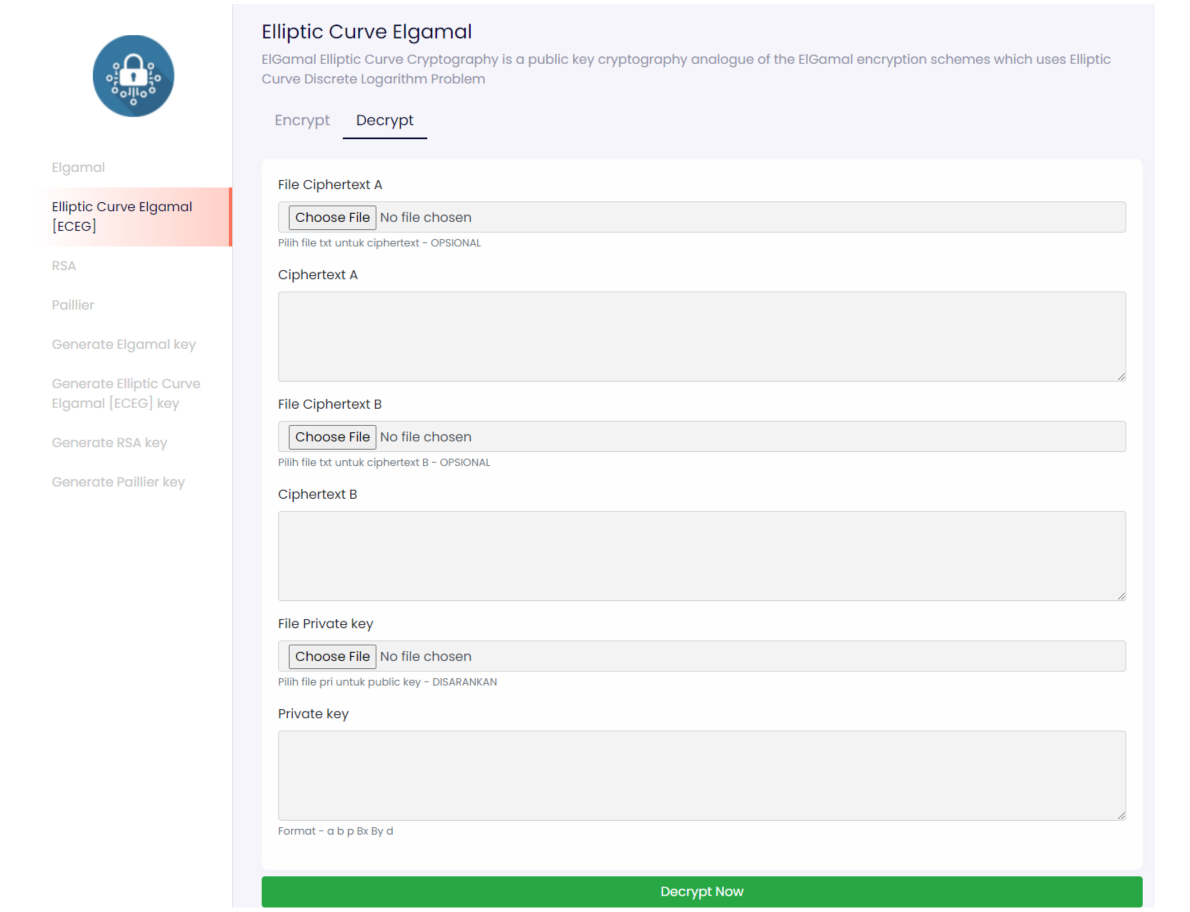




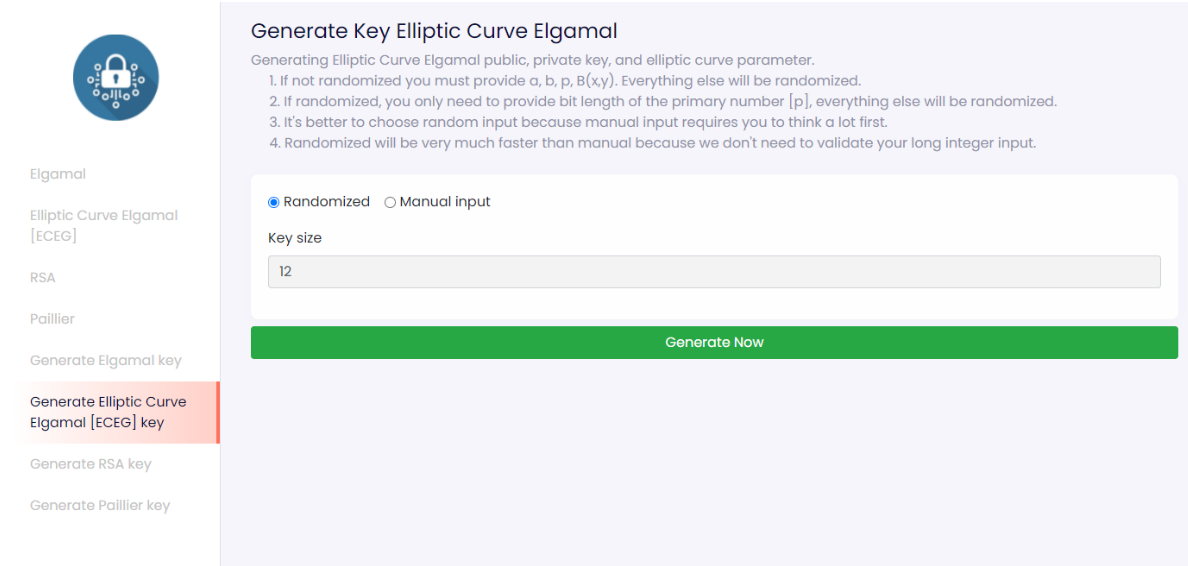
4. ECEG

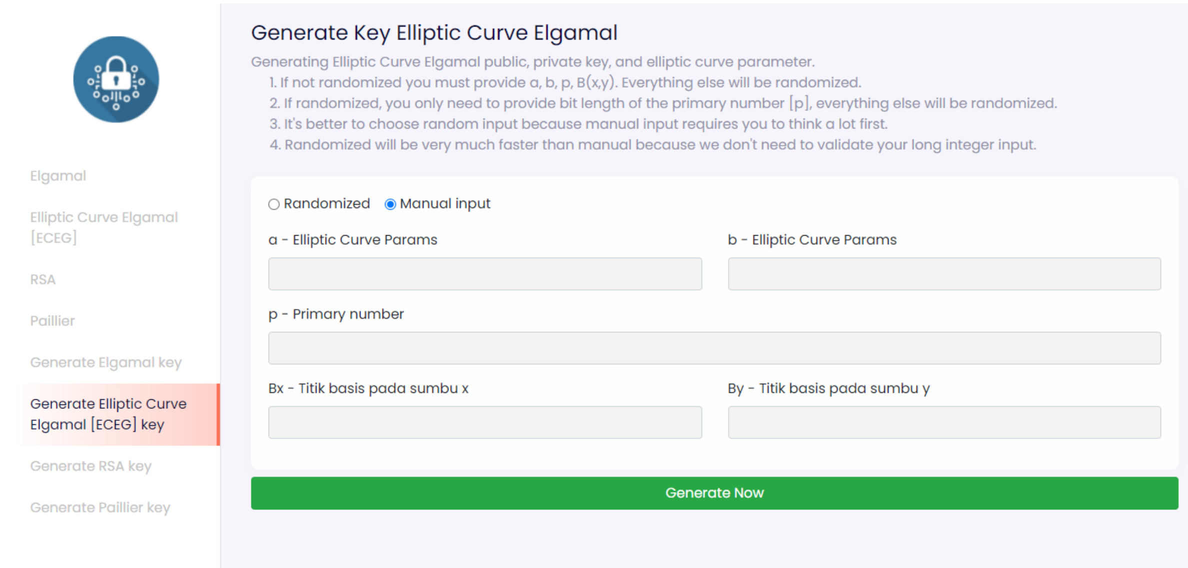
Encryption/Decryption





Generate Key





BAB III

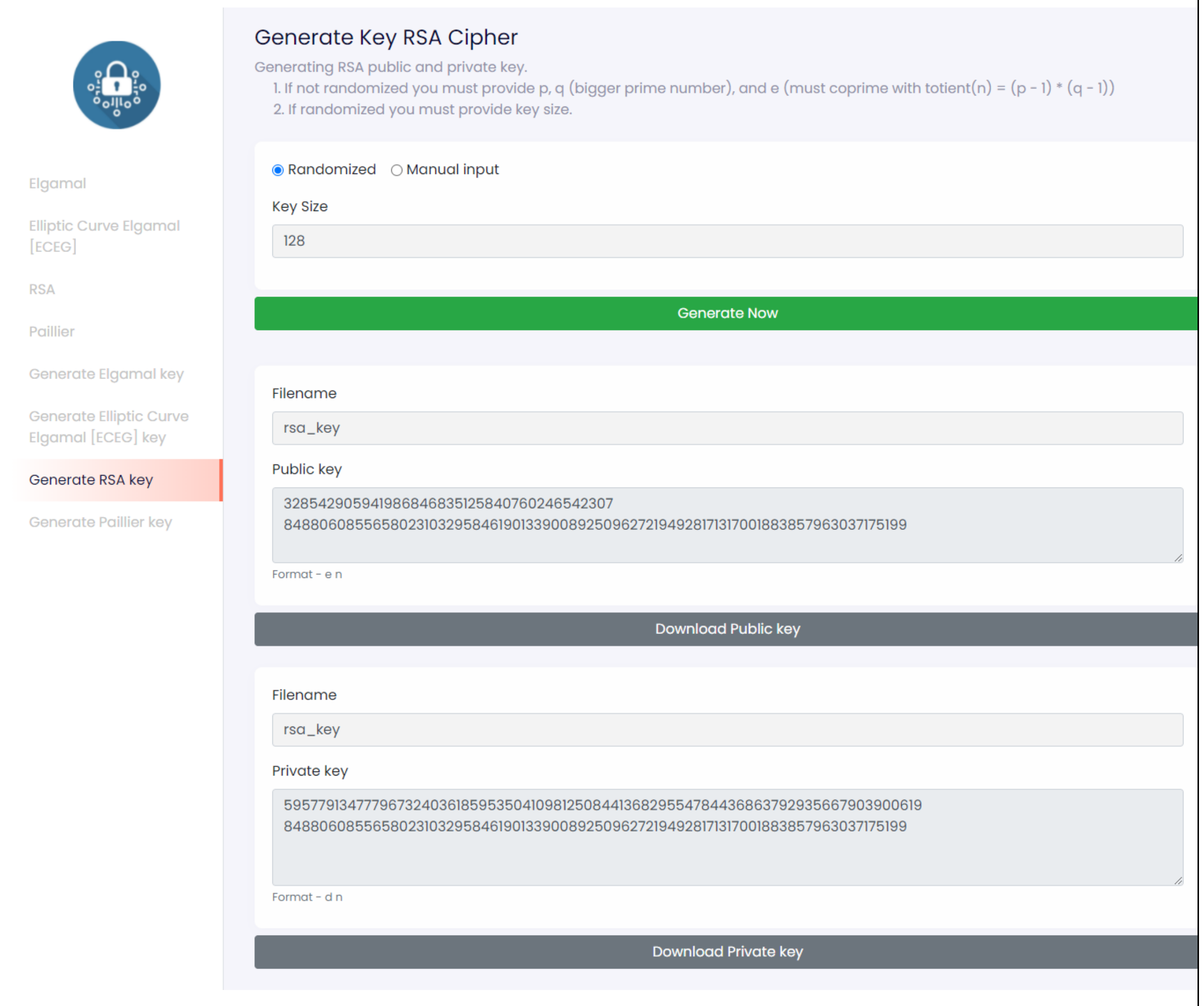
Hasil Percobaan

Semua file kunci/text yang disimpan bisa diakses di folder test

1. RSA

a. Generate Key

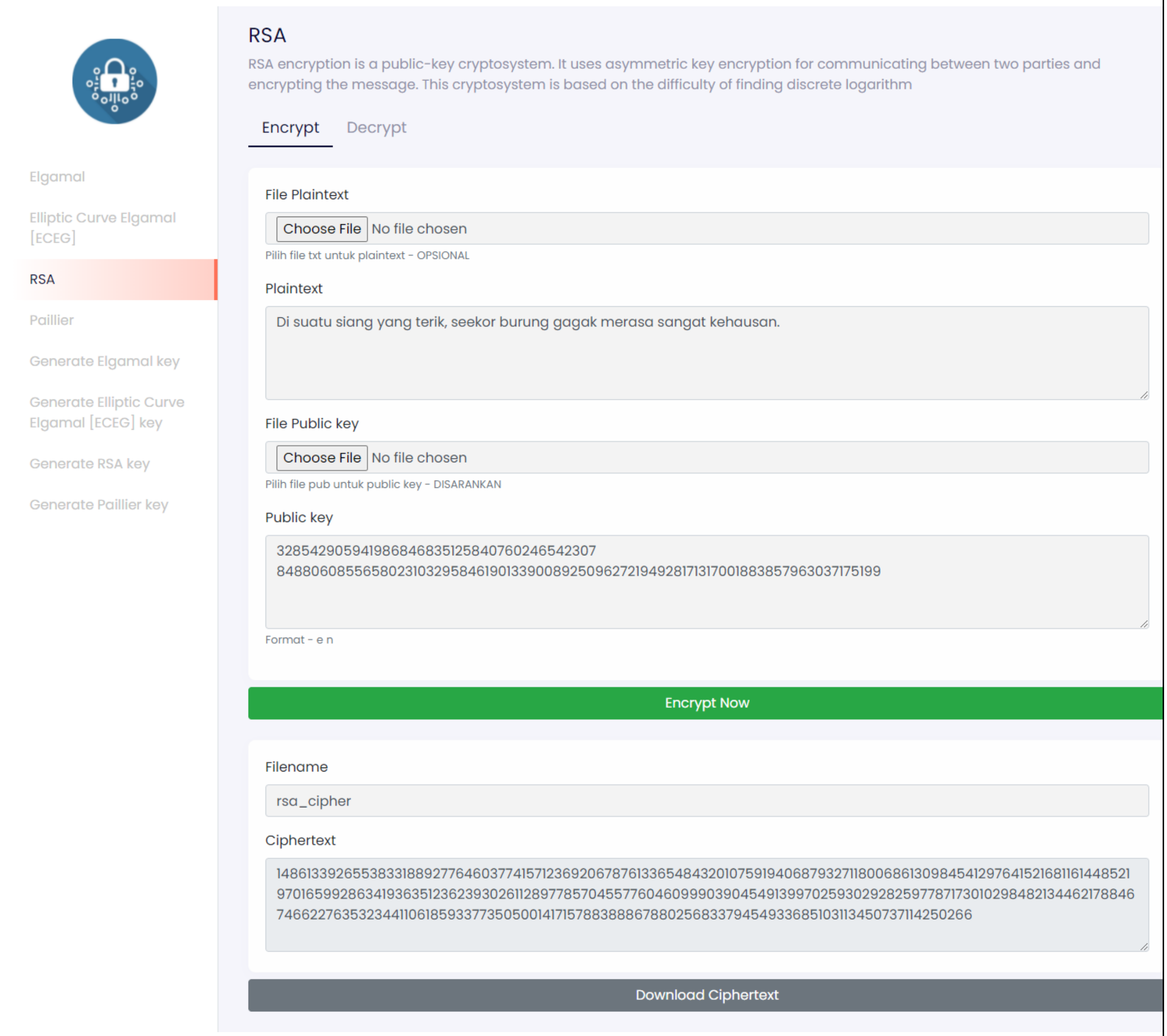
Ukuran Kunci	128
Kunci Publik	rsa_key.pub
Kunci Privat	rsa_key.pri
Hasil	



b. Enkripsi

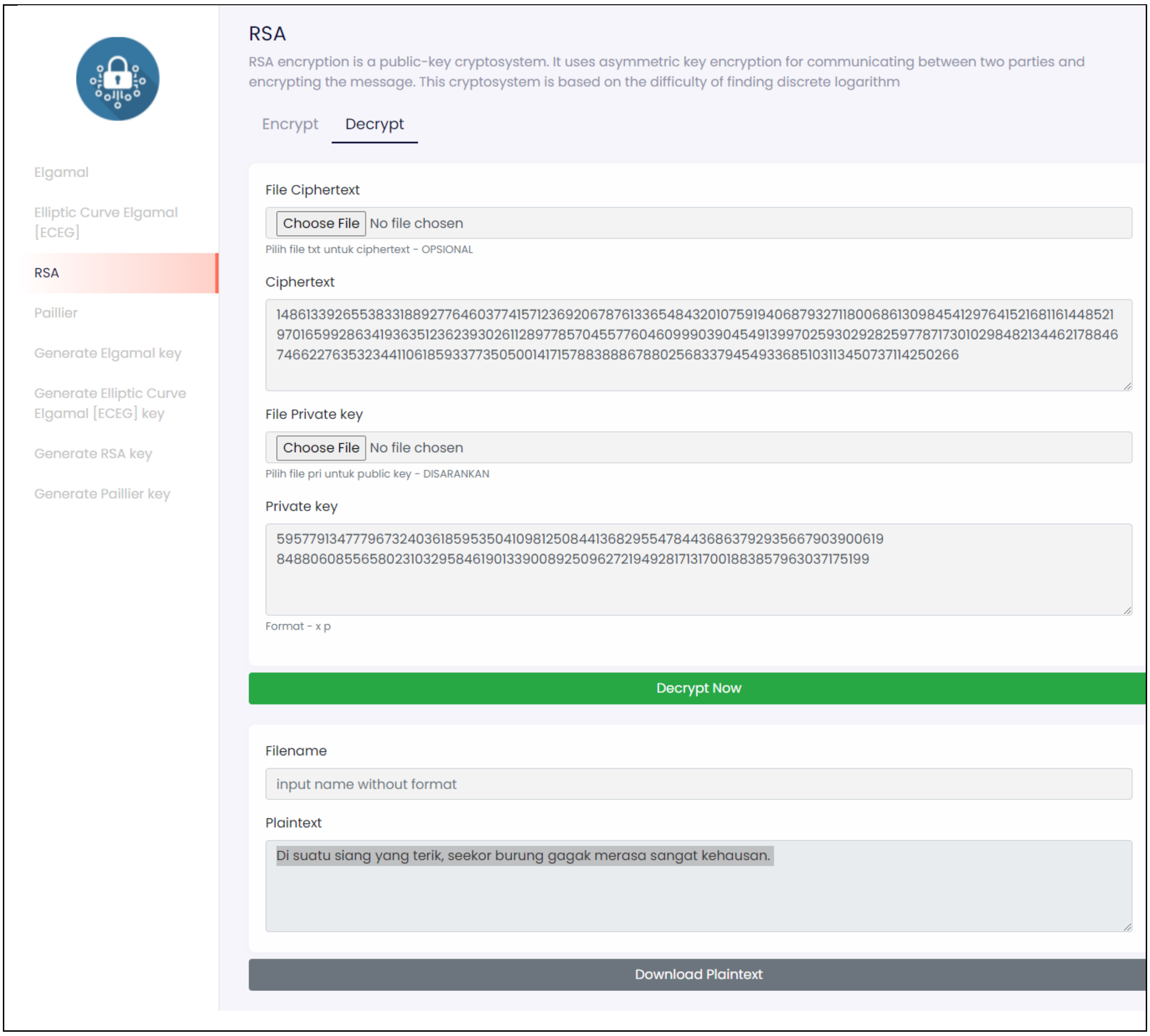
Kunci Publik	rsa_key.pub
Ciphertext Hasil	rsa_cipher.txt

Plaintext	Di suatu siang yang terik, seekor burung gagak merasa sangat kehausan.
Hasil	



c. Dekripsi

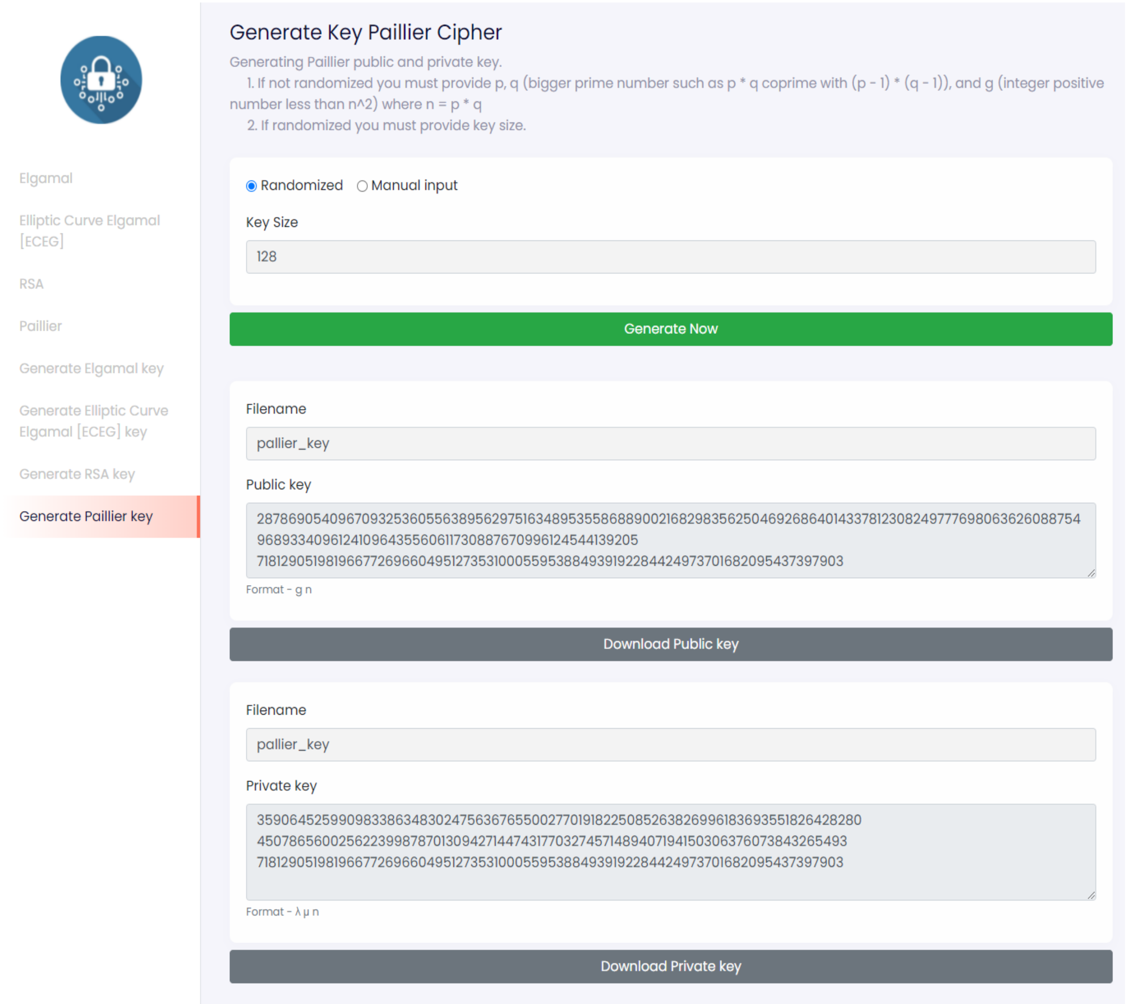
Kunci Privat	rsa_key.pri
Ciphertext	rsa_cipher.txt
Plaintext Hasil	Di suatu siang yang terik, seekor burung gagak merasa sangat kehausan
Hasil	


Elgamal
Elliptic Curve Elgamal [ECEG]
RSA
Paillier
Generate Elgamal key
Generate Elliptic Curve Elgamal [ECEG] key
Generate RSA key
Generate Paillier key
RSA
RSA encryption is a public-key cryptosystem. It uses asymmetric key encryption for communicating between two parties and encrypting the message. This cryptosystem is based on the difficulty of finding discrete logarithm
Encrypt
Decrypt
File Ciphertext
Choose File No file chosen
Pilih file txt untuk ciphertext - OPSIONAL
Ciphertext
14861339265538331889277646037741571236920678761336548432010759194068793271180068613098454129764152168116144852197016599286341936351236239302611289778570455776046099903904549139970259302928259778717301029848213446217884674662276353234411061859337735050014171578838886788025683379454933685103113450737114250266
File Private key
Choose File No file chosen
Pilih file pri untuk public key - DISARANKAN
Private key
59577913477796732403618595350410981250844136829554784436863792935667903900619 84880608556580231032958461901339008925096272194928171317001883857963037175199
Format - x p
Decrypt Now
Filename
input name without format
Plaintext
Di suatu siang yang terik, seekor burung gagak merasa sangat kehausan.
Download Plaintext

2. Pallier

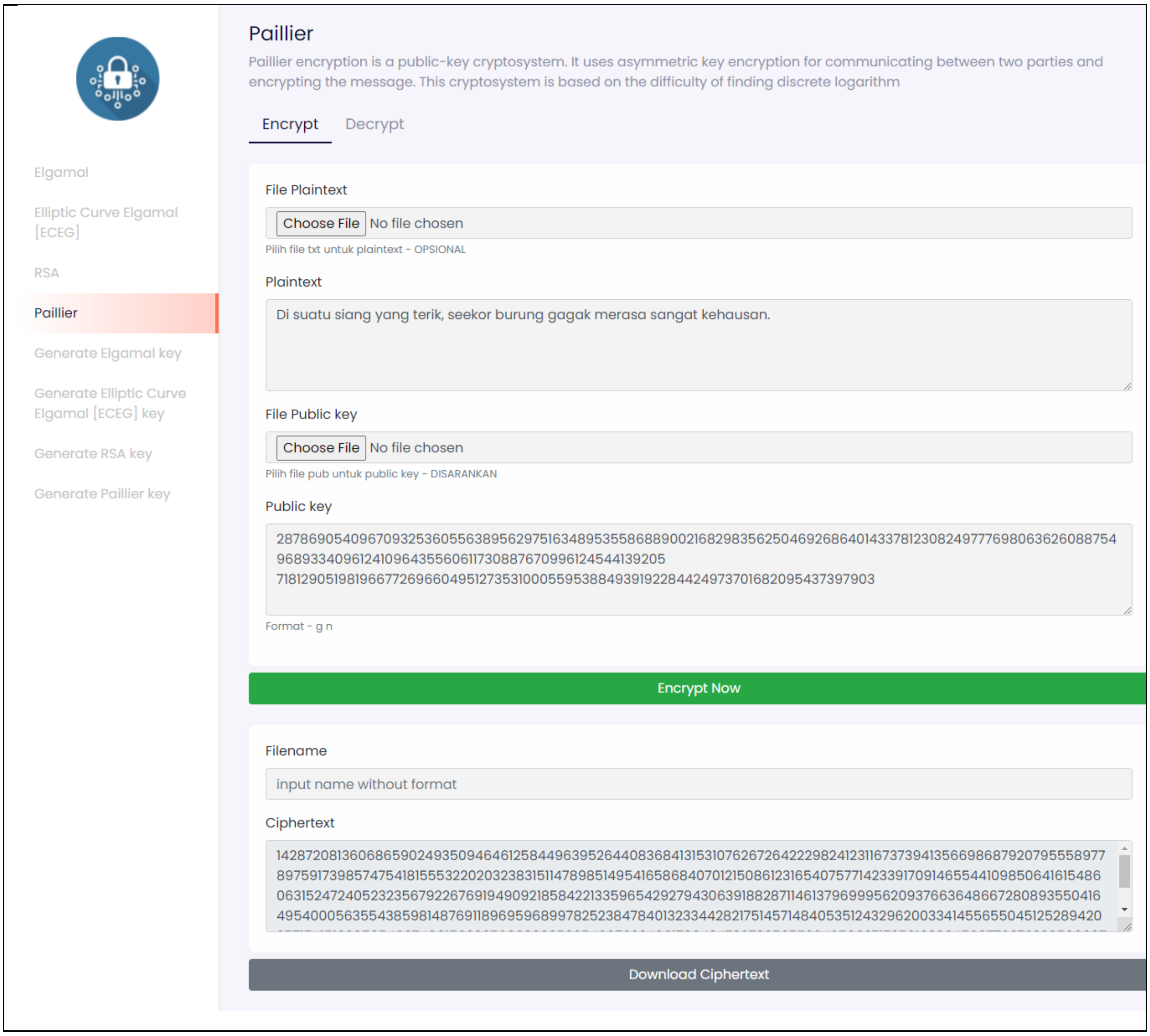
a. Generate Key

Ukuran Kunci	128
Kunci Publik	pallier_key.pub
Kunci Privat	pallier_key.pri
Hasil	



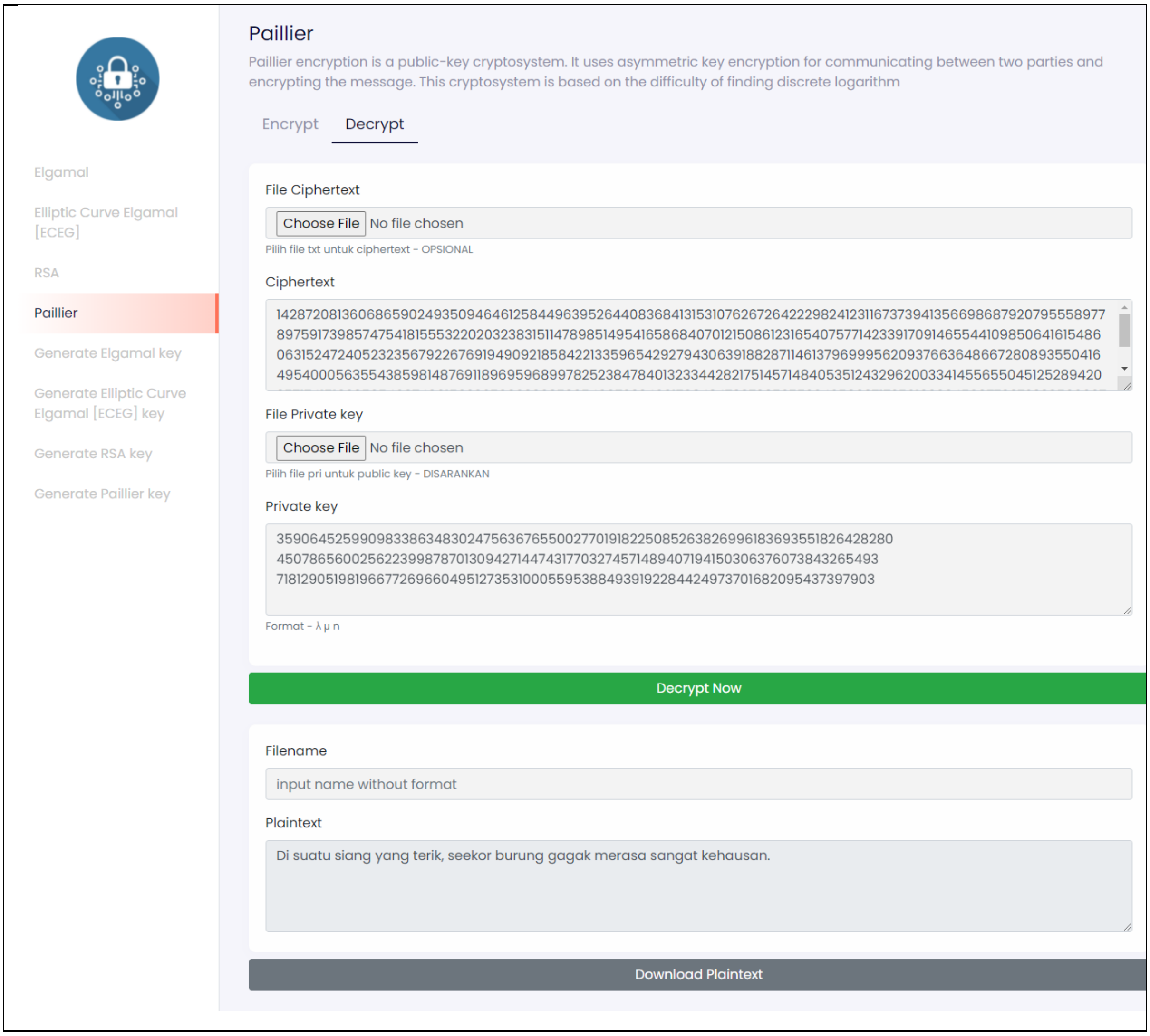
b. Enkripsi

Kunci Publik	pallier_key.pub
Plaintext	Di suatu siang yang terik, seekor burung gagak merasa sangat kehausan.
Ciphertext Hasil	pallier_cipher.txt
Hasil	



c. Dekripsi

Kunci Privat	pallier_key.pri
Ciphertext	pallier_cipher.txt
Plaintext Hasil	Di suatu siang yang terik, seekor burung gagak merasa sangat kehausan.
Hasil	

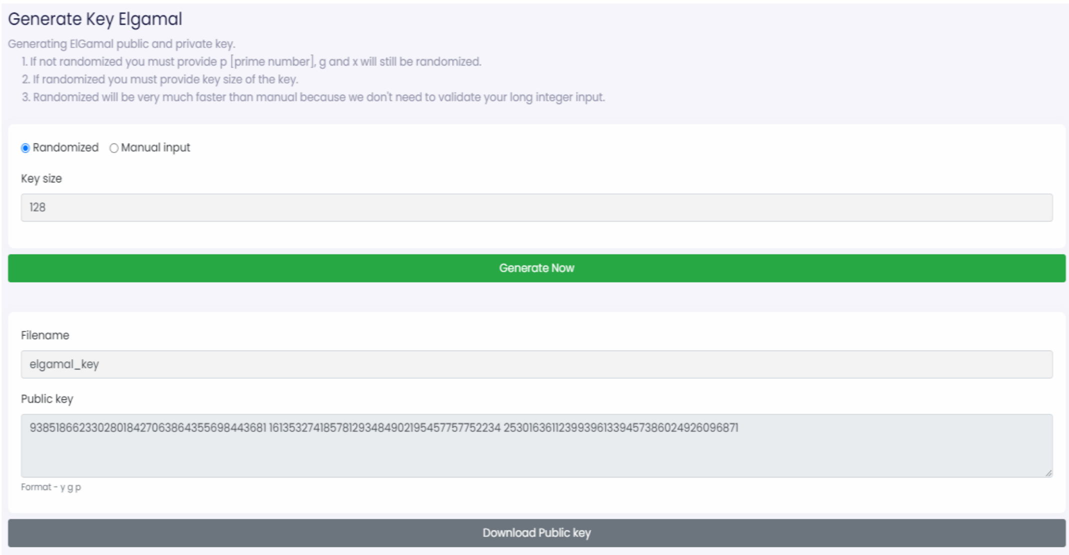


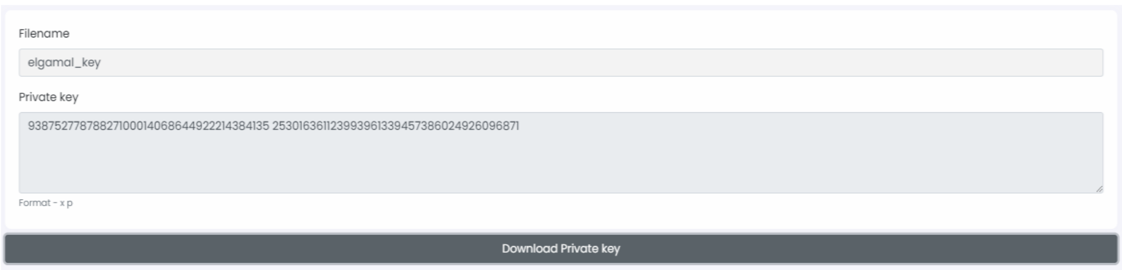
Paillier
Paillier encryption is a public-key cryptosystem. It uses asymmetric key encryption for communicating between two parties and encrypting the message. This cryptosystem is based on the difficulty of finding discrete logarithm
Encrypt
Decrypt
Elgamal
Elliptic Curve Elgamal [ECEG]
RSA
Paillier
Generate Elgamal key
Generate Elliptic Curve Elgamal [ECEG] key
Generate RSA key
Generate Paillier key
File Ciphertext
Choose File No file chosen
Pilih file txt untuk ciphertext - OPSIONAL
Ciphertext
1428720813606865902493509464612584496395264408368413153107626726422298241231167373941356698687920795558977
8975917398574754181555322020323831511478985149541658684070121508612316540757714233917091465544109850641615486
0631524724052323567922676919490921858422133596542927943063918828711461379699956209376636486672808935504166
4954000563554385981487691189695968997825238478401323344282175145714840535124329620033414556550451252894200
File Private key
Choose File No file chosen
Pilih file pri untuk public key - DISARANKAN
Private key
359064525990983386348302475636765500277019182250852638269961836935518264282800
45078656002562239987870130942714474317703274571489407194150306376073843265493
71812905198196677269660495127353100055953884939192284424973701682095437397903
Format - λ μ n
Decrypt Now
Filename
input name without format
Plaintext
Di suatu siang yang terik, seekor burung gagak merasa sangat kehausan.
Download Plaintext

3. Elgamal

a. Generate Key

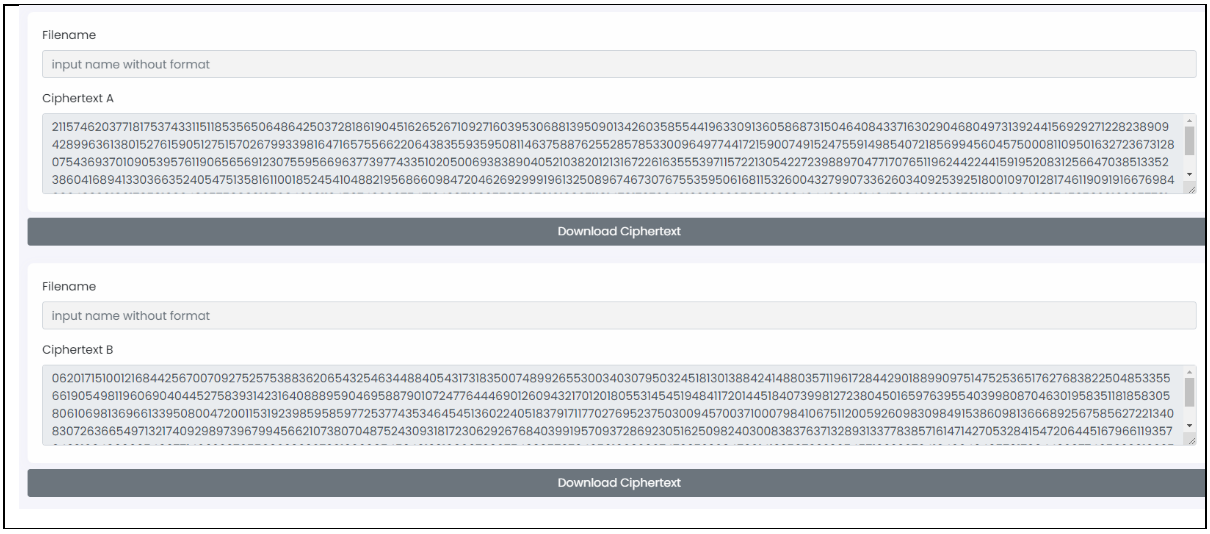
Ukuran Kunci	128
Kunci Publik	elgamal_key.pub
Kunci Privat	elgamal_key.pri
Hasil	





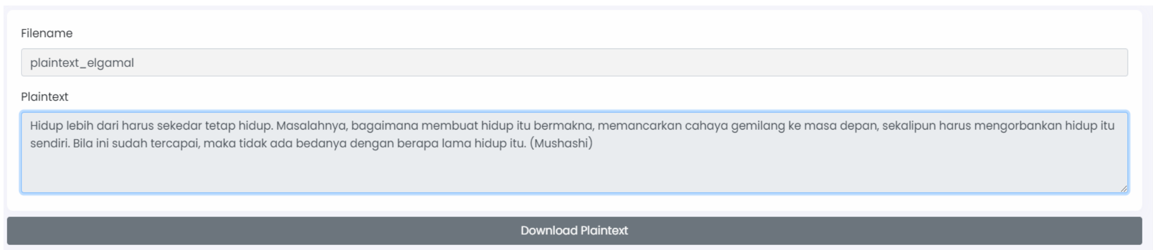
a. Enkripsi

Kunci Publik	Elgamal_key.pub
Plaintext	Hidup lebih dari harus sekedar tetap hidup. Masalahnya, bagaimana membuat hidup itu bermakna, memancarkan cahaya gemilang ke masa depan, sekalipun harus mengorbankan hidup itu sendiri. Bila ini sudah tercapai, maka tidak ada bedanya dengan berapa lama hidup itu. (Mushashi)
Ciphertext	ciphertext_elgamal_1.txt, ciphertext_elgamal_2.txt
Hasil	



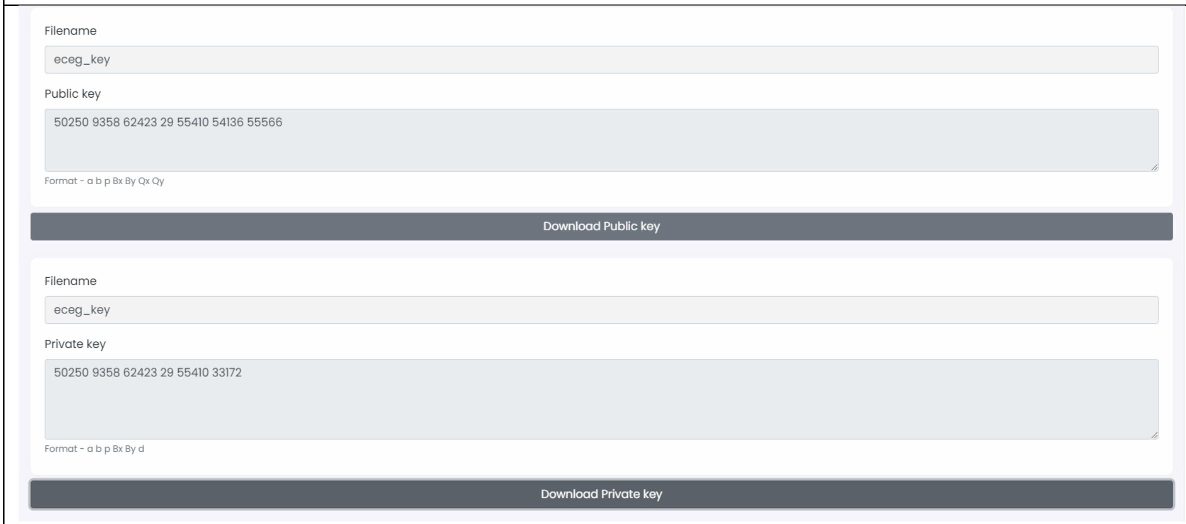
b. Dekripsi

Kunci Privat	elgamal_key.pri
Ciphertext	ciphertext_elgamal_1.txt, ciphertext_elgamal_2.txt
Plaintext	plaintext_elgamal.txt
Hasil	



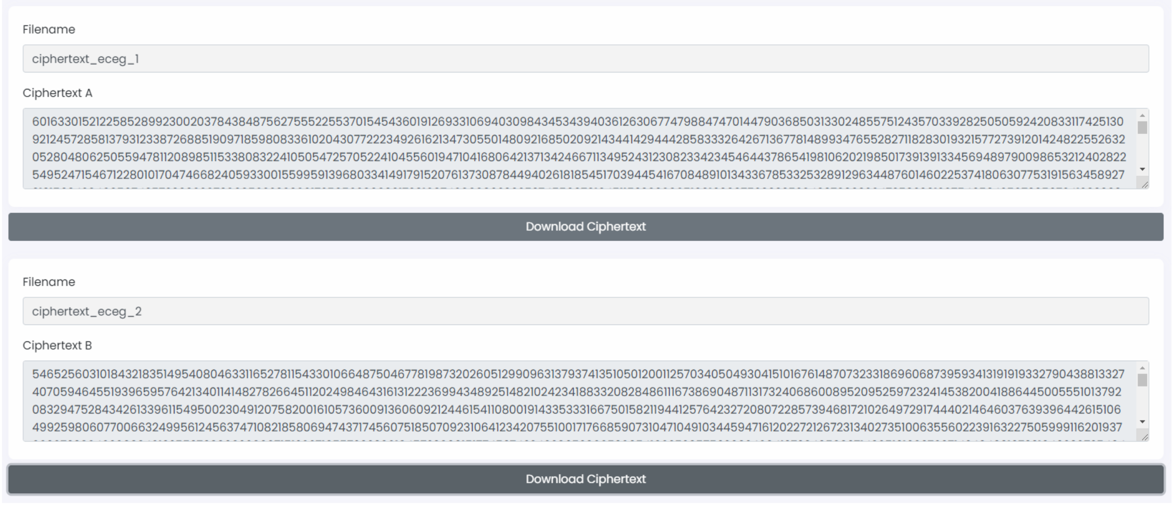
4. ECEG

a. Generate Key

Ukuran Kunci	16
Kunci Publik	eceg_key.pub
Kunci Privat	eceg_key.pri
Hasil	
	

b. Enkripsi

Kunci Publik	
Plaintext	Hidup lebih dari harus sekedar tetap hidup. Masalahnya, bagaimana membuat hidup itu bermakna, memancarkan cahaya gemilang ke masa depan, sekalipun harus mengorbankan hidup itu sendiri. Bila ini sudah tercapai, maka tidak ada bedanya dengan berapa lama hidup itu. (Mushashi)
Ciphertext	ciphertext_eceg_1.txt, ciphertext_eceg _2.txt
Hasil	



c. Dekripsi

Kunci Privat	elgamal_key.pri
Ciphertext	ciphertext_eceg_1.txt, ciphertext_eceg _2.txt
Plaintext	plaintext_eceg.txt
Hasil	

BAB IV

Lampiran

1. **Pranala github yang berisi kode program**

   https://github.com/AndhikaRei/Modern-Cryptography