IF4020 Kriptografi

Oleh :

Andjani Kiranadewi 13518109

Semester I Tahun 2021/2022

INSTITUT TEKNOLOGI BANDUNG

A. Source Code

Source code dapat dilihat di https://github.com/ifakid/tucil4_kriptografi/tree/master

Kelas RSA

```python
class RSA:
    def __init__(self, key: RSAKey):
        self.key = key

    def generate_key(self, key_length):
        self.key.generate_key(key_length)

    def encrypt(self, m: bytes):
        if not self.key.e or not self.key.n:
            raise ValueError('Key invalid')
        if isinstance(m, str):
            m = m.encode()
        total_size = math.ceil(self.key.n.bit_length() / 8)
        chunk_size = total_size - 11  # total_size = 3 bytes flags + at least 8 bytes padding + message chunk
        plain_chunks = [m[i:i+chunk_size] for i in range(0, len(m), chunk_size)]
        encrypted_chunks = []

        def nonzero_byte():
            b = random.getrandbits(8)
            while b == 0x00:
                b = random.getrandbits(8)
            return b.to_bytes(1, byteorder='big')

        for chunk in plain_chunks:
            '''c_len = len(chunk)
            p_len = total_size - 3 - c_len
            padding = nonzero_byte() * p_len
            m_chunk = b'\x00\x02'+padding+b'\x00'+chunk  # PKCS#1 v1.5 padding'''
            m_chunk = padding.pkcs1_pad(chunk, total_size)
            encrypted_chunks.append(self._encrypt(m_chunk, total_size))
        encrypted = b''.join(encrypted_chunks)
        return encrypted

    def _encrypt(self, m: bytes, block_size):
        i = int.from_bytes(m, byteorder='big')
        return pow(i, self.key.e, self.key.n).to_bytes(block_size, byteorder='big')

    def decrypt(self, m: bytes):
        if not self.key.d or not self.key.n:
            raise ValueError('Key invalid')
        block_size = math.ceil(self.key.n.bit_length() / 8)
        encrypted_blocks = [m[i:i+block_size] for i in range(0, len(m), block_size)]
        decrypted_blocks = []
        for block in encrypted_blocks:
            decrypted_block = self._decrypt(block, block_size)
            decrypted_blocks.append(decrypted_block.split(b'\x00', 2)[-1])
        decrypted = b''.join(decrypted_blocks)
        return decrypted

    def _decrypt(self, m: bytes, block_size):
        i = int.from_bytes(m, byteorder='big')
        return pow(i, self.key.d, self.key.n).to_bytes(block_size, byteorder='big')
```

Kelas RSAKey

```
class RSAKey:
    def __init__(self):
        self.p = None
        self.q = None
        self.n = None
        self.e = None
        self.d = None
        self.phi = None

    def generate_key(self, key_length):
        while True:
            self.p = prime.generate_prime(key_length // 2)
            self.q = prime.generate_prime(key_length // 2)
            self.n = self.p * self.q
            if self.n.bit_length():
                break
        self.phi = prime.toitent(self.p, self.q)

        while True:
            self.e = random.randint(2, self.phi)
            if prime.is_coprime(self.e, self.phi):
                break

        self.d = pow(self.e, -1, self.phi)
        assert ((self.e * self.d) % self.phi == 1)
```

Kelas ElGamal

```python
class ElGamal:
    def __init__(self, key: ElGamalKey):
        self.key = key

    def encrypt(self, m: bytes):
        # TODO: Consider padding
        if not (self.key.p and self.key.g and self.key.y):
            raise ValueError('Key invalid!')
        block_length = (self.key.p - 1).bit_length()-1  # Interval [0, p-1]
        plain_blocks = [m[i:i + block_length] for i in range(0, len(m), block_length)]
        encrypted_blocks = []
        for i, block in enumerate(plain_blocks):
            print(f'Encrypting {i+1}/{len(plain_blocks)}')
            encrypted_blocks.append(self._encrypt(block, block_length))
        return b''.join(encrypted_blocks)

    def _encrypt(self, m:bytes, block_length):
        k = random.randint(1, self.key.p-2)
        i = int.from_bytes(m, byteorder='big')
        a = (pow(self.key.g, k, self.key.p)).to_bytes(block_length, byteorder='big')
        b = ((i * pow(self.key.y, k, self.key.p)) % self.key.p).to_bytes(block_length, byteorder='big')
        return a+b

    def decrypt(self, m: bytes):
        if not (self.key.p and self.key.x):
            raise ValueError('Key invalid!')
        block_length = (self.key.p - 1).bit_length() - 1  # Interval [0, p-1]
        encrypted_blocks = [m[i:i + block_length*2] for i in range(0, len(m), block_length*2)]
        decrypted_blocks = []
        for i, block in enumerate(encrypted_blocks):
            decrypted_blocks.append(self._decrypt(block))
        return b''.join(decrypted_blocks)

    def _decrypt(self, m: bytes):
        block_length = len(m)//2
        a = int.from_bytes(m[:block_length], byteorder='big')
        b = int.from_bytes(m[block_length:], byteorder='big')
        a_inv = pow(a, self.key.p - 1 - self.key.x, self.key.p)
        return ((b * a_inv) % self.key.p).to_bytes(block_length, byteorder='big')
```

Kelas ElGamalKey

```
class ElGamalKey:
    def __init__(self):
        self.p = 0
        self.g = 0
        self.x = 0
        self.y = 0

    def generate_key(self, bit_length):
        self.p = prime.generate_prime(bit_length)
        while self.p.bit_length() != bit_length:
            self.p = prime.generate_prime(bit_length)

        self.x = random.randint(1, self.p-2)
        self.g = random.randrange(self.p)
        self.y = pow(self.g, self.x, self.p)
```

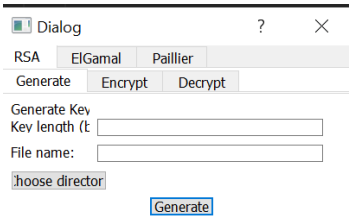
Kelas Paillier

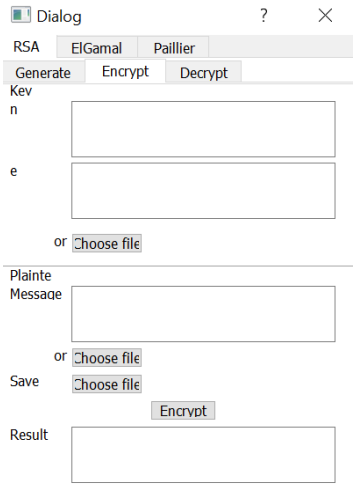
```python
class Paillier:
    def __init__(self, key: PaillierKey):
        self.key = key

    def encrypt(self, m:bytes):
        if not (self.key.g and self.key.n):
            raise ValueError('Key invalid!')
        block_length = self.key.n.bit_length() - 1
        plain_blocks = [m[i:i + block_length] for i in range(0, len(m), block_length)]
        encrypted_blocks = []
        for block in plain_blocks:
            encrypted_blocks.append(self._encrypt(block, block_length))
        return b''.join(encrypted_blocks)

    def _encrypt(self, m: bytes, block_length):
        while True:
            r = random.randrange(0, self.key.n)
            if is_coprime(r, self.key.n):
                break
        i = int.from_bytes(m, byteorder='big')
        c = (pow(self.key.g, i, self.key.n**2)*pow(r, self.key.n, self.key.n**2)) % (self.key.n**2)
        return c.to_bytes(block_length, byteorder='big')

    def decrypt(self, m: bytes):
        if not (self.key.lam and self.key.mu):
            raise ValueError('Key invalid!')
        block_length = self.key.n.bit_length() - 1
        encrypted_blocks = [m[i:i + block_length] for i in range(0, len(m), block_length)]
        decrypted_blocks = []
        for block in encrypted_blocks:
            decrypted_blocks.append(self._decrypt(block, block_length))
        return b''.join(decrypted_blocks)

    def _decrypt(self, m: bytes, block_length):
        i = int.from_bytes(m, byteorder='big')
        lc = self.key.L(pow(i, self.key.lam, self.key.n ** 2))
        p = (lc * self.key.mu) % self.key.n
        return p.to_bytes(block_length, byteorder='big')
```

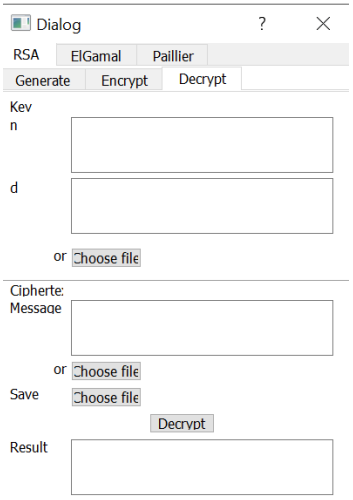
Kelas PaillierKey

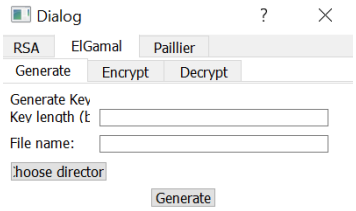
```python
class PaillierKey:
    def __init__(self):
        self.p = 0
        self.q = 0
        self.n = 0
        self.phi = 0
        self.lam = 0
        self.g = 0
        self.mu = 0

    def generate_key(self, key_length):
        while True:
            self.p = prime.generate_prime(key_length // 2)
            self.q = prime.generate_prime(key_length // 2)
            self.n = self.p * self.q
            self.phi = prime.toitent(self.p, self.q)
            if prime.is_coprime(self.n, self.phi):
                break
        self.lam = prime.lcm(self.p - 1, self.q - 1)
        self.g = random.randint(1, self.n ** 2)

        self.mu = pow(self.L(pow(self.g, self.lam, self.n ** 2)), -1, self.n)

    def L(self, x):
        return (x - 1) // self.n
```

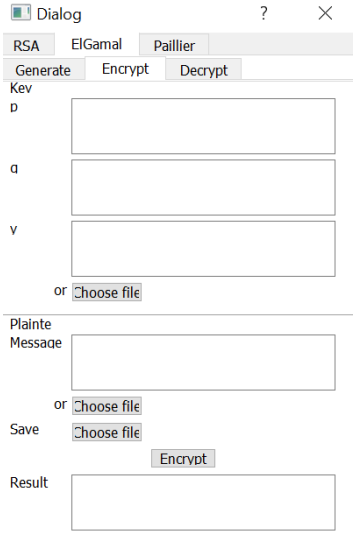
B. Screenshot

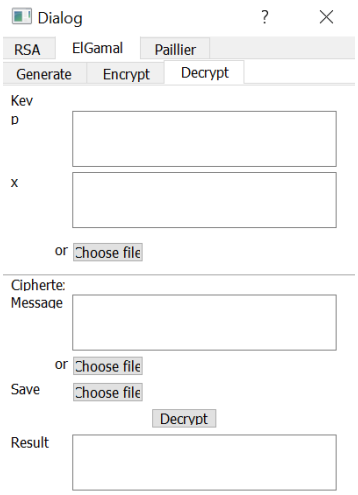


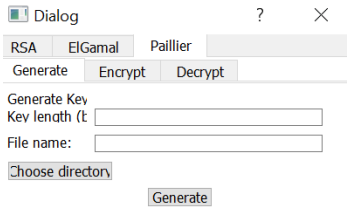

Dialog
?
RSA
ElGamal
Paillier
Generate
Encrypt
Decrypt
Key
n
e
or
Choose file
Plainte
Message
or
Choose file
Save
Choose file
Encrypt
Result

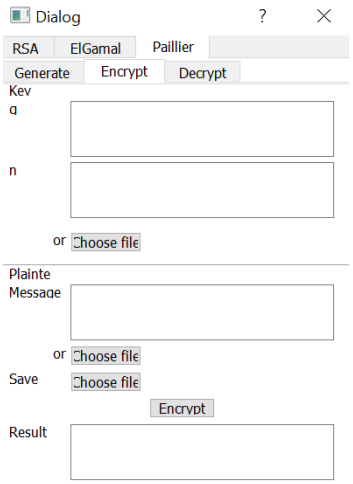

Dialog
?
RSA
ElGamal
Paillier
Generate
Encrypt
Decrypt
Key
n
d
or
Choose file
Cipherte:
Message
or
Choose file
Save
Choose file
Decrypt
Result

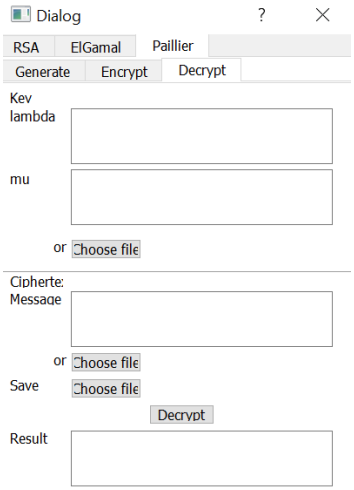

Dialog
RSA
ElGamal
Paillier
Generate
Encrypt
Decrypt
Generate Key
Key length (b
File name:
hoose director
Generate


Dialog
RSA
ElGamal
Paillier
Generate
Encrypt
Decrypt
Key
p
q
y
or
Choose file
Plainte
Message
or
Choose file
Save
Choose file
Encrypt
Result


Dialog
RSA
ElGamal
Paillier
Generate
Encrypt
Decrypt
Key
p
x
or
Choose file
Ciphertext
Message
or
Choose file
Save
Choose file
Decrypt
Result


Dialog
RSA
ElGamal
Paillier
Generate
Encrypt
Decrypt
Generate Key
Key length (b
File name:
Choose directory
Generate





C. Contoh

a. Private key

i. RSA (128 bit)
{"p": 9132772239323549441, "q": 2454735370980459407, "n": 22418539050975934140799657536058041487, "e": 21871710920135821552409246068192584981, "d": 157084405765591666341747653309287040 61}

ii. ElGamal (128 bit)
{"p": 257874603144350564474370317439335016037, "x": 64675031121674517908110953170403819653}

iii. Paillier (128 bit)

{"lam": 7793970403734378423303677983215438373 2, "mu": 762094941166020961877783682047897287 24}

b. Public key
    i. RSA (128 bit)
    {"n": 22418539050975934140799657536058041487, "e": 21871710920135821552409246068192584981}

    ii. ElGamal (128 bit)
    {"p": 25787460314435056447437031743933501603 7, "g": 21162596713967953711121321696774917838 3, "y": 25394607153399989228770401772972411853 7}

    iii. Paillier (128 bit)
    {"n": 15587940807468756849123772196875893509 9, "g": 23395701847185522025041984660259422675614186648061272966829253699066436491346}

c. Plainteks
    Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

d. Cipherteks
    Contoh ciphertext dapat dilihat di folder Example/Result