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

```
class RSA:
    def __init__(self, key: RSAKey, min_pad=8):
        self.key = key
        self.pad = min_pad

    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')
        total_size = math.ceil(self.key.n.bit_length() / 8)
        chunk_size = total_size - 3 - self.pad
        if chunk_size <= 0:
            raise ValueError()
        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 RSA harus diinstansiasi dengan parameter RSAKey untuk enkripsi dan dekripsinya. Karena RSA (dan algoritma-algoritma dibawah) hanya bisa mengenkripsi plainteks yang lebih pendek dari kuncinya, algoritma digunakan seperti halnya block cipher. Agar lebih aman, setiap block dilakukan padding, dengan standar yang digunakan adalah PKCS#1 v1.5.
Data yang dienkripsi harus sudah dalam bentuk bytes (ini dihandle oleh bagian program lainnya)
Enkripsi dilakukan dengan memangkatkan plaintext dengan e (public exponent) dan di mod n. Dekripsi dilakukan dengan memangkatkan ciphertext dengan d (private exponent) dan di mod n.

Kelas RSAKey

```python
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 RSAKey merepresentasikan kunci RSA. Kunci publik dan privat tidak dibedakan, pembedaan dilakukan dengan mengecek properti oleh kelas RSA.
Kelas ini dapat melakukan key generation dengan panjang yang disediakan oleh pengguna.
Berikut tahap pembuatan pasangan Key:

- Pilih dua buah bilangan prima p dan q
- Hitung nilai n dengan mengalikan p dan q
- Hitung nilai phi dengan mengalikan p-1 dan q-1
- Pilih bilangan acak e yang terletak harus kurang dari phi dan ber-PBB 1 dengan phi
- Hitung nilai d dengan menghitung invers modular e terhadap phi

Kelas ElGamal

```python
class ElGamal:
    def __init__(self, key: ElGamalKey, min_pad=8):
        self.key = key
        self.pad = min_pad

    def encrypt(self, m: bytes):
        if not (self.key.p and self.key.g and self.key.y):
            raise ValueError('Key invalid!')
        total_length = math.ceil((self.key.p - 1).bit_length() / 8)  # Interval [0, p-1]
        block_length = total_length - 3 - self.pad
        if block_length <= 0:
            raise ValueError()
        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)}')
            padded = padding.pkcs1_pad(block, total_length)
            encrypted_blocks.append(self._encrypt(padded, total_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 = math.ceil((self.key.p - 1).bit_length() / 8)  # 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)
        decrypted = ((b * a_inv) % self.key.p).to_bytes(block_length, byteorder='big')
        return padding.pkcs1_unpad(decrypted)
```

Kelas ElGamal harus diinstansiasi dengan parameter ElGamalKey untuk enkripsi dan dekripsinya.
Data yang dienkripsi harus sudah dalam bentuk bytes (ini dihandle oleh bagian program lainnya)
Data dienkripsi dengan memilih bilangan bulat k yang jatuh didalam interval [1, p-2]. Kemudian menghitung a = g^k mod p dan b = m*y^k mod p. Ciphertext merupakan gabungan dari a dan b.
Dekripsi dilakukan dengan memisahkan ciphertext menjadi dua bagian a dan b. Kemudian hitung plaintext dengan c = b * a^(p-1-x) mod p

Kelas ElGamalKey

```python
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 ElGamalKey merepresentasikan kunci ElGamal. Berikut tahap key generation:

- Pilih bilangan prima acak p
- Pilih bilangan bulat acak x dengan x didalam interval [1, p-2]
- Pilih bilangan bulat acak g dengan g < p
- Hitung y dengan menghitung g^x mod p

Kelas Paillier

```python
class Paillier:
    def __init__(self, key: PaillierKey, min_pad=8):
        self.key = key
        self.pad = min_pad

    def encrypt(self, m: bytes):
        if not (self.key.g and self.key.n):
            raise ValueError('Key invalid!')
        total_length = math.ceil(self.key.n.bit_length()/8)
        block_length = total_length - 3 - self.pad
        if block_length <= 0:
            raise ValueError("Key is too short!")
        plain_blocks = [m[i:i + block_length] for i in range(0, len(m), block_length)]
        encrypted_blocks = []
        for block in plain_blocks:
            padded = padding.pkcs1_pad(block, total_length)
            encrypted_blocks.append(self._encrypt(padded, total_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
        m_byte = int.from_bytes(m, byteorder='big')
        c = (pow(self.key.g, m_byte, self.key.n**2) * pow(r, self.key.n, self.key.n**2)) % (self.key.n**2)
        return c.to_bytes(block_length*2, byteorder='big')

    def decrypt(self, m: bytes):
        if not (self.key.lam and self.key.mu and self.key.n):
            raise ValueError('Key invalid!')
        block_length = math.ceil(self.key.n.bit_length()/8)
        print(f'block_length {block_length}')
        encrypted_blocks = [m[i:i + block_length*2] for i in range(0, len(m), block_length*2)]
        decrypted_blocks = []
        for block in encrypted_blocks:
            p_text = self._decrypt(block, block_length)
            decrypted_blocks.append(padding.pkcs1_unpad(p_text))
        return b''.join(decrypted_blocks)

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

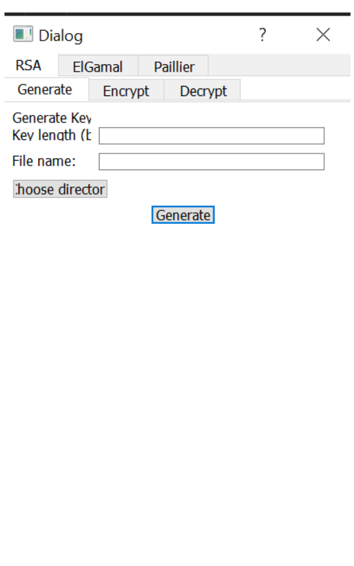
Kelas PaillierKey

```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)

        while True:
            while True:
                self.g = random.randint(2, self.n ** 2)
                if prime.is_coprime(self.g, self.n**2):
                    break
            try:
                lpow = int(self.L(pow(self.g, self.lam, self.n ** 2)))
                self.mu = pow(lpow, -1, self.n)  # will return an error if L results in float
                break
            except:
                pass

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

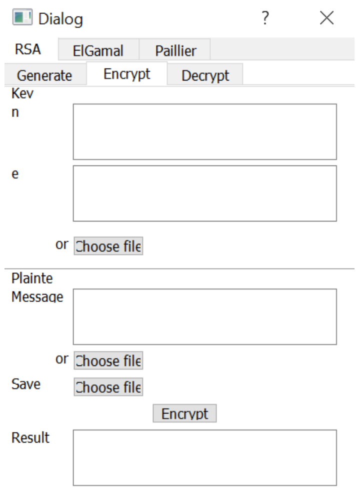
B. Screenshot

Tampilan antar muka aplikasi dibuat menggunakan library PyQt5 dari Python.

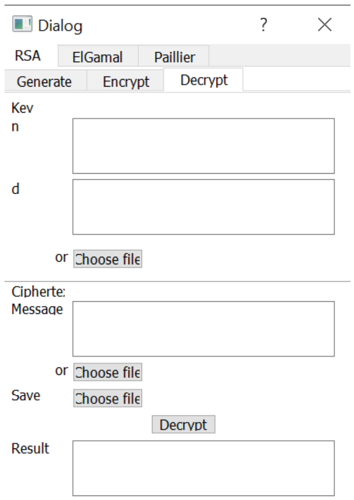
RSA Generate



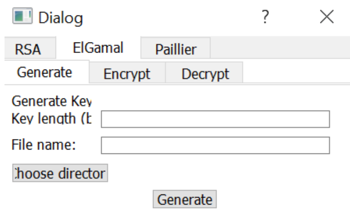
RSA Encrypt



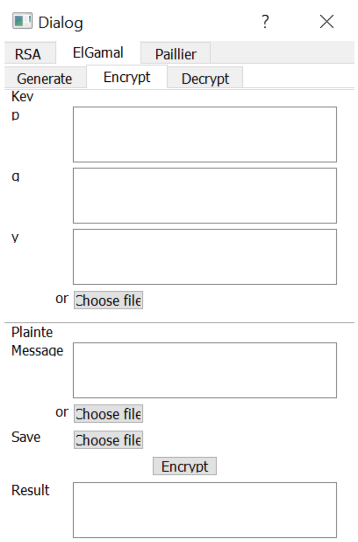
RSA Decrypt



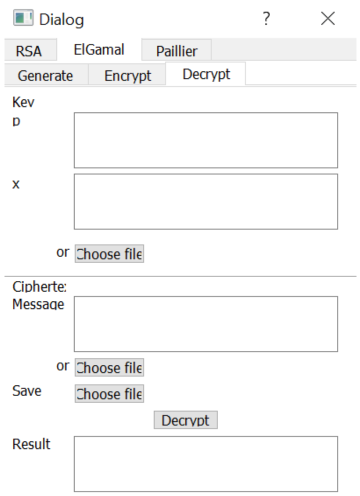
ElGamal Generate



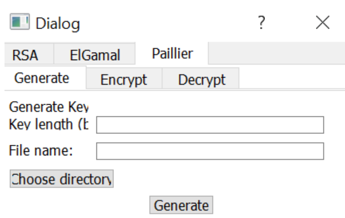
ElGAmal Encrypt



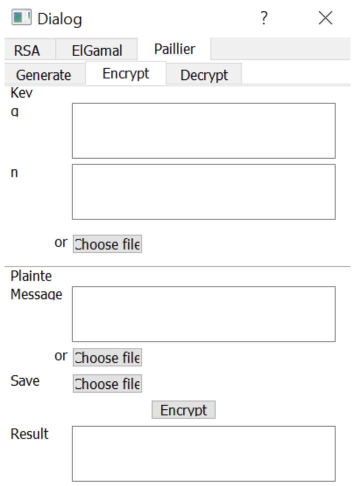
ElGamal Decrypt



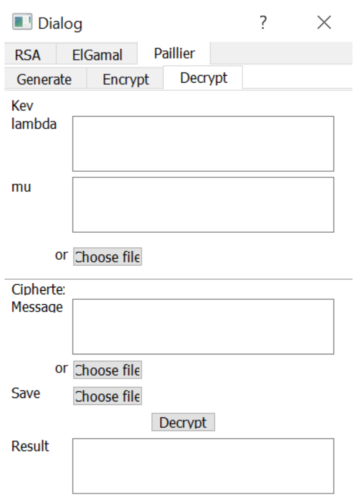
Paillier Generate



Paillier Encrypt



Paillier Decrypt



C. Contoh

Contoh key ada pada folder Example/Key dalam repository. Key umumnya disimpan dengan standar tertentu (seperti DER dan PEM), namun untuk tugas ini kunci disimpan dalam bentuk JSON.

a. Private key

   i. RSA (128 bit)

{"p": 9132772239323549441, "q": 2454735370980459407, "n": 22418539050975934140799657536058041487, "e": 21871710920135821552409246068192584981, "d": 15708440576559166634174765330928704061}

   ii. ElGamal (128 bit)

{"p": 181557488690931335861885139658026681533, "x": 100552721847471079261546603797119502909}

iii. Paillier (128 bit)
{"lam": 3091132021804685378798135592984276596, "mu": 102576261869022061148005083858578635189, "n": 191650185351890493514064882734752037531}

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

   ii. ElGamal (128 bit)
   {"p": 181557488690931335861885139658026681533, "g": 149830678938376478385325969741544773559, "y": 168605243593072853915925070539887129300}

   iii. Paillier (128 bit)
   {"n": 191650185351890493514064882734752037531, "g": 19643541124310090161191555002499123472802489274728353952051416363228147879587}

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. Setiap file ciphertext bernama {algoritma}{panjang key}_enc