

Laporan Tugas Kecil 3
Implementasi Algoritma RSA, Elgamal dan Diffie-Hellman
IF4020 Kriptografi



I Putu Gede Wirasuta	13517015
Hilmi Naufal Yafie	13517035

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2020

1. Source Code

3.1. Algoritma RSA

Ukuran bit / key size yang digunakan sebesar 1024 bit.

Public and Private Key Generator:

```
def generate_rsa_key() -> Tuple[Tuple[int, int], Tuple[int, int]]:
    """
    return public and private key for rsa algorithm
    """
    p = number.getPrime(KEY_SIZE)
    q = number.getPrime(KEY_SIZE)
    n = p*q
    euler = (p-1)*(q-1)
    public = 65537
    private = modinv(public, euler)

    return (public, n), (private, n)
```

Algoritma Enkripsi:

```
def encrypt(plaintext: bytes, public_key: Tuple[int, int]) -> bytes:
    key = public_key[0]
    n = public_key[1]

    block_size = KEY_SIZE // 4

    plaintext = pad(plaintext, block_size)
    message_block = msg_to_hex_of_n_bytes(plaintext, block_size)

    ciphertext = bytearray()
    for block in message_block:
        msg_val = int(block.decode(), 16)
        c = pow(msg_val, key, n)
        ciphertext += c.to_bytes((c.bit_length() + 7) // 8, 'big')

    return bytes(ciphertext)
```

Algoritma Dekripsi:

```
def decrypt(ciphertext: bytes, private_key: Tuple[int, int]) -> bytes:
    key = private_key[0]
    n = private_key[1]

    block_size = KEY_SIZE // 4
```

```

message_block = msg_to_hex_of_n_bytes(ciphertext, block_size)

plaintext = bytearray()
for block in message_block:
    msg_val = int(block.decode(), 16)
    p = pow(msg_val, key, n)
    plaintext += (p.to_bytes((p.bit_length() + 7) // 8, 'big'))

plaintext = bytes(plaintext)
plaintext = unpad(plaintext, block_size)
return plaintext

```

3.2. Algoritma Elgamal

Ukuran bit / key size yang digunakan sebesar 1024 bit.

Public and Private Key Generator:

```

def generate_eg_keypair() -> Tuple[Tuple[int, int, int], Tuple[int, int]]:
    """
    returns public and private key
    """
    p = number.getPrime(KEY_SIZE)
    g = randint(1, p-1)
    x = randint(1, p-2)
    y = pow(g, x, p)

    return (y, g, p), (x, p)

```

Algoritma Enkripsi:

```

def encrypt(plaintext: bytes, public_key: Tuple[int, int, int]) -> bytes:
    """
    returns ciphertext with double the size of plaintext
    ciphertext is arranged in a_0, b_0, a_1, b_1, ..., a_n, b_n
    where a_x and b_x are the a and b component of block x respectively
    """
    y = public_key[0]
    g = public_key[1]
    p = public_key[2]
    k = randint(1, p-2)
    block_size = (KEY_SIZE-1) // 8
    plaintext = pad(plaintext, block_size)
    pt_blocks = msg_to_hex_of_n_bytes(plaintext, block_size)

```

```

ciphertext = []
for block in pt_blocks:
    m = int(block.decode(), 16)

    a = pow(g, k, p)
    b = (pow(y, k, p) * (m % p)) % p

    ciphertext.append(f'{a:0256x}'.encode())
    ciphertext.append(f'{b:0256x}'.encode())
ciphertext = hex_of_n_bytes_to_msg(ciphertext)

return ciphertext

```

Algoritma Dekripsi:

```

def decrypt(ciphertext: bytes, private_key: Tuple[int, int]) -> bytes:
    """
    returns plaintext with half the size of ciphertext
    ciphertext is arranged in a_0, b_0, a_1, b_1, ..., a_n, b_n
    where a_x and b_x are the a and b component of block x respectively
    """
    x = private_key[0]
    p = private_key[1]
    block_size = KEY_SIZE // 8
    pt_block_size = (KEY_SIZE-1) // 8
    ct_blocks = msg_to_hex_of_n_bytes(ciphertext, block_size)

    plaintext = []
    for i in range(0, len(ct_blocks), 2):
        a = int(ct_blocks[i].decode(), 16)
        b = int(ct_blocks[i+1].decode(), 16)

        a_inv = pow(a, p-1-x, p)
        m = (b * a_inv) % p

        plaintext.append(f'{m:0254x}'.encode())
    plaintext = hex_of_n_bytes_to_msg(plaintext)
    plaintext = unpad(plaintext, pt_block_size)

    return plaintext

```

3.3. Algoritma Pertukaran Kunci Diffie-Hellman

Shared Key Generator:

```

def generate_shared_parameter() -> Tuple[int,int]:
    """
    return prime number g and n where  $g < n$ 
    (g,n)
    """
    a = number.getPrime(KEY_SIZE)
    b = number.getPrime(KEY_SIZE)
    while(a==b):
        b = number.getPrime(KEY_SIZE)
    if (a>b):
        return b, a
    return a, b

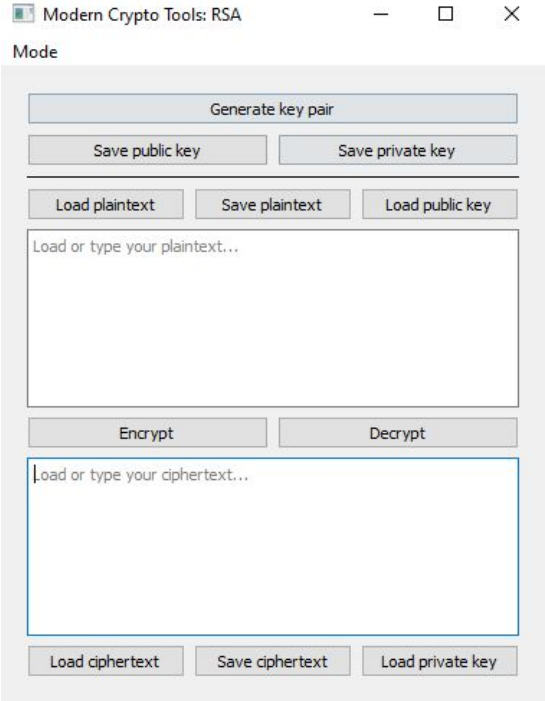
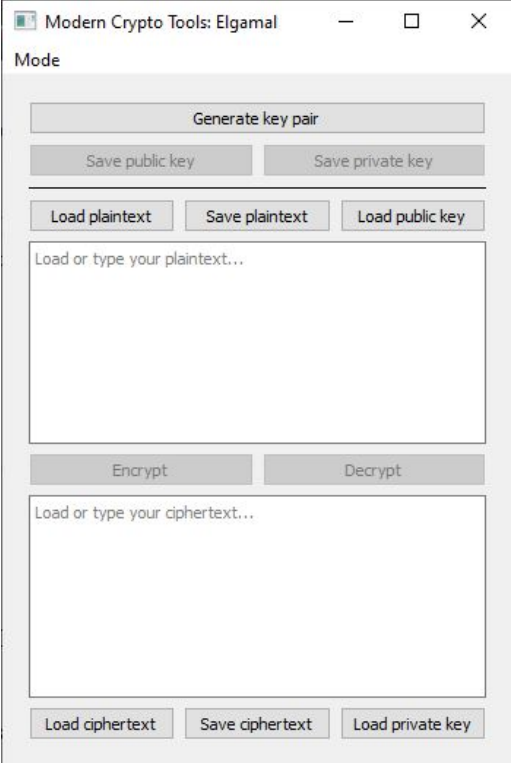
def generate_secret_pow() -> int:
    """
    return big random number
    the number should be kept as secret by the owner
    """
    return randint(1,2**KEY_SIZE-1)

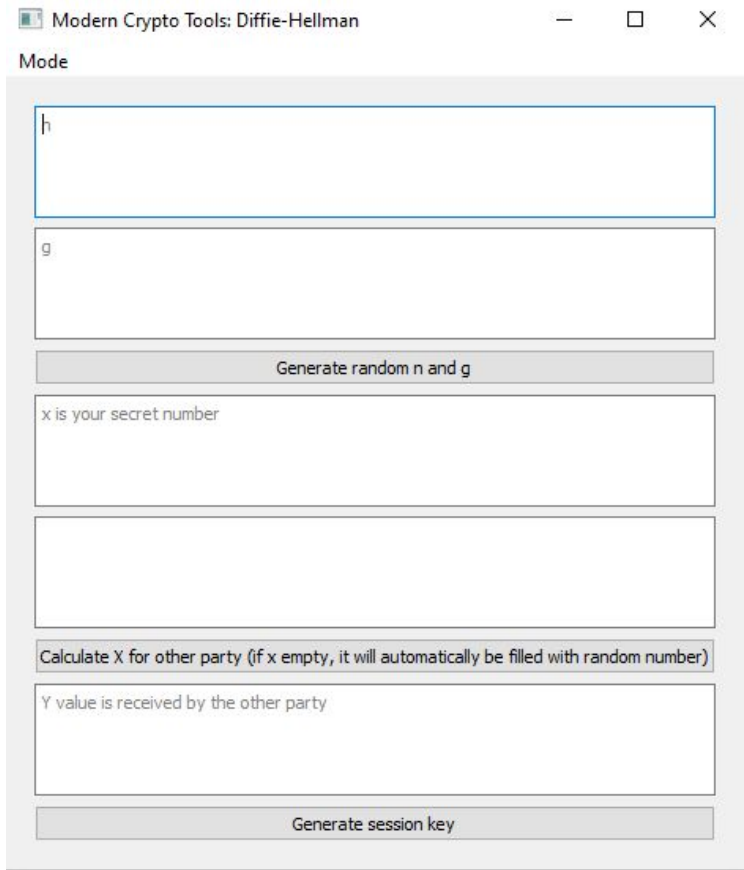
def shared_op(g:int, n:int, secret_pow:int) -> int:
    """
    return X where  $X == g^{secret\_pow} \pmod n$ 
    the value of X will be shared to the other party to generate private
    key
    """
    return pow(g, secret_pow, n)

def generate_private_key(shared_op_result:int, n:int, secret_pow:int) ->
int:
    """
    return private key where private_key ==
    other_party_op_result**secret_pow (mod n)
    the value of private key should be the same as the other party
    """
    return pow(shared_op_result, secret_pow, n)

```

2. Screenshot Program

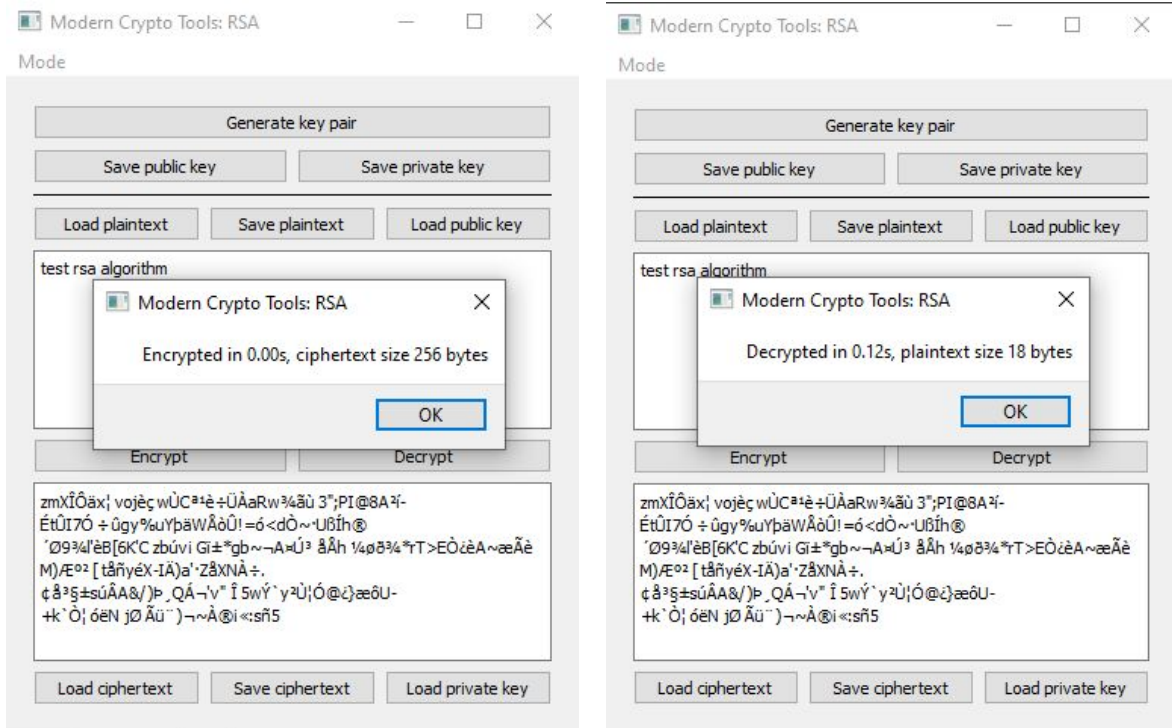
Mode	Screenshot
RSA	 <p>The screenshot shows the 'Modern Crypto Tools: RSA' application window. It features a 'Mode' section with buttons for 'Generate key pair', 'Save public key', and 'Save private key'. Below these are buttons for 'Load plaintext', 'Save plaintext', and 'Load public key'. A large text area is labeled 'Load or type your plaintext...'. At the bottom of this section are 'Encrypt' and 'Decrypt' buttons. Another large text area is labeled 'Load or type your ciphertext...'. At the very bottom are buttons for 'Load ciphertext', 'Save ciphertext', and 'Load private key'.</p>
Elgamal	 <p>The screenshot shows the 'Modern Crypto Tools: Elgamal' application window. It features a 'Mode' section with buttons for 'Generate key pair', 'Save public key', and 'Save private key'. Below these are buttons for 'Load plaintext', 'Save plaintext', and 'Load public key'. A large text area is labeled 'Load or type your plaintext...'. At the bottom of this section are 'Encrypt' and 'Decrypt' buttons. Another large text area is labeled 'Load or type your ciphertext...'. At the very bottom are buttons for 'Load ciphertext', 'Save ciphertext', and 'Load private key'.</p>

<p>Pertukaran Kunci Diffie-Hellman</p>	
--	---

3. Cara Penggunaan

3.1. RSA

Algoritma RSA membutuhkan public key dan private key. Jika public key dan private key belum dimiliki, sepasang key tersebut dapat dibuat dengan menekan tombol generate key pair. Sepasang key yang digunakan tersebut dapat disimpan ke dalam file agar nantinya dapat digunakan kembali, untuk private key disimpan dengan format file .pri dan public key dengan format file .pub. Plainteks yang akan dienkripsi dapat diinput dengan cara diketikkan pada kolom yang tersedia, atau dengan meload file yang telah dipilih. Setelah melakukan enkripsi, akan muncul dialog yang memberitahu waktu yang dibutuhkan untuk enkripsi.



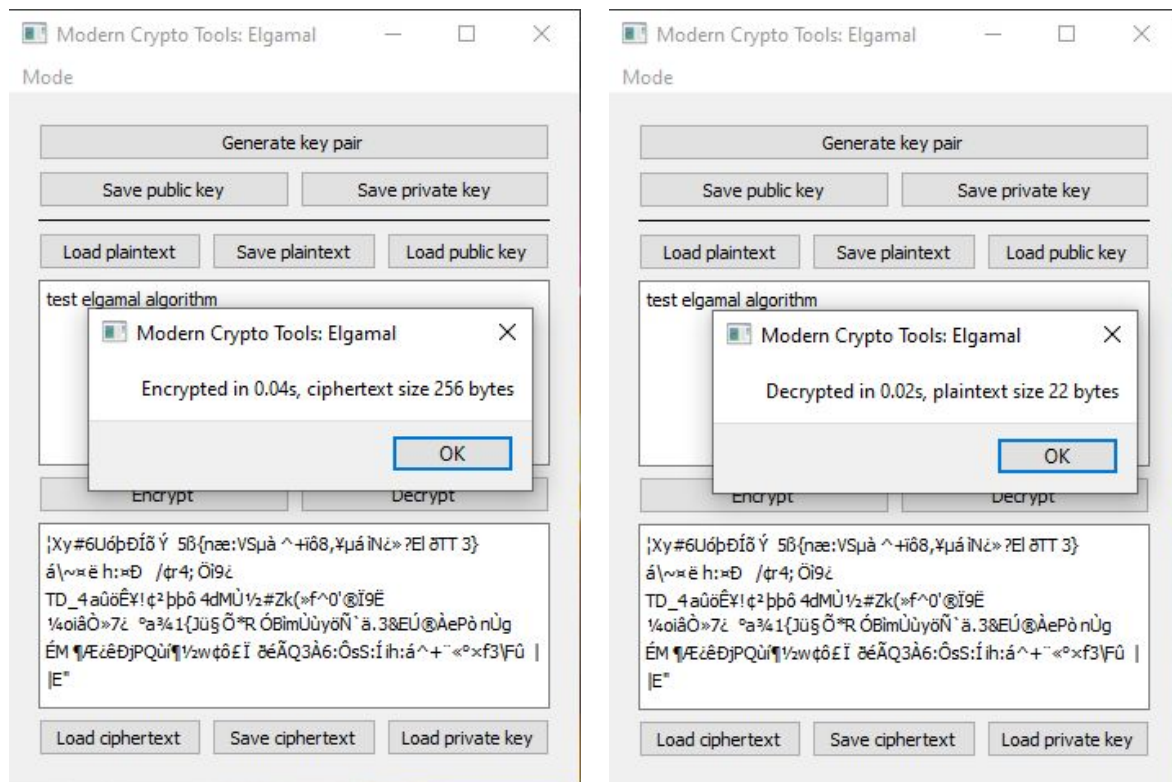
Hasil dari ciphertext juga dapat disimpan ke dalam file dengan format file .enc. Untuk melakukan dekripsi, ciphertext dapat dituliskan pada kolom chiperteks atau meload file ciphertexts dengan format file .enc. Setelah dekripsi, akan muncul dialog yang menampilkan waktu yang dibutuhkan untuk dekripsi.

Plainteks	Cipherteks
test rsa algorithm	?1ÝÛî3Ê.(ZëBäÈàlã T¶Rf@øäf sg£«D\ÐtuS:h§°÷Æ+iêc8w3ô Á*ûS¥ ²?Ä7Q îid ¬"u□E`ÜðÏè",^Ç×.¾ 3Û`ÐeÝyòa`óG[J,¢#uÊHHNJØÎêÜ1¾×ðuâd¥ÛÇÄaõ ¢ûH¾ü¹½úÐÈ ¬Í{),Í)\$drÊFunRXG-fMd5ç7B&6□ Ã¾ Êei4(u}æDÓV?`Íýª.n[C'Dq
import sys from PyQt5.QtWidgets import QApplication from gui import App if __name__ == "__main__":	ÆÆÏ½å) ^x³ûgbñ±ñãôÔW=U*n8nnLLJÉbl>Óððèp½D(ÿÁiªt /;àÛ#û;÷~Ô°i0«@p#Û°Pì=¶\α×ðCÃ@4 ukø§)i+WÈph~.;Î[jÐ*qÿ0Njr(ôpXlurvðÈèô Èoi&²HÃùâh

<pre>app = QApplication(sys.argv) main = App() sys.exit(app.exec_())</pre>	<pre>iaûJÄ91Øý««â½{z'2ûî¬b (iÖP%ÃWNÄDRoÊ²8Ö&é =g'[_çqHuÜPÛù53ÔéVaB¿âI</pre>
--	---

3.2. Elgamal

Algoritma elgamal membutuhkan sepasang public key dan private key. Jika public key dan private key belum dimiliki, sepasang key tersebut dapat dibuat dengan menekan tombol generate key pair. Sepasang key yang digunakan tersebut dapat disimpan ke dalam file agar nantinya dapat digunakan kembali. Untuk private key disimpan dengan format file .pri dan public key dengan format file .pub. Public key dan private key yang digunakan hanya dapat diinput melalui generate key pair atau load public key dan load private key. Plainteks yang akan dienkripsi dapat diinput dengan cara diketikkan pada kolom yang tersedia, atau dengan meload file yang telah dipilih. Setelah melakukan enkripsi, akan muncul dialog yang memberitahu waktu yang dibutuhkan untuk enkripsi.

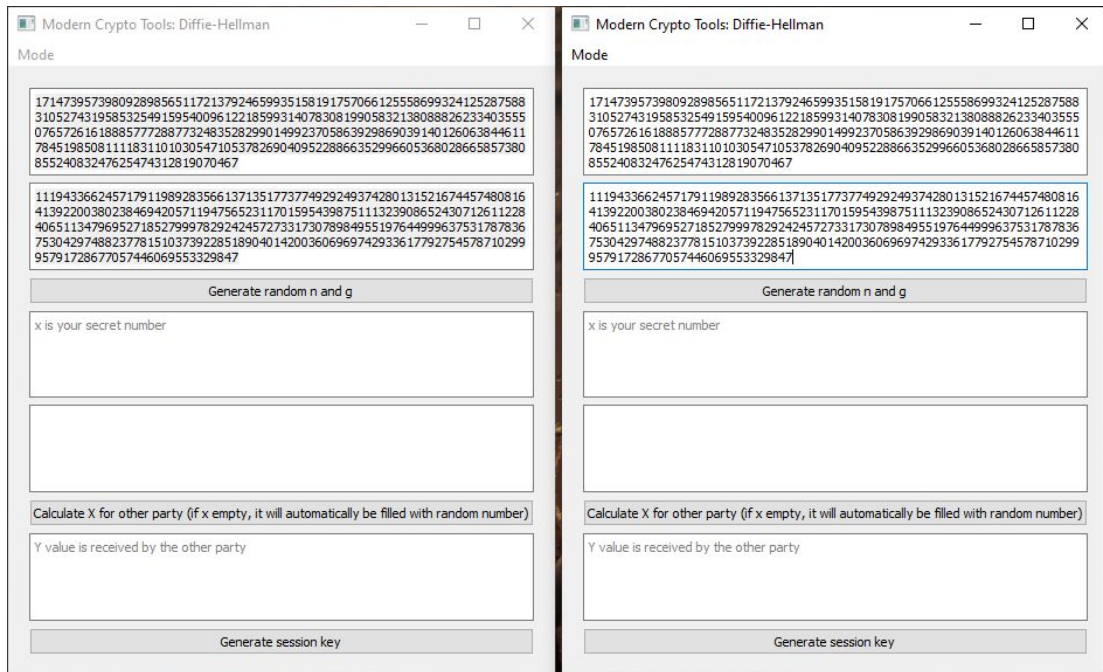


Hasil dari ciphertext juga dapat disimpan ke dalam file dengan format file .enc. Untuk melakukan dekripsi, ciphertext dapat dituliskan pada kolom chiperteks atau meload file ciphertexts dengan format file .enc. Setelah dekripsi, akan muncul dialog yang menampilkan waktu yang dibutuhkan untuk dekripsi.

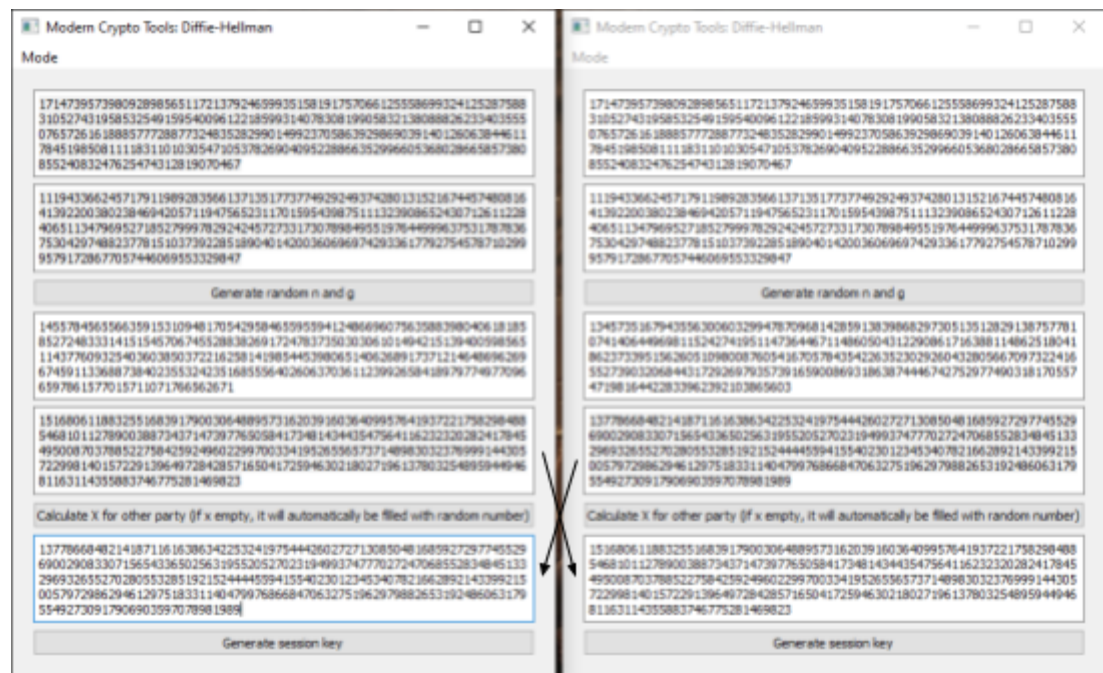
Plainteks	Cipherteks
test elgamal algorithm	<pre> ¥®êÊcñÔgã;,&,@ëbU³±á~mÔVM°JMh7ÈÂtNàÊj,\$ ßi,:¼ O»^i4Á³M²#\TñÊ eiÓ_£Ê2X*5eü-6Kûó·7*ÔQMÎY= ç,Yèp.®ð·Û `VDø'ÿ·¥XÔ[:3§B?ð>£ey,B¯ö²}{ÄuØPú¥Ç^±-&jrú n R®¥¥sÚøMßAµaghl:ÂÛ2N«oÛRýj§R½÷k]©RNUE âP¹⁄ø9<ÓròÊr_(</pre>
<pre> import sys from PyQt5.QtWidgets import QApplication from gui import App if __name__ == "__main__": app = QApplication(sys.argv) main = App() sys.exit(app.exec_()) </pre>	<pre> > -6îeGàx>½V·Ñ" Íé3ÀÂØF2p#Jé1ao[ÊâdÐ °×kÂd²ÍÐiµ¹Æð°3ò§'+ãðc S,4zÔ6ziÚ À,pöòò»jäVt¶ z,ª²h!gùÔ/èö_/ä;0 ibëZ°H½Ñ©Pe«}ã@íáÑAJ÷ë/NÛ,5wM ·ô7C£Û§¼psñuNÊfbhÑ8£hCú0ÂÝYÂYÑ2,¢*jUw3 DrGh#ÎÖo)üÜ&ð8m~éËÖð> -6îeGàx>½V·Ñ" Íé3ÀÂØF2p#Jé1ao[ÊâdÐ °×kÂd²ÍÐiµ¹Æð°3ò§'+ãðc S,4zÔ6ziÚ À,pöòò»jäVt¶ z,ª²h!gùÔ/èö_/ä;0 iÐú8Sh,Î½òmîm3æë"³a wúíîRÓdðt !L³⁄KxLW\$Ðø³o èYidV9·ÖÉ)&éÖVO¼: ®³⁄Ufq\$!² quªØ;Z²Ù»¥F·JòÖWÑ q0Êù;Ö </pre>

3.3. Pertukaran Kunci Diffie-Hellman

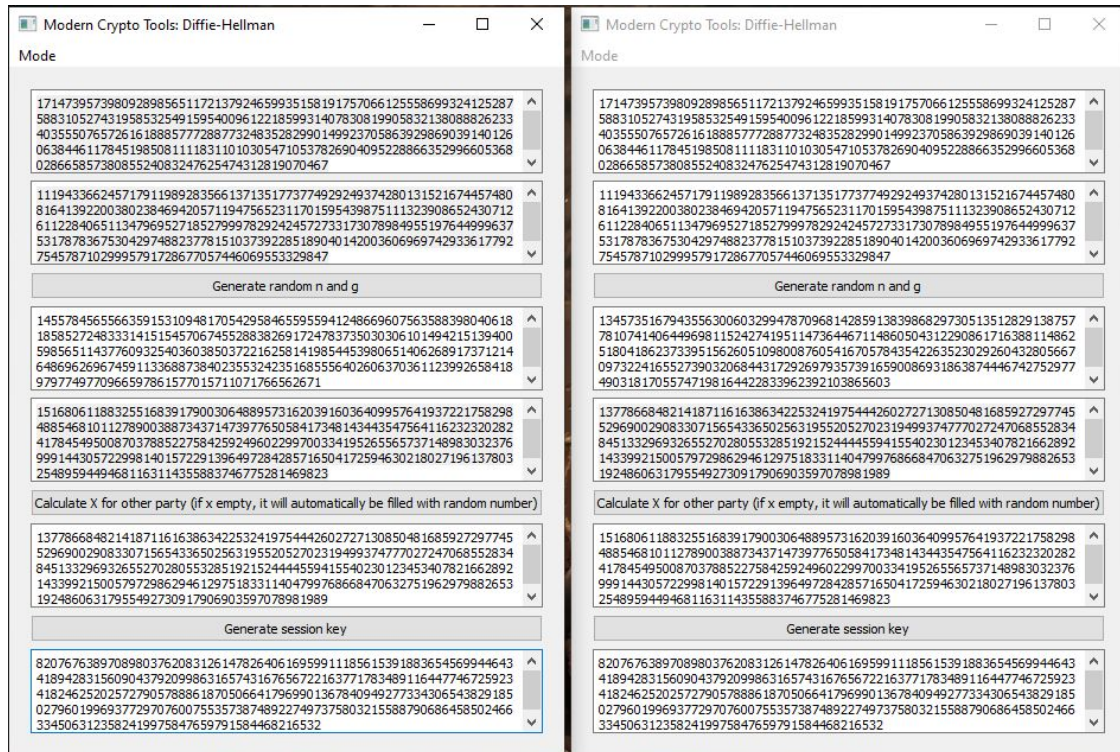
Untuk menggunakan pertukaran kunci Diffie-Hellman, diperlukan dua orang pengguna. Pertama-tama, kedua pengguna menentukan dua buah bilangan prima, n dan g , dimana $g < n$. Kedua bilangan tersebut dapat diinput oleh pengguna atau digenerate secara random.



Lalu, masing-masing pengguna memilih nilai x yang menjadi secret number dari pengguna, dan melakukan kalkulasi menggunakan nilai n , g dan x , dimana hasil kalkulasi itu akan diberikan kepada pasangan pengguna. Pada aplikasi, nilai x dapat dipilih secara random dengan mengosongkan kolom x ketika melakukan kalkulasi.



Setelah kedua pengguna sudah mendapatkan hasil kalkulasi dari pengguna lainnya, maka key dapat dibuat dengan menekan generate session key. Hasil dari session key ini akan sama untuk kedua pengguna, dimana key bisa digunakan pada algoritma kriptografi dengan kunci simetrik.



Komponen	Nilai
n	16014772759624243468632430887541462863549692023330653024906 02086073813989786682941143622989817134784793630758212407162 19344060939960016926665242636845258953243568584356476596035 18952387589005114672764709712868334213292531995571731393918 03407360366640055531474217929902559009662954747671289997063 74016835273397
g	10320265273954043015339459388209909785107015479561351896206 03297890533042609035321517943438616656191467096335195349177 21476000848988540633288388689300605858335531423383821518357 77829715911097161115278993155022228091185914916934245715439 92876843269034707986670232767545945240923985723966726282657 36147738506877
x	16941463696878385256650439467048387553030957357692378246256

	40364447842848144744270942359132526280947866610223039337130 35162158377383328375077930705435314617423182012908814693486 29256496757754753544285854039022185714045276944700414900994 59509572370211240253969488124223789864225973595729235646591 30334130097644
y	15542990617243389705419720874768273343441214173527817430801 32031753088808519376891769505586829977955274776329232327130 60277579006464627737351562071091286888884632927318760606061 41847901051222711163625345973558918370329657084441367291079 66228411412590690778399955401956541414957279885625135774910 2727264454532
X	13259283824459989791578876378505059774265468874566621255068 85405284612213290833374559402751724736729978301339653162638 14435602628374796248383997286179643685270985776093891085556 68812580947234394646321642525403465505844048170149478041500 22270126300876742327849266354790898507751800901491538931563 6586396063597
Y	20758505468712915378579977848964356917555757267678379933660 45549218742210209939739955031855940828618864805654885532611 03273837894992722576276637916156961815655247368560456338447 17697233068247602760380323822848528114947832797343935167260 73723200163290462113655656665909736088170526285111729831573 8380774494563
Session Key	64625305026110919431414653089430312505056179109628690642122 68444898900837325358424117679972629822639062251558998250901 90128593791507970634792794455796002848873715761032477980846 90000372776545976261537939911514272028614898239863774984098 65150445384010821603948555433215019884019935509692927878754 4544350453235