LAPORAN KRIPTOGRAFI HIBRIDA

NIM: 221111798
Nama: Naomi Prisella

DAFTAR ISI

A. Pendahuluan

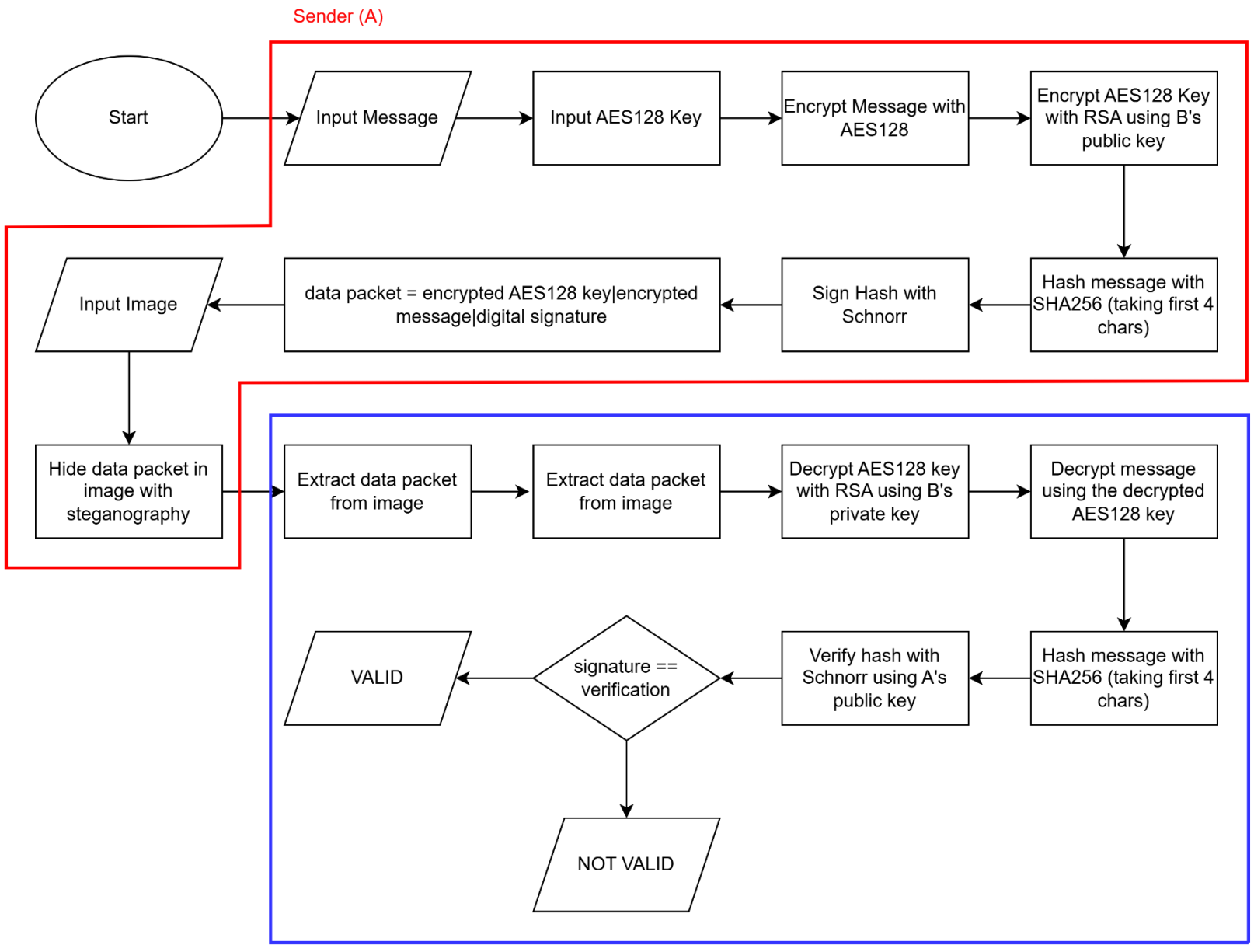
Kode lengkap tersimpan di https://github.com/naomehmi/hybrid-cryptography

Program ini bertujuan untuk mensimulasikan pertukaran pesan antara Alice (A) dan Bob (B), dengan mengimplementasikan kriptografi hibrida yang terdiri dari algoritma berikut.

- AES128
- RSA
- SHA256
- Schnorr

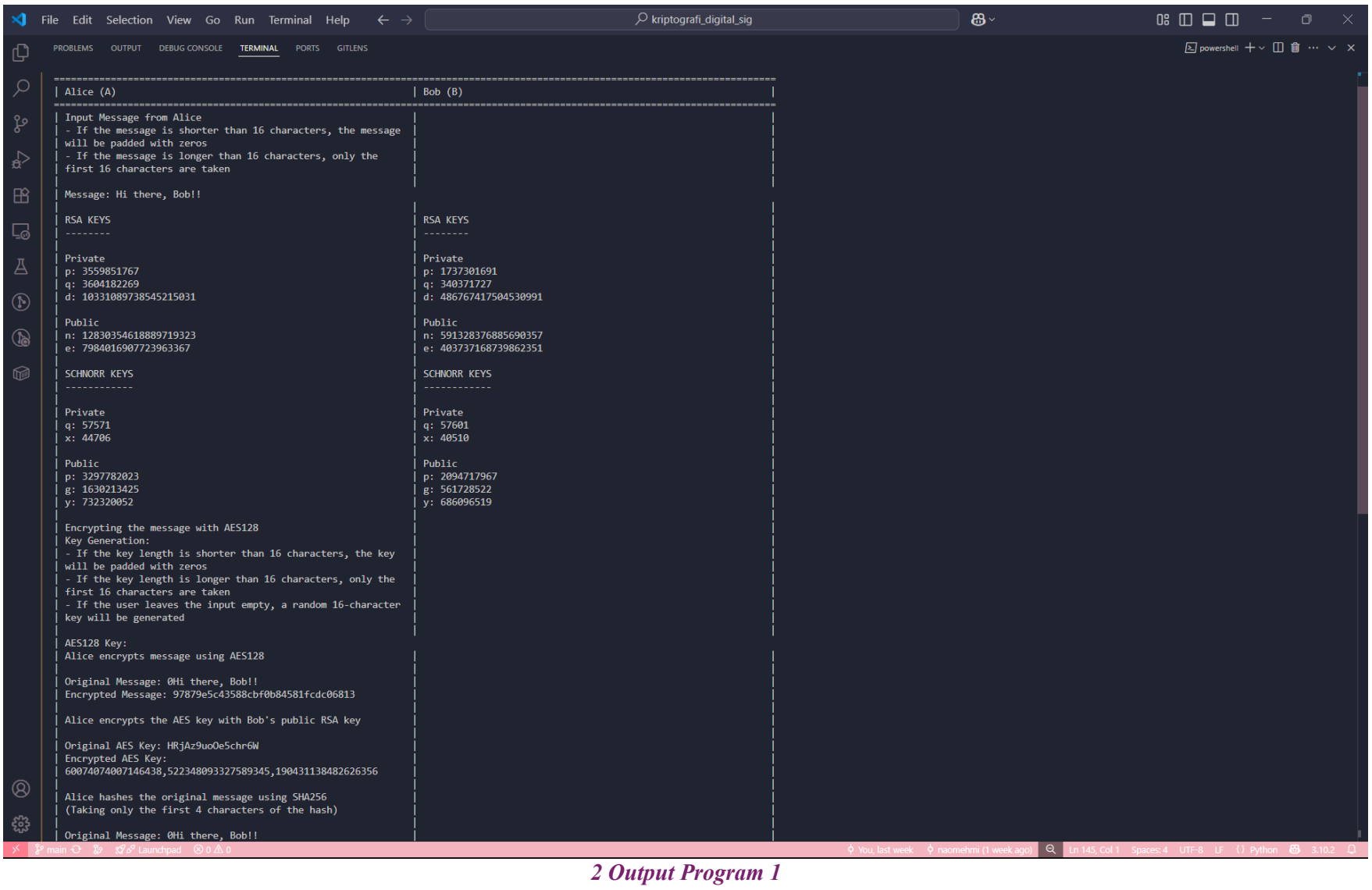
Laporan akan meyertakan flowchart, output program, penjelasan code, serta lampiran fungsi-fungsi pembantu yang digunakan.

B. Flowchart Program

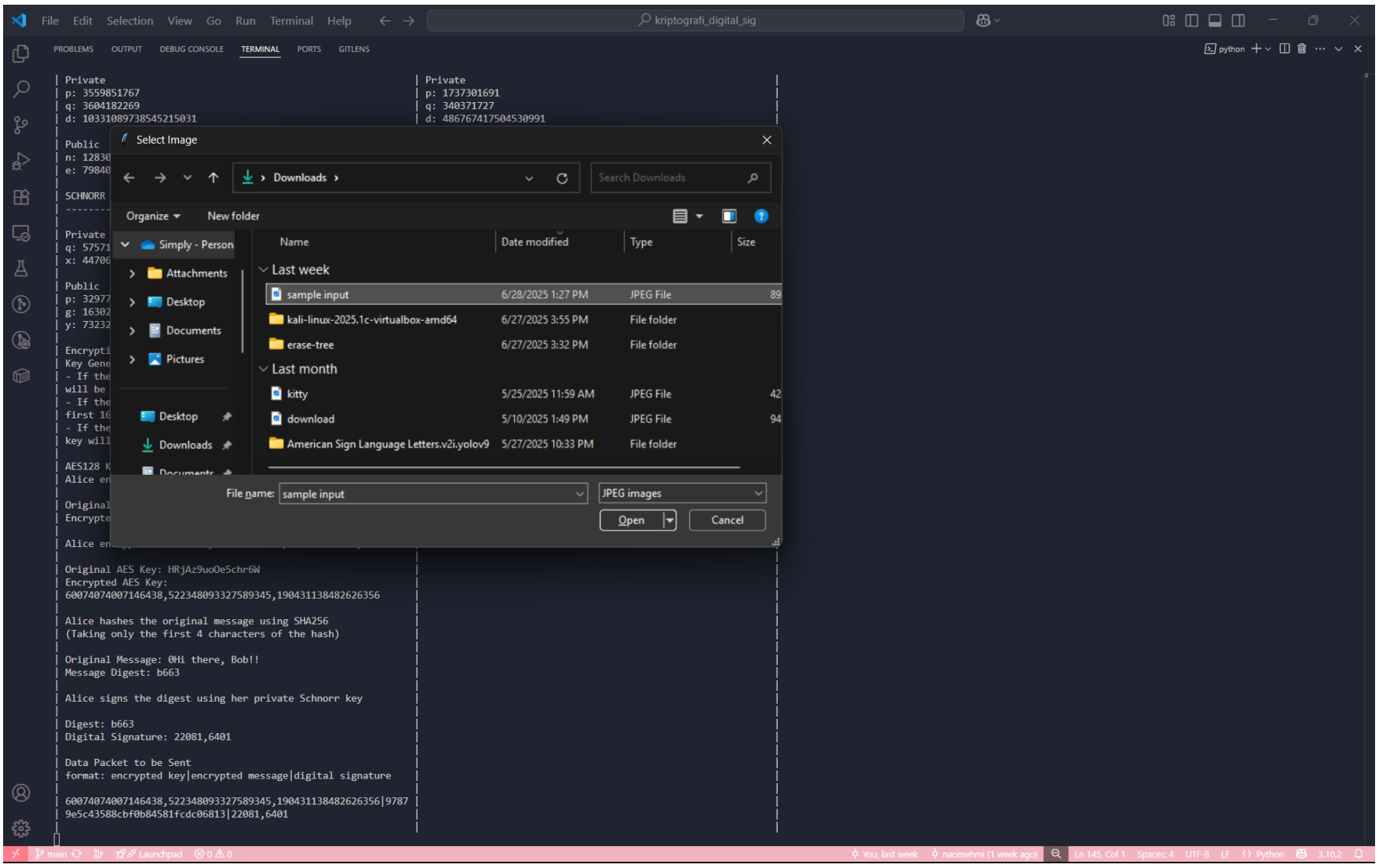


1 Flowchart Program

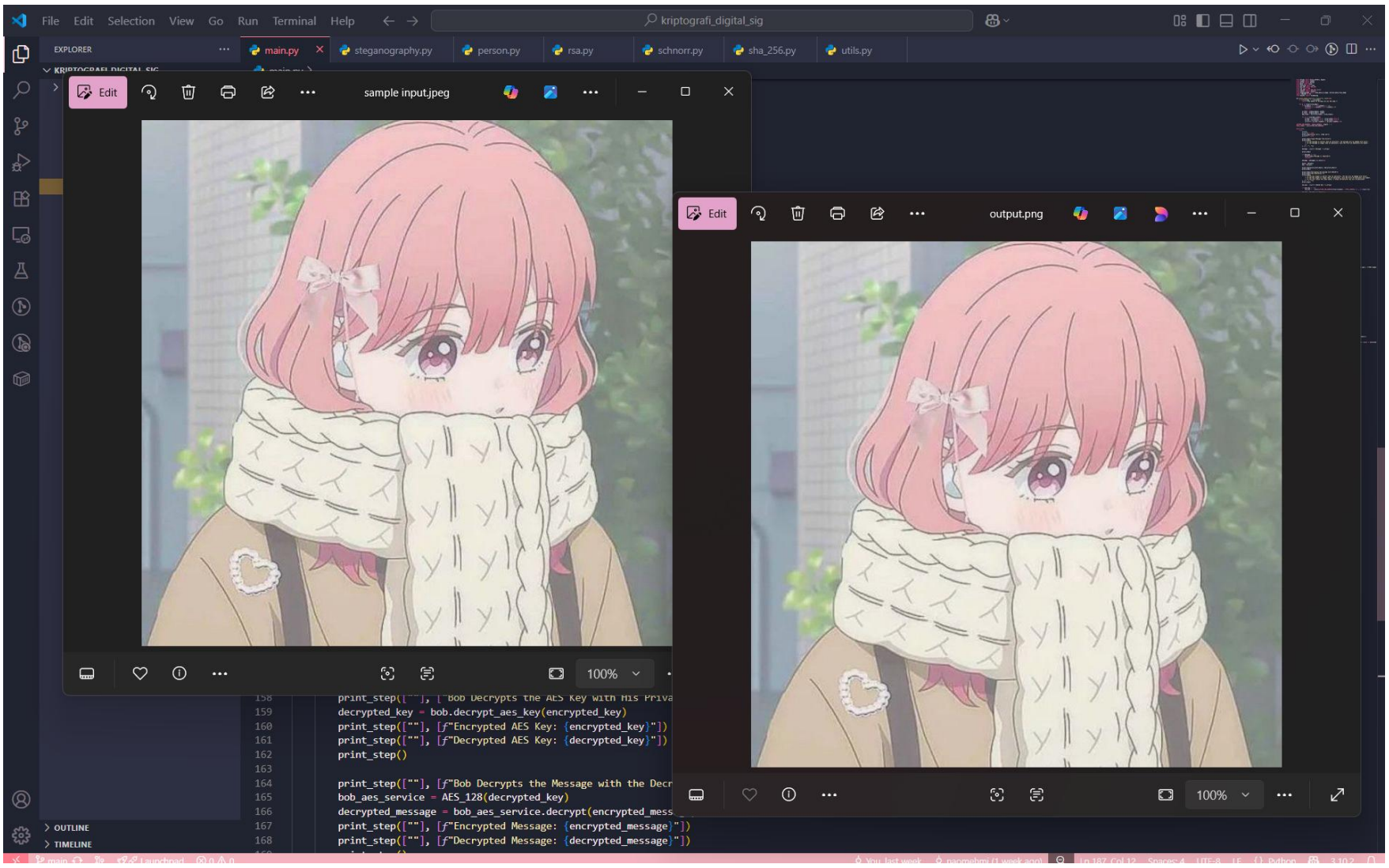
C. Output Program



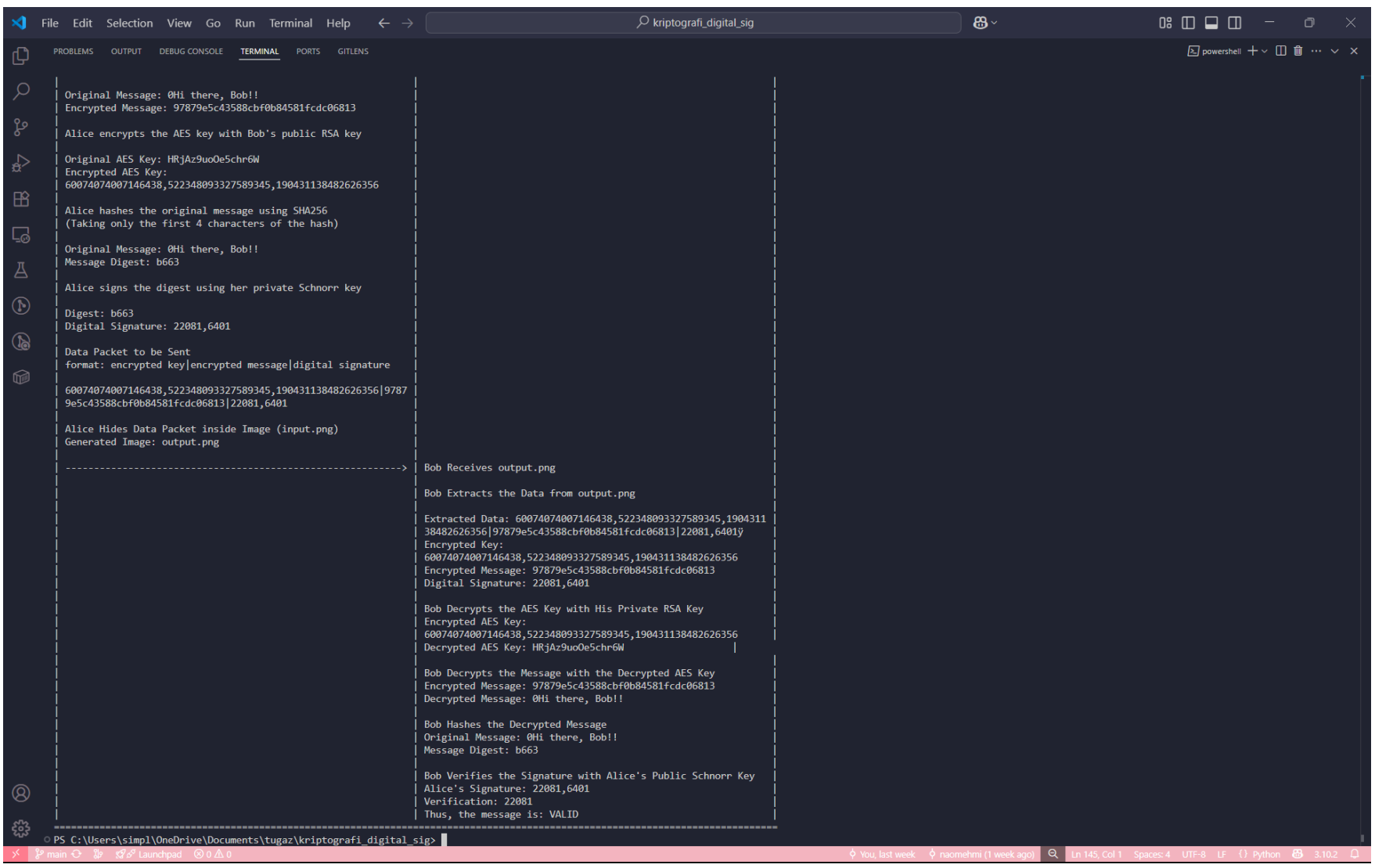
2 Output Program 1



3 Output Program 2 (upload foto untuk penyisipan)



4 Output Program 3 (perbandingan hasil penyisipan)



5 Output Program 4

D. Proses Pengiriman Pesan

1. Input Pesan

Pengguna memasukkan pesan berisi string dengan maksimum panjang 16 karakter. Jika panjang pesan kurang dari 16 karakter, maka awal pesan akan ditambahkan padding dengan karakter '0' sehingga mencapai 16 karakter. Namun, jika panjang pesan melebihi 16 karakter, maka hanya 16 karakter pertama yang diambil.

```
message = input("| Message: ").strip()

print_step()

if (message == ''):
    print_step(["Message is required"])
    continue

if len(message) > 16:
    message = message[:16]

message = message.zfill(16)
```

6 Input Pesan dari Pengirim

2. Generate Kunci Private dan Public Alice dan Bob

Pengirim dan Penerima diinisialisasi sebagai objek $Person$. Pada tahap inisialisasi objek, akan dibuatkan kunci private dan public RSA dan Schnorr masing-masing.

```
alice = Person()
bob = Person()
```

7 Inisialisasi Object Pengirim dan Penerima

Private key dan public key berbentuk dua tuple, dimana tuple pertama merupakan private key, dan tuple kedua merupakan private key.

```
class Person:
    def __init__(self):
        self.__schnorr_private_key, self.__schnorr_public_key = Schnorr.key_gen()
        self.__rsa_private_key, self.__rsa_public_key = RSA().key_gen()
```

8 Proses Inisialisasi Object Person

Kunci-kunci Schnorr dihasilkan dengan pertama inisialisasi p yang merupakan sebuah bilangan prima 32 bit, dan q yang merupakan integer 16 bit yang merupakan faktor prima dari $p - 1$. Kemudian dicari nilai g yang merupakan generator mod dari p. x merupakan bilangan bulat acak lebih kecil daripada q. Kemudian dihitung nilai y yang merupakan hasil dari $g^{-x} mod\ p$. Kunci private merupakan tuple berisi q dan x, dan kunci public merupakan tuple berisi p, g, dan y.

```
class Schnorr:
    @staticmethod
    def key_gen():
        p, q = generate_schnorr_pq()
        g = find_generator(p, q)
        x = randbelow(q)
        inverse_mod_g = inverse_mod(g, p)
        y = fast_mod_exp(inverse_mod_g, x, p)

        # private = q, x
        # public = p g y
        return (q, x), (p, g, y)
```

9 Pembentukan Kunci Schnorr

Kunci-kunci RSA dihasilkan dengan pertama inisialisasi p dan q yang masing-masing merupakan bilangan prima 32 bit. Kemudian n merupakan hasil perkalian dari p dan q. totient_n ($\phi(n)$) merupakan hasil perkalian $p-1$ dan $q-1$. d merupakan sebuah integer antara $MAX(p,q)+1$ dan $\phi(n)-1$ dan relatif prima dengan $\phi(n)$. e dihasilkan dari $d^{-1} mod\ \phi(n)$. Kunci private merupakan tuple berisi p, q, dan d, dan kunci public merupakan tuple berisi n dan e.

```python
class RSA:
    def key_gen(self):
        p, q = generate_large_primes(32, 2)
        n = p * q
        totient_n = (p-1) * (q-1)
        d = generate_valid_d(p, q, totient_n)
        e = inverse_mod(d, totient_n)

        # private = p, q, d
        # public = n, e
        return (p, q, d), (n, e)
```

10 Pembentukan Kunci RSA

3. Input Kunci AES128

```python
aes_key = input("| AES128 Key: ").strip()

if (aes_key == ''):
    aes_key = ''.join([LETTERS_AND_NUMBERS[floor(random() * TOTAL_CHARS)] for _ in range(16)])

aes_key = aes_key[:16].zfill(16)
```

11 Input Kunci AES128

Sebelum enkripsi pesan menggunakan algoritma AES128, pengguna input kunci AES128 yang merupakan string dengan panjang maksimum 16 karakter. Jika user membiarkan isi input kosong, akan digenerate string random dengan panjang 16 karakter. Jika user input lebih pendek dari 16 karakter, maka akan ditambahkan padding dengan karakter '0' sehingga string mencapai panjang 16 karakter. Jika input user lebih panjang dari 16 karakter, hanya 16 karakter pertama yang diambil.

4. Enkripsi Kunci Simetris AES128

Kunci tersebut digunakan untuk inisialisasi service AES yang merupakan object dengan class AES_128.

```python
alice_aes_service = AES_128(aes_key)
```

12 Inisialisasi Object AES_128

Saat proses inisialisasi, pertama merupakan deklarasi matriks ukuran block $\_Nb$ dan kunci $\_Nk$ sebesar 4, dan jumlah ronde $\_Nr$ sebesar 10. Kemudian deklarasi array *Forward S-Box* dan *Inverse S-Box* yang akan digunakan saat pada saat key scheduling, enkripsi, dan dekripsi. Input kunci akan melalui tahap key scheduling.

Key scheduling diawali dengan mengubah setiap karakter kunci menjadi nilai ASCII masing-masing, dan disusun secara menurun dalam sebuah matriks berukuran 4x4 yang dianggap sebagai ronde 0. Matriks tersebut disimpan dalam variabel w yang akan menyimpan gabungan semua mutasi key pada setiap ronde, dimana $w[0]$-$w[3]$ menyimpan key pada ronde 0, $w[4]$-$w[7]$ menyimpan key pada ronde 1, dan seterusnya. Pada setiap ronde, matriks akan melalui proses berikut.

1. Ambil nilai kolom sebelumnya yang disimpan dalam variabel temp.
2. Untuk kolom pertama setiap ronde, temp ditransformasi dimana baris pertama dipindahkan ke paling bawah dan baris dua, tiga, dan empat dipindahkan ke atas. Setiap nilai pada temp digantikan dengan nilai Forward S-Box masing-masing. Baris paling atas XOR dengan nilai $rcon$ (dinisialisasi pertama kali dengan nilai 1). Nilai $rcon$ melakukan perkalian Galois Field nilai $rcon$ dan 2
3. Nilai kolom tersebut XOR dengan kolom keempat sebelumnya lalu diappend ke w.

Nilai w kemudian ditranspose dan disimpan dalam array tiga dimensi bernama $round\_keys$ dengan panjang 11 x 4 x 4, dimana masing-masing ronde berbentuk array dua dimensi 4 x 4 di dalamnya mulai dari ronde 0 hingga 10.

```python
class AES_128:
    __Nb = 4  # Block size (4 words = 16 bytes)
    __Nk = 4  # Key length (4 words = 16 bytes)
    __Nr = 10 # Number of rounds
    __forwardSBox, __inverseSBox = generate_aes_sbox()

    def __init__(self, key: str):
        self.__roundKeys = self.__key_scheduling([ord(c) for c in key])

    def __key_scheduling(self, key_bytes):
        def sub_word(word):
            return [self.__forwardSBox[b] for b in word]

        def rot_word(word):
            return word[1:] + word[:1]

        w = []
        for i in range(self.__Nk):
            w.append(key_bytes[4 * i : 4 * (i + 1)])

        rcon = 1
        for i in range(self.__Nk, self.__Nb * (self.__Nr + 1)):
            temp = w[i - 1][:]
            if i % self.__Nk == 0:
                temp = sub_word(rot_word(temp))
                temp[0] ^= rcon
                rcon = gf_mul(rcon, 2)
            w.append([a ^ b for a, b in zip(w[i - self.__Nk], temp)])

        # Convert to round key matrices
        round_keys = []
        for r in range(self.__Nr + 1):
            currentRound = []
            for c in range(4):
                currentRound.append([w[4 * r + c][row] for row in range(4)])
            round_keys.append(currentRound)
        return round_keys
```

13 Proses Inisialisasi dan Key Scheduling AES128

Pada proses enkripsi, pertama diinisialisasi sebuah matriks 4x4 berisi ASCII dari setiap karakter pada plaintext (pesan) bernama $state\_array$. Pada ronde pertama, dilakukan proses add round key, dimana $state\_array$ XOR dengan $round\_key$ ronde pertama (index nol). Kemudian untuk setiap ronde berikutnya, setiap nilai $state\_array$ digantikan dengan nilai forward S-Box nilai tersebut. Setelah itu, $state\_array$ akan melalui tahap shift row, dimana setiap index i dari 1 sampai 3, i kolom pertama pada baris i dipindahkan ke belakang. Untuk setiap ronde kecuali ronde terakhir, dilakukan proses mix columns, dimana $state\_array$ melakukan operasi perkalian Galois Field dengan matriks heksadesimal bernilai berikut.

$$\begin{matrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{matrix}$$

Setelah itu, iterasi ronde diakhiri dengan operasi add round key. Setiap nilai dalam $state\_array$ setelah iterasi di-concantenate dalam bentuk heksadesimal 8 bit sebagai hasil akhir enkripsi.

```python
def encrypt(self, plaintext: str):
    state_array = [ [ 0 for _ in range(4) ] for _ in range(4) ]

    # put plaintext into matrix
    for i in range(16):
        row = i % 4
        col = i // 4
        state_array[col][row] = ord(plaintext[i])

    # add round key
    for i in range(4):
        for j in range(4):
            state_array[i][j] ^= self.__roundKeys[0][i][j]

    for i in range(1, self.__Nr + 1):
        # sub bytes
        for j in range(4):
            for k in range(4):
                state_array[j][k] = self.__forwardSBox[state_array[j][k]]

        # shift rows
        for r in range(1, 4):
            row = [state_array[c][r] for c in range(4)]
            row = row[r:] + row[:r]
            for c in range(4):
                state_array[c][r] = row[c]

        # mix columns
        if (i != self.__Nr):
            for col in range(4):
                a = state_array[col]
                temp = [
                    gf_mul(0x02, a[0]) ^ gf_mul(0x03, a[1]) ^ a[2] ^ a[3],
                    a[0] ^ gf_mul(0x02, a[1]) ^ gf_mul(0x03, a[2]) ^ a[3],
                    a[0] ^ a[1] ^ gf_mul(0x02, a[2]) ^ gf_mul(0x03, a[3]),
                    gf_mul(0x03, a[0]) ^ a[1] ^ a[2] ^ gf_mul(0x02, a[3]),
                ]
                for x in range(4):
                    state_array[col][x] = temp[x]

        # add round key
        for j in range(4):
            for k in range(4):
                state_array[j][k] ^= self.__roundKeys[i][j][k]

    return ''.join(f'{state_array[c][r]:02x}' for c in range(4) for r in range(4))
```

14 Proses enkripsi AES128

5. Enkripsi Kunci AES menggunakan RSA

String kunci AES128 yang digunakan enkripsi pesan kemudian akan dienkripsi meggunakan RSA. Proses enkripsi RSA dipanggil melalui metode static *encrypt* pada kelas RSA dengan argument plaintext yang berupa kunci AES, dan public key RSA Bob yang akan digunakan pada operasi enkripsi.

```python
encrypted_key = RSA.encrypt(aes_key, bob.get_rsa_public_keys())
```

15 Pemanggilan Metode Enkripsi RSA

Proses enkripsi RSA diawali dengan menghitung nilai b yang merupakan panjang bit n dikurangi 1. Plaintext kemudian diubah menjadi biner berdasarkan ASCII, yang kemudian dibagi menjadi blok-blok yang berukuran b bit, dan ditambahkan padding 0 di belakang jika jumlah bit pada blok tidak sampai b. Kemudian setiap blok melakukan operasi $block[i]^e mod\ n$ yang akan menghasilkan sebuah array bilangan bulat yang kemudian di-concatenate dipisahkan dengan tanda koma (,) sebagai hasil enkripsi.

```python
@staticmethod
def encrypt(plaintext: str, public_key: tuple[int, int]):
    n, e = public_key
    b = n.bit_length() - 1
    plaintextBin = ''.join([format(ord(p), '08b') for p in plaintext])

    # divide into b blocks
    m = [ int(plaintextBin[i:i+b].ljust(b, '0'), 2) for i in range(0, len(plaintextBin), b) ]
    c = [ fast_mod_exp(i, e, n) for i in m ]
    return ','.join(map(str, c))
```

16 Proses Enkripsi RSA

6. Melakukan Hashing SHA256 terhadap Pesan

Sebelum proses tanda tangan, pesan akan di-hash terlebih dahulu. Pertama, diinialisasi service SHA256 yang merupakan objek dengan kelas $SHA256\_custom$.

```
alice_sha_service = SHA_256_custom()
digest = alice_sha_service.hash(message)
```

17 Inisialisasi dan Pemanggilan Metode Hash SHA256

Proses ini pertama-tama akan inisialisasi array konstan k yang terdiri dari bilangan heksadesimal 32 bit sisa akar tiga dari enam puluh empat bilangan prima pertama selama 32 kali. Kode lengkap pada fungsi terlampir pada bab Fungsi-Fungsi Pembantu.

```
class SHA_256_custom:
    k = generate_rounded_values()
```

18 Proses Inisialisasi Object SHA256

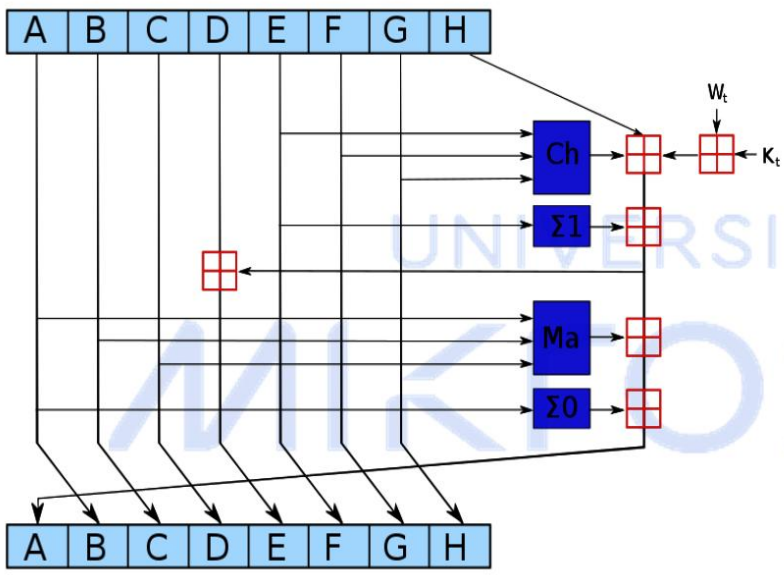
Untuk proses hashing terhadap plaintext, diinialisasi terlebih dahulu array $hashValues$ yang terdiri dari bilangan heksadesimal 32 bit sisa akar dua dari delapan bilangan prima pertama. Lalu setiap karakter pada plaintext diubah ke dalam bentuk biner berdasarkan nilai ASCII dan ditambahkan angka 1 di paling belakang. Lalu biner plaintext tersebut di-padding dengan nol sampai panjang karakter sama dengan $448\ mod\ 512$. Setelah itu, ditambahkan panjang bit plaintext asli sebesar 64 bit di belakangnya. Block berukuran 512 tersebut akan dibagi menjadi 16 block berukuran 32 bit masing-masing yang disimpan dalam variabel w, lalu berdasarkan nilai tersebut, dicari nilai block ke-17 sampai block ke-64 menggunakan rumus

$$s0 = ROR(w[j-15], 7)\ XOR\ ROR(w[j-15], 18)\ XOR\ SHR(w[j-15], 3)$$

$$s1 = ROR(w[j-2], 17)\ XOR\ ROR(w[j-2], 19)\ XOR\ SHR(w[j-2], 10)$$

$w[j] = w[j-16] + s0 + w[j-7] + s1$, dimana j merupakan iterator dari 16 sampai 63.

Setelah mendapat nilai $w[0]$-$w[63]$, inisialisasi variabel a, b, c, d, e, f, g, dan h masing-masing nilai $hashValues$ pertama hingga terakhir. Lalu, dilakukan perulangan sebanyak 64 kali untuk melakukan mutasi pada $hashValues$ untuk mendapat digest akhir menggunakan rumus berikut.



1. S_1 = (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25)
2. ch = (e and f) xor ((not e) and g);
3. temp1 = (h + S_1 + ch + k[i] + w[i]) mod 2^32
4. S_0 = (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22)
5. maj = (a and b) xor (a and c) xor (b and c)
6. temp2 := (S_0 + maj) mod 2^32
7. h = g
8. g = f
9. f = e
10. e = (d + temp1) mod 2^32
11. d = c
12. c = b
13. b = a
14. a = (temp1 + temp2) mod 2^32

19 Algoritma Pembentukan Digest pada SHA256

```python
def hash(self, message: str):
    hashValues = generate_hash_values()

    # Convert message to binary
    message_bits = ''.join(format(ord(c), '08b') for c in message)
    original_length = len(message_bits)

    # Append '1' bit
    message_bits += '1'

    # Pad with zeros until length ≡ 448 mod 512
    while (len(message_bits) + 64) % 512 != 0:
        message_bits += '0'

    # Append original length as 64-bit big-endian integer
    message_bits += format(original_length, '064b')

    # Process in 512-bit blocks
    for i in range(0, len(message_bits), 512):
        block = message_bits[i:i+512]

        # Break block into sixteen 32-bit words (w0-w15)
        w = [int(block[j:j+32], 2) for j in range(0, 512, 32)]

        # Extend to 64 words (w16-w63)
        for j in range(16, 64):
            s0 = right_rotate(w[j-15], 7) ^ right_rotate(w[j-15], 18) ^ (w[j-15] >> 3)
            s1 = right_rotate(w[j-2], 17) ^ right_rotate(w[j-2], 19) ^ (w[j-2] >> 10)
            w.append((w[j-16] + s0 + w[j-7] + s1) & 0xFFFFFFFF)

        a, b, c, d, e, f, g, h = hashValues

        for j in range(64):
            S1 = right_rotate(e, 6) ^ right_rotate(e, 11) ^ right_rotate(e, 25)
            ch = (e & f) ^ (~e & g)
            temp1 = (h + S1 + ch + self.k[j] + w[j]) & 0xFFFFFFFF
            S0 = right_rotate(a, 2) ^ right_rotate(a, 13) ^ right_rotate(a, 22)
            maj = (a & b) ^ (a & c) ^ (b & c)
            temp2 = (S0 + maj) & 0xFFFFFFFF

            h = g
            g = f
            f = e
            e = (d + temp1) & 0xFFFFFFFF
            d = c
            c = b
            b = a
```

20 Proses Hashing pada SHA256 1

```python
            a = (temp1 + temp2) & 0xFFFFFFFF

        # Add to hash values
        hashValues = [
            (hashValues[0] + a) & 0xFFFFFFFF,
            (hashValues[1] + b) & 0xFFFFFFFF,
            (hashValues[2] + c) & 0xFFFFFFFF,
            (hashValues[3] + d) & 0xFFFFFFFF,
            (hashValues[4] + e) & 0xFFFFFFFF,
            (hashValues[5] + f) & 0xFFFFFFFF,
            (hashValues[6] + g) & 0xFFFFFFFF,
            (hashValues[7] + h) & 0xFFFFFFFF,
        ]

    # Final digest
    digest = ''.join(format(hv, '08x') for hv in hashValues)
    return digest[:4]
```

21 Proses Hashing pada SHA256 2

Setelah semua selesai, setiap nilai dalam $hashValues$ akan di-concantenate dalam bentuk heksadesimal 32 bit. Untuk simulasi ini, yang dibutuhkan hanya 4 karakter pertama dari digest.

7. Digital Signing Menggunakan Schnorr

```
digital_signature = alice.sign_message(digest)
```

22 Pemanggilan Metode Tanda Tangan Digital

Setelah mendapat digest dari input pesan, maka akan dilakukan proses tanda tangan digital oleh pengirim.

```
def sign_message(self, digest: str) -> str:
    digital_signature = Schnorr.sign(digest, self.__schnorr_private_key, self.__schnorr_public_key)
    return digital_signature
```

23 Argumen untuk Metode Tanda Tangan pada Schnorr

Program akan invoke metode static $sign$ yang menerima parameter plaintext untuk ditandatangani, dalam kasus ini berupa digest dari pesan, serta private dan public key dari pengirim. Pertama, akan generate sebuah bilangan bulat acak di bawah q, dan akan menghitung nilai $r = g^k mod\ p$. Lalu akan dilakukan proses hash SHA256 dengan argumen $\{message\}|\{r\}$, mengambil 4 karakter pertama yang diubah ke integer dan disimpan dalam variabel e. Lalu dihitung nilai $s = (k + x * e)\ mod\ q$. Nilai yang dikembalikan merupakan concantenation e dan s, dipisahkan oleh koma.

```
@staticmethod
def sign(message: str, private_key: tuple[int, int], public_key: tuple[int, int, int]):
    q, x = private_key
    p, g, y = public_key
    k = randbelow(q)
    r = fast_mod_exp(g, k, p)
    sha_service = SHA_256_custom()
    e = int(sha_service.hash(f"{message}|{r}"), 16)
    s = (k + x * e) % q
    return f"{e},{s}"
```

24 Proses signing pada Schnorr

Data packet akhir merupakan concantenation dari kunci AES128 yang telah dienkripsi menggunakan algoritma RSA, pesan yang telah dienkripsi menggunakan algoritma AES128, dan tanda tangan digital meggunakan SHA256 dan Schnorr dipisahkan oleh garis (|).

```
data_packet = f"{encrypted_key}|{encrypted_message}|{digital_signature}"
```

25 Hasil paket data

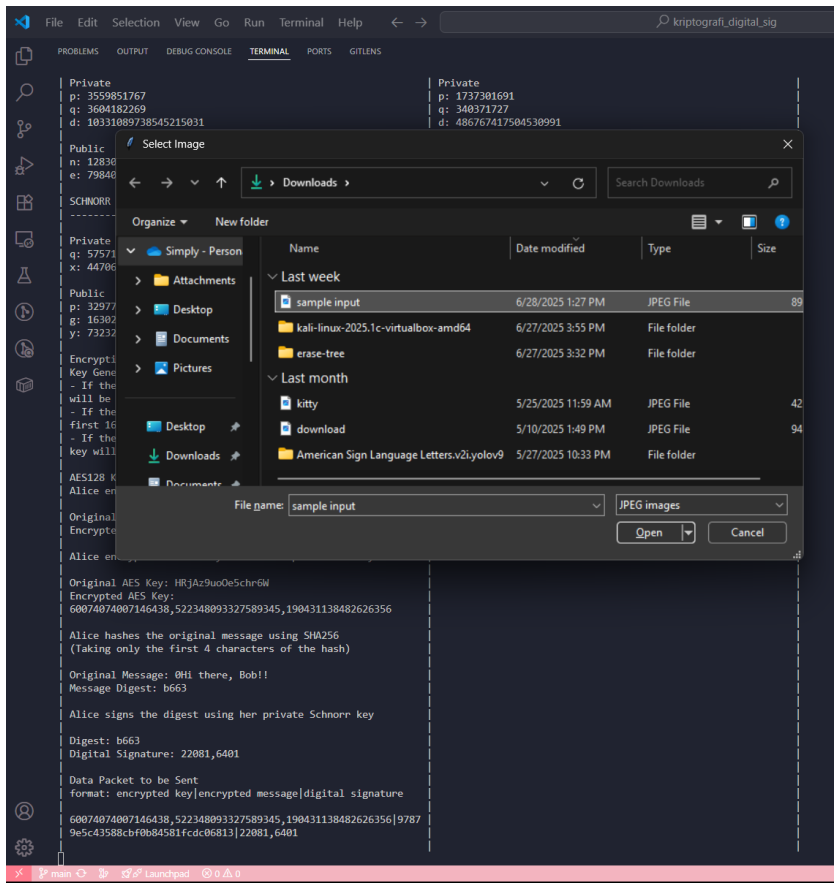
8. Penyisipan Paket Data ke dalam Gambar

Pengguna akan disuruh untuk upload sebuah gambar untuk dilakukan proses penyisipan bit pesan di dalam gambar. Citra yang diperbolehkan berupa tipe citra PNG, JPG, BMP, dan JPEG.

```
input_path = filedialog.askopenfilename(
    title="Select Image",
    filetypes=[("PNG images", "*.png"), ("BMP images", "*.bmp"), ("JPG images", "*.jpg"), ("JPEG images", "*.jpeg")]
)
print_step(["Alice Hides Data Packet inside Image (input.png)"])

if not input_path:
    print_step("Please pick an image")
    continue
```

26 Proses upload foto

Ketika program berjalan, akan muncul dialog popup seperti berikut agar pengguna bisa upload foto.



27 Dialog untuk Upload Foto

Setelah pengguna berhasil upload foto, akan menjalankan metode untuk melakukan proses penyisipan.

```
hide_data_in_image(input_path, "output.png", data_packet)
```

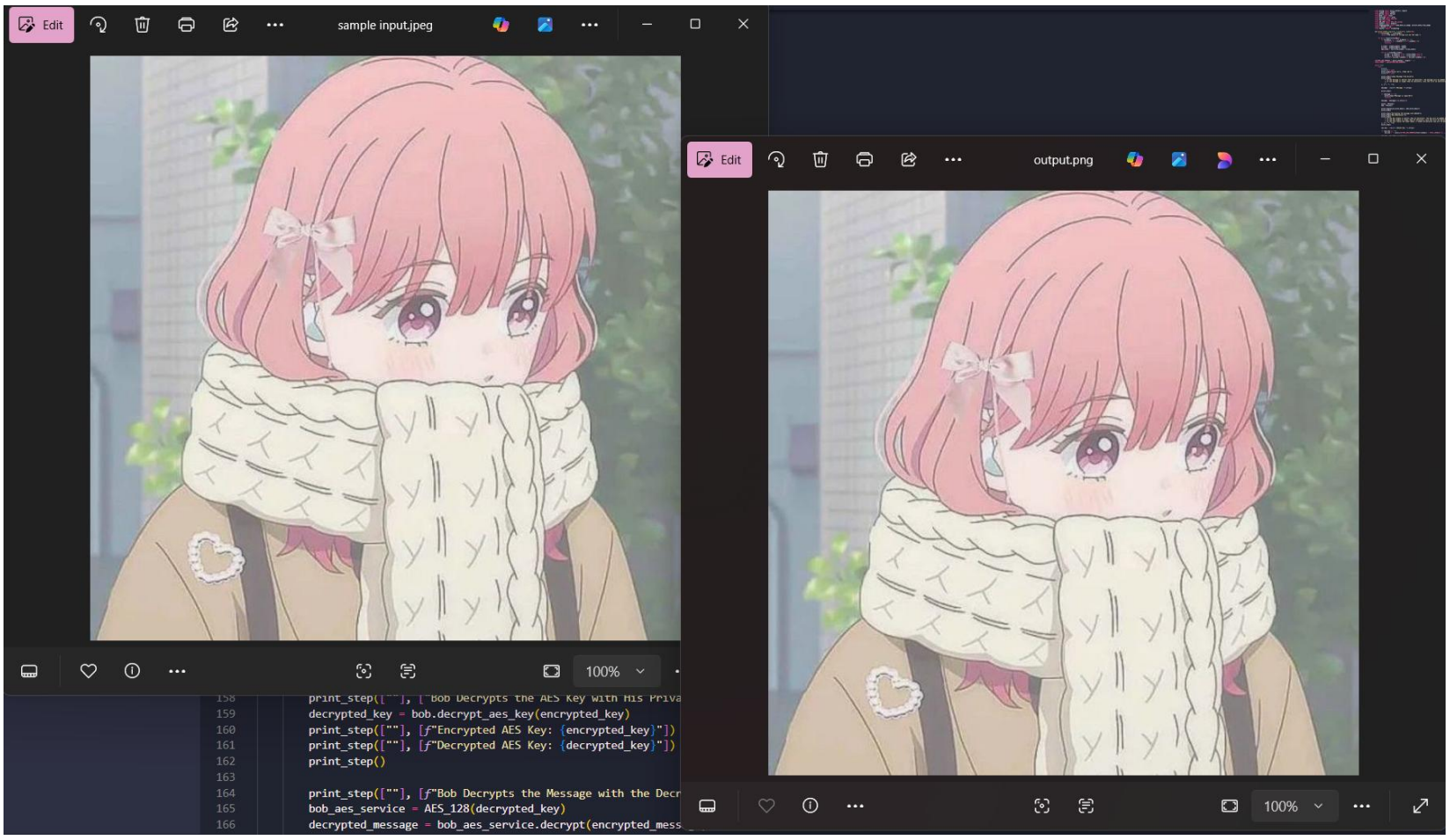
28 Pemanggilan Metode Penyisipan Packet Data dalam Foto

Proses penyisipan menggunakan algoritma steganografi sederhana. Pertama-tama, foto input akan dibaca untuk load pixel-pixel citra. Paket data dikonversi menjadi biner dan ditambah karakter penanda akhir pesan, yaitu 1111111111111110. Untuk setiap channel R, G, B pada semua pixel citra, akan disisip setiap bit pada paket data ke dalam LSB channel tersebut. Citra yang telah dimodifikasi akan disimpan pada root directory project dengan nama file ‘output.png’.

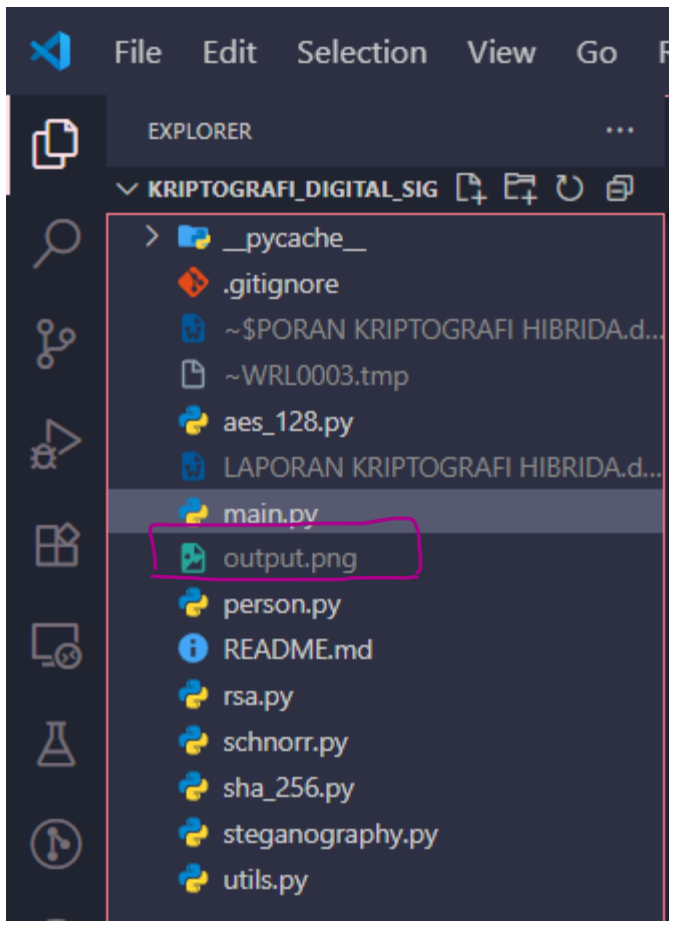
```python
def hide_data_in_image(input_path, output_path, data):
    img = Image.open(input_path)
    if img.mode != 'RGB':
        img = img.convert('RGB')
    pixels = img.load()

    binary_data = str_to_bin(data) + '1111111111111110'  # EOF marker
    data_len = len(binary_data)
    width, height = img.size

    idx = 0
    for y in range(height):
        for x in range(width):
            if idx >= data_len:
                break
            r, g, b = pixels[x, y]
            # Replace LSB of R, G, B
            r_bin = format(r, '08b')
            g_bin = format(g, '08b')
            b_bin = format(b, '08b')

            if idx < data_len:
                r_bin = r_bin[:-1] + binary_data[idx]
                idx += 1
            if idx < data_len:
                g_bin = g_bin[:-1] + binary_data[idx]
                idx += 1
            if idx < data_len:
                b_bin = b_bin[:-1] + binary_data[idx]
                idx += 1

            pixels[x, y] = (int(r_bin, 2), int(g_bin, 2), int(b_bin, 2))

        if idx >= data_len:
            break

    img.save(output_path)
```

29 Proses Penyisipan Data dalam Foto

Berikut merupakan contoh perbandingan citra input awal dan citra hasil modifikasi.



30 Perbandingan Foto Sebelum dan Sesudah Penyisipan



31 Posisi Generate output.png setelah Penyisipan

E. Proses Penerimaan Pesan

1. Ekstraksi Data dari Gambar

Setelah penerima, Bob, mendapat citra yang dimodifikasi, ia akan melakukan proses ekstraksi.

```
decoded_data = extract_data_from_image("output.png")
```

32 Pemanggilan Metode Ekstraksi Data

Pertama-tama, program akan membaca citra dan load setiap pixel-pixel di dalamnya berisi bit pada channel R, G, dan B masing-masing. Lalu akan diambil setiap nilai LSB pada citra yang disimpan dalam variabel $binary\_data$ sampai menemukan penanda akhir pesan. Setelah itu, dibagi jadi block berisi 8 bit yang diubah menjadi karakter string sesuai nilai ASCII.

```python
def extract_data_from_image(image_path):
    img = Image.open(image_path)
    pixels = img.load()
    width, height = img.size

    binary_data = ""
    for y in range(height):
        for x in range(width):
            r, g, b = pixels[x, y]
            binary_data += format(r, '08b')[-1]
            binary_data += format(g, '08b')[-1]
            binary_data += format(b, '08b')[-1]

    # Split binary by 8 bits
    all_bytes = [binary_data[i:i+8] for i in range(0, len(binary_data), 8)]
    decoded_data = ""
    for byte in all_bytes:
        if byte == '11111110':  # Our EOF marker
            break
        decoded_data += chr(int(byte, 2))
    return decoded_data
```

33 Proses Ekstraksi Data dari Foto

Data hasil ekstraksi kemudian dipisahkan berdasarkan tanda garis (|) yang terdiri dari kunci AES yang dienkripsi, pesan yang dienkripsi, dan tanda tangan digital.

```python
encrypted_key, encrypted_message, digital_signature = decoded_data.split('|')
digital_signature = digital_signature[:-1] # Remove EOF market

print_step(["", "", ""], [
    f"Encrypted Key: {encrypted_key}",
    f"Encrypted Message: {encrypted_message}",
    f"Digital Signature: {digital_signature}",
])
```

34 Hasil Ekstraksi

2. Dekripsi Kunci AES menggunakan RSA

Setelah mendapatkan paket data, akan dilakukan proses dekripsi menggunakan kunci private penerima.

```python
decrypted_key = bob.decrypt_aes_key(encrypted_key)
```

35 Pemanggilan Fungsi Dekripsi RSA

```python
def decrypt_aes_key(self, aes_encrypted_key: str) -> str:
    aes_decrypted_key = RSA.decrypt(aes_encrypted_key, self.__rsa_private_key)
    return aes_decrypted_key
```

36 Argumen Metode Dekripsi pada RSA

Program akan invoke metode static decrypt pada class RSA. Pertama, cipher text yang merupakan kumpulan angka dipisahkan oleh koma, masing-masing akan melalui operasi perhitungan $cipherText[i]^d \ mod \ n$ untuk mendapatkan nilai block sebelum dienkripsi yang kemudian dikonversi menjadi biner. Kumpulan biner tersebut di-concantenate lalu dipisahkan menjadi block berukuran 8 bit yang lalu dikonversi menjadi karakter string berdasarkan nilai ASCII. Sehingga, hasil tersebut merupakan plaintext awal. Dalam kasus ini, plaintext berupa kunci AES128 yang telah digunakan untuk enkripsi pesan dari pengirim.

```python
@staticmethod
def decrypt(cipherText: str, private_key: tuple[int, int, int]):
    p, q, d = private_key
    n = p * q
    b = n.bit_length() - 1
    cipherBlocks = list(map(int, cipherText.split(',')))
    m = [ fast_mod_exp(i, d, n) for i in cipherBlocks ]
    plaintextBin = ''.join([format(i, f"0{b}b") for i in m])
    plaintextAscii = [ int(plaintextBin[i:i+8], 2) for i in range(0, len(plaintextBin), 8) ]
    plaintext = ''.join([chr(i) for i in plaintextAscii])
    return plaintext
```

37 Proses Dekripsi RSA

3. Dekripsi Pesan menggunakan AES

Diinisialisasi dulu service AES milik bob dengan kunci AES hasil dekripsi. Kemudian, akan dimulai proses dekripsi pesan.

```python
bob_aes_service = AES_128(decrypted_key)
decrypted_message = bob_aes_service.decrypt(encrypted_message)
```

38 Inisialisasi dan Pemanggilan Fungsi Dekripsi AES128

Pertama-tama, diinisialisasi sebuah matriks 4x4 bernama $state\_array$ yang akan menampung setiap bilangan heksadesimal 8 bit dalam cipher text. Untuk ronde pertama, akan dilakukan operasi add round key dengan round key pada ronde terakhir. Lalu untuk dekripsi, akan dilakukan iterasi secara mundur dari 9 hingga 0. $state\_array$ akan melalui tahap inverse shift row, dimana setiap index i dari 1 sampai 3, i kolom terakhir pada baris i dipindahkan ke depan. Setelah itu, akan melalui proses inverse sub bytes, dimana nilai dalam $state\_array$ digantikan oleh nilainya berdasarkan Inverse S-Box. Setelah itu akan melalui operasi add round key dengan round key ronde tersebut. Kecuali ronde iterasi terakhir, akan dilakukan proses inverse mix columns, dimana $state\_array$ akan melakukan operasi perkalian Galois Field dengan matriks 4x4 berikut.

$$\begin{matrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{matrix}$$

Setiap nilai dalam matriks kemudian akan diubah menjadi karakter string berdasarkan ASCII yang lalu di-concatenate untuk menghasilkan plaintext.

```python
def decrypt(self, cipherText: str):
    state_array = [ [ 0 for _ in range(4) ] for _ in range(4) ]

    for i in range(0, 16 * 2, 2):
        row = (i // 2) % 4
        col = (i // 2) // 4
        state_array[col][row] = int(cipherText[i:i+2], 16)

    # add round key
    for i in range(4):
        for j in range(4):
            state_array[i][j] ^= self.__roundKeys[self.__Nr][i][j]

    for i in range(self.__Nr - 1, -1, -1):
        # inv shift rows
        for r in range (1, 4):
            row = [state_array[c][r] for c in range(4)]
            row = row[-r:] + row[:-r]
            for c in range(4):
                state_array[c][r] = row[c]

        # inv sub bytes
        for j in range(4):
            for k in range(4):
                state_array[j][k] = self.__inverseSBox[state_array[j][k]]

        # add round key
        for j in range(4):
            for k in range(4):
                state_array[j][k] ^= self.__roundKeys[i][j][k]

        # inv mix columns
        if (i != 0):
            for col in range(4):
                a = state_array[col]
                temp = [
                    gf_mul(0x0e, a[0]) ^ gf_mul(0x0b, a[1]) ^ gf_mul(0x0d, a[2]) ^ gf_mul(0x09, a[3]),
                    gf_mul(0x09, a[0]) ^ gf_mul(0x0e, a[1]) ^ gf_mul(0x0b, a[2]) ^ gf_mul(0x0d, a[3]),
                    gf_mul(0x0d, a[0]) ^ gf_mul(0x09, a[1]) ^ gf_mul(0x0e, a[2]) ^ gf_mul(0x0b, a[3]),
                    gf_mul(0x0b, a[0]) ^ gf_mul(0x0d, a[1]) ^ gf_mul(0x09, a[2]) ^ gf_mul(0x0e, a[3]),
                ]
                for x in range(4):
                    state_array[col][x] = temp[x]

    return ''.join(f'{chr(state_array[c][r])}' for c in range(4) for r in range(4))
```

39 Proses Dekripsi AES128

4. Verifikasi Tanda Tangan Pengirim menggunakan Schnorr

Sebelum verifikasi, penerima hash pesan hasil dekripsi menggunakan algoritma SHA256, mengambil 4 karakter pertama.

```python
bob_sha_service = SHA_256_custom()
digest = bob_sha_service.hash(decrypted_message)
```

40 Inisialisasi dan Pemanggilan Fungsi Hash SHA256

Untuk tahap terakhir, penerima akan verifikasi tanda tangan menggunakan public key pengirim untuk memastikan integritas informasi.

```python
verification = Schnorr.verify(digest, digital_signature, alice.get_schnorr_public_keys())
```

41 Pemanggilan Metode Verifikasi Tanda Tangan

Pada metode static $verify$ dalam class Schnorr. Nilai $gs = g^s mod\ p$ dan $ye = y^e mod\ p$ dilakukan terlebih dahulu menggunakan nilai e dan s dari tanda tangan, serta p pada kunci public. Setelah itu, dikalkulasi nilai $r = (gs \cdot ye) * p$. Lalu dengan SHA256, dilakukan hashing terhadap teks $\{message\}|\{r\}$ dan mengambil 4 karakter pertama, dimana message merupakan hasil digest pesan dekripsi. Setelah itu, hasil hash dikonversi menjadi integer.

```python
@staticmethod
def verify(message: str, signature: str, public_key: tuple[int, int, int])-> int:
    p, g, y = public_key
    e, s = map(int, signature.split(','))
    gs = fast_mod_exp(g, s, p)
    ye = fast_mod_exp(y, e, p)
    r = (gs * ye) % p
    sha_service = SHA_256_custom()
    ev = int(sha_service.hash(f"{message}|{r}"), 16)
    return ev
```

42 Proses Verifikasi Tanda Tangan pada Schnorr

Terakhir, dilakukan perbandingan apakah nilai *e* pada tanda tangan digital sama dengan hasil verifikasi. Jika sama, maka pesan tersebut valid. Jika tidak sama, maka pesan tersebut tidak valid.

```python
f"Thus, the message is: {'VALID' if int(digital_signature.split(',')[0]) == verification else 'NOT VALID'}"
```

43 Hasil Akhir

F. Fungsi-Fungsi Pembantu dalam Program

Beberapa code snippet yang telah ditunjukkan pada laporan, terdapat banyak fungsi yang membantu abstraksi program sehingga lebih mudah dimengerti secara keseluruhan, seperti bagian generate angka prima, fast exponentiation, dll. Berikut terlampir kode lengkap fungsi tersebut agar pembaca dapat memahami kode secara detail.

```python
from math import floor, isqrt, sqrt
from random import randrange, getrandbits
from secrets import randbelow

def rotate_left(n: int, d: int, max_bits=8):
    d %= max_bits
    return ((n << d) | (n >> (max_bits - d))) & ((1 << max_bits) - 1)

def right_rotate(n: int, d: int,max_bits=32):
    return ((n >> d) | (n << (max_bits - d))) & 0xFFFFFFFF

def fast_mod_exp(b: int, exp: int, m: int):
    res = 1
    while exp > 1:
        if exp & 1:
            res = (res * b) % m
        b = b ** 2 % m
        exp >>= 1
    return (b * res) % m

def inverse_mod(n: int, m: int):
    r0, r1 = m, n
    t0, t1 = 0, 1

    while r1 != 0:
        q = r0 // r1
        r = r0 % r1

        r0 = r1
        r1 = r

        t = t0 - t1 * q
        t0 = t1
        t1 = t

    if (t0 < 0):
        t0 += m

    return t0

def gcd(a: int, b: int):
```

```python
  if b == 0:
    return a
  return gcd(b, a % b)

def factorize(n: int):
    factors = set()
    for i in range(2, isqrt(n) + 1):
        while n % i == 0:
            factors.add(i)
            n //= i
    if n > 1:
        factors.add(n)
    return factors

def find_generator(p: int, q: int):
    for h in range(2, p):
        g = fast_mod_exp(h, (p - 1) // q, p)
        if g != 1:
            return g
    raise ValueError("No generator found")

# === PRIME NUMBER GENERATION ===
def sieve(limit: int):
    is_prime = [True] * (limit + 1)
    is_prime[0:2] = [False, False]
    for i in range(2, int(limit**0.5) + 1):
        if is_prime[i]:
            for j in range(i*i, limit+1, i):
                is_prime[j] = False
    return [i for i, val in enumerate(is_prime) if val]

def is_probable_prime(n: int, k: int = 5):  # Miller-Rabin
    if n < 2:
        return False
    for p in [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]:
        if n % p == 0:
            return n == p
    r, d = 0, n - 1
    while d % 2 == 0:
        d //= 2
        r += 1
    for _ in range(k):
        a = randrange(2, n - 1)
        x = fast_mod_exp(a, d, n)
        if x in (1, n - 1):
            continue
        for _ in range(r - 1):
            x = fast_mod_exp(x, 2, n)
            if x == n - 1:
                break
        else:
            return False
    return True

small_primes = sieve(10**7)

def is_prime(n: int):
    limit = int(n**0.5) + 1
    for p in small_primes:
        if p > limit:
            break
        if n % p == 0:
            return False
    return is_probable_prime(n)

def generate_large_primes(bit_length: int, count: int) -> list[int]:
    result = []
```

```python
    while len(result) < count:
        n = getrandbits(bit_length) | 1  # ensure odd
        if is_prime(n):
            result.append(n)
    return result

# === AES UTILS ===
def gf_mul(a: int, b: int):
    result = 0
    for _ in range(8):
        if b & 1:
            result ^= a
        high_bit_set = a & 0x80
        a <<= 1
        if high_bit_set:
            a ^= 0x11B
        a &= 0xFF
        b >>= 1
    return result

def gf_inverse(a: int):
    if a == 0:
        return 0  # Defined in AES

    r0, r1 = 0x11B, a
    t0, t1 = 0, 1

    while r1 != 0:
        q = 0
        deg_r0 = r0.bit_length() - 1
        deg_r1 = r1.bit_length() - 1
        while deg_r0 >= deg_r1:
            shift = deg_r0 - deg_r1
            q ^= (1 << shift)
            r0 ^= r1 << shift
            t0 ^= t1 << shift
            deg_r0 = r0.bit_length() - 1

        r0, r1 = r1, r0
        t0, t1 = t1, t0

    return t0 & 0xFF

def generate_aes_sbox():
    forwardSBox = [0] * 256
    inverseSBox = [0] * 256

    for i in range(256):
        inv = gf_inverse(i)
        s = inv
        for r in range(1, 5):
            s ^= rotate_left(inv, r)
        s ^= 0x63
        forwardSBox[i] = s
        inverseSBox[s] = i

    return forwardSBox, inverseSBox

# === RSA UTILS ===
def generate_valid_d(p: int, q: int, totient_n: int):
    lower = max(p, q) + 1
    upper = totient_n - 1
    while True:
        d = randbelow(upper - lower + 1) + lower  # [lower, upper]
        if gcd(d, totient_n) == 1:
            return d
```

```python
# === SHA UTILS ===
def generate_hash_values():
    first8Primes = sieve(19)
    hashValues = []

    for primeNum in first8Primes:
        squareRootOfPrimeNum = sqrt(primeNum)
        squareRootOfPrimeNum = squareRootOfPrimeNum - floor(squareRootOfPrimeNum)

        rootPrimeBinary = '0b'
        for _ in range (32):
            squareRootOfPrimeNum *= 2
            floorOfRoot = floor(squareRootOfPrimeNum)
            squareRootOfPrimeNum = squareRootOfPrimeNum - floorOfRoot

            rootPrimeBinary += str(floorOfRoot)

        hashValues.append(int(rootPrimeBinary, 2))

    return hashValues

def generate_rounded_values():
    first64Primes = sieve(311)
    roundedValues = []

    for primeNum in first64Primes:
        cubeRootOfPrimeNum = pow(primeNum, 1/3)
        cubeRootOfPrimeNum = cubeRootOfPrimeNum - floor(cubeRootOfPrimeNum)

        rootPrimeBinary = '0b'
        for _ in range (32):
            cubeRootOfPrimeNum *= 2
            floorOfRoot = floor(cubeRootOfPrimeNum)
            cubeRootOfPrimeNum = cubeRootOfPrimeNum - floorOfRoot

            rootPrimeBinary += str(floorOfRoot)

        roundedValues.append(int(rootPrimeBinary, 2))

    return roundedValues

# === SCHNORR UTILS ===
def generate_schnorr_pq(q_bits: int = 16, p_bits: int = 32, max_attempts: int = 10000):
    while True:
        q = randrange(2**(q_bits-1), 2**q_bits)
        if not is_prime(q):
            continue

        # Try to find k such that p = kq + 1 is prime
        for _ in range(max_attempts):
            k = randrange(2**(p_bits - q_bits - 1), 2**(p_bits - q_bits))
            p = k * q + 1
            if is_prime(p):
                return p, q
```