



Ege Üniversitesi
Mühendislik
Fakültesi
Elektrik-Elektronik Mühendisliği
Bölümü

Computer Networks 2 Final Project Report

Grup Üyeleri

İsim/Soyisim:

Barışcan İlter

Ersan Ergin

Grup Üyeleri No:

05200000485

05210000667

Rapor Teslim Tarihi:

04.07.2025

1) Introduction:

Problem Statement:

Insecure email communication is one of the biggest problems in today's interconnected digital world. Sensitive information transmitted via email is vulnerable to tampering, unauthorized access, or modification. This creates serious security vulnerabilities, especially in terms of the confidentiality and integrity of personal, financial, or corporate content. Traditional email systems lack strong encryption and authentication mechanisms to protect message content. Therefore, it is not possible to guarantee that a message sent can only be read by the intended recipient and that its content arrives unchanged as it was sent.

2) Approach:

In this project, we propose the implementation of a secure email communication system between two peers based on the PGP (Pretty Good Privacy) protocol. The primary goal is to simulate PGP's hybrid cryptosystem, which combines both symmetric and asymmetric encryption, along with digital signatures for authentication and data integrity. The following steps will be followed in the implementation of the project (This project will implement in Python and we will use "pycryptodome" library for RSA, MD5, IDEA algorithms and "zlib" library for compressing - decompressing):

Hashing and Digital Signature:

The message is summarized with the help of the MD5 algorithm. MD5 (Message Digest 5) is a cryptographic hash algorithm that converts data of any length into a fixed-length hash value. This algorithm usually produces an output of 128 bits. After the summary, a digital signature is created with the help of the sender's private RSA key. RSA (Rivest-Shamir-Adleman) is an asymmetric encryption algorithm. In this algorithm, 2048-bit private and public key pairs are created. Private keys are used to decrypt encrypted messages or to create digital signatures. Public keys are used to encrypt messages.

Compressing:

The digitally signed message is compressed with the help of the "compress" function belonging to the "zlib" library of the Python.

Encryption:

After the digitally signed message is compressed, it is encrypted using the IDEA algorithm with a randomly generated 128-bit symmetric key. IDEA (International Data Encryption Algorithm) is a symmetric key block encryption algorithm. It typically uses a 128-bit block size and takes 64 bits, i.e. 64-bit data, as input. It also produces 64-bit encrypted data as output. IDEA performs 8.5 identical encryption and decryption rounds using six different subkeys. It uses four keys for output transformation.

Encoding:

The encrypted key and the encrypted message are combined. It is converted to ASCII format with Base64 encoding and made ready to be sent to the recipient. In Base64 encoding, data is first separated into 8-bit bytes. These bytes are divided into groups of 6 bits. Each group of 6 bits is interpreted as a number between 0-63. These numbers are converted to letters, numbers or symbols according to a special Base64 character table.

Decoding:

On the receiver side, the encrypted message is decrypted with Base64 decoding. In Base64 decoding, Base64 code is read character by character. Each character is converted to the corresponding number between 0-63 in the Base64 table. Each 4 Base64 characters create 24 bits of binary data. These 24 bits of data is split into 3 bytes again.

Decryption:

The symmetric key is obtained with the recipient's private RSA key and the message is decrypted. The receiver receives a message encrypted with the IDEA algorithm and a symmetric key encrypted with the RSA algorithm. The receiver decrypts the symmetric key using his private key. The decrypted symmetric key is also used in the IDEA algorithm and the encrypted message is decrypted with the help of this algorithm.

Decompressing:

The message and the digital signature are separated from each other with the "decompress" function of the "zlib" library of the Python.

Verification:

With the help of MD5 algorithm, verification is done by comparing the hash (digest) from the message with the hash (digest) from the digital signature. The receiver generates a new MD5 digest from the content of the incoming message. The incoming digital signature is decrypted with the sender's public RSA key. As a result of this process, the MD5 digest created by the sender while sending the message is obtained. Then these two digest values are compared. If the digests are the same, the message is verified. In other words, the message has not been changed, and it really belongs to the sender.

3) Work Performed:

We first started by writing plaintext message and generating public and private RSA key pairs for the sender and receiver. Figure 2 shows the codes that perform these operations.

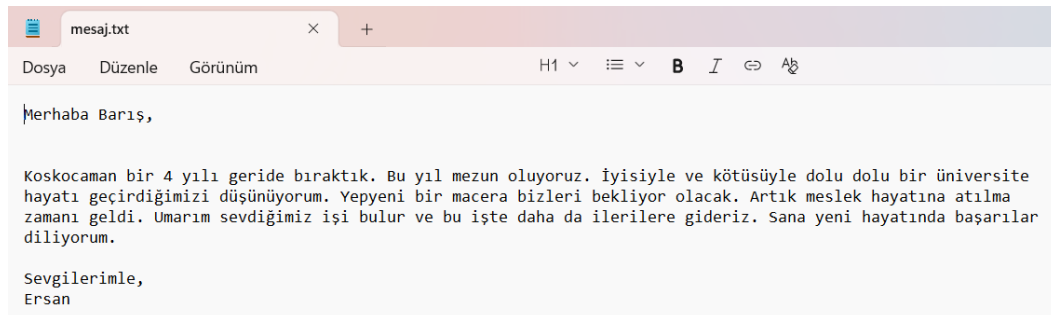


Figure 1: The plaintext message

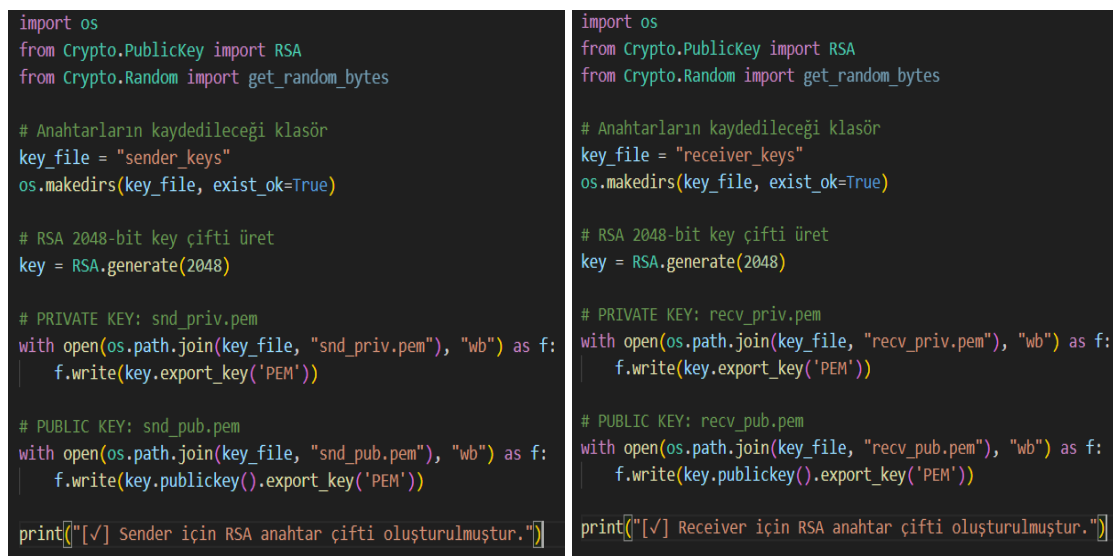
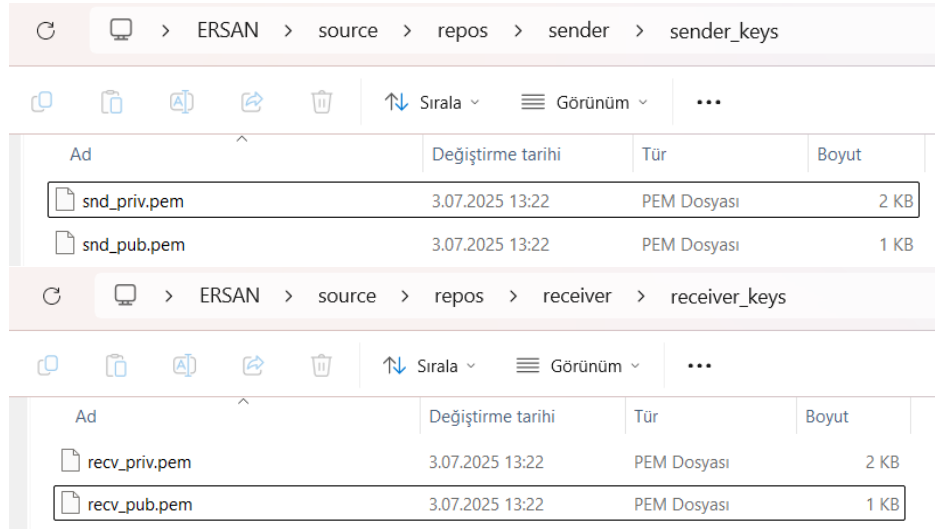


Figure 2: Sender and receiver public - private RSA key pairs generating code

The output files obtained as a result of running these codes are given in Figure 3.



Ad	Değiştirme tarihi	Tür	Boyut
snd_priv.pem	3.07.2025 13:22	PEM Dosyası	2 KB
snd_pub.pem	3.07.2025 13:22	PEM Dosyası	1 KB

Ad	Değiştirme tarihi	Tür	Boyut
recv_priv.pem	3.07.2025 13:22	PEM Dosyası	2 KB
recv_pub.pem	3.07.2025 13:22	PEM Dosyası	1 KB

Figure 3: The output files

Then we started writing the sender side code. In the first stage, we created a class for the implementation of the IDEA algorithm.

```
import socket
import os
import base64
import json
import zlib
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from Crypto.Hash import MD5
from Crypto.Signature import pkcs1_15
from Crypto.Random import get_random_bytes

# IDEA implementasyonu (basit bir blok şifreleme benzetimi)
class IDEA:
    def __init__(self, key):
        """IDEA için 128-bit anahtar (16 byte)"""
        if len(key) != 16:
            raise ValueError("IDEA anahtar 16 byte olmalı")
        self.key = key

    def encrypt(self, data):
        """Veriyi şifrele - basit XOR tabanlı benzetim"""
        encrypted = bytearray()
        key_len = len(self.key)
        for i, byte in enumerate(data):
            encrypted.append(byte ^ self.key[i % key_len])
        return bytes(encrypted)

    def decrypt(self, data):
        """Veriyi çöz - XOR işlemi simetrik"""
        return self.encrypt(data)
```

Figure 4: The IDEA class for sender side

This class simulates a simple XOR encryption algorithm instead of the actual IDEA encryption algorithm. It works with a 128-bit (16 byte) key. The encrypt and decrypt operations work the same because XOR is symmetric.

Then, the files containing the public-private key pairs for the sender, the plaintext message, the public key file of the receiver to be received from the server, and the encrypted, signed packet files to be sent to the receiver were loaded. In addition, the necessary settings were made to establish a TCP connection with the receiver. Finally, key loading functions were created.

```

# === Dosya yolları ===
SND_PRIV_KEY = "sender_keys/snd_priv.pem"
SND_PUB_KEY = "sender_keys/snd_pub.pem"
RECV_PUB_KEY = "sender_keys/recv_pub.pem"
MESSAGE_FILE = "sender_keys/mesaj.txt"
PACK_FILE = "sender_keys/encrypted_package.json"

# === TCP ayarlamaları ===
host = "127.0.0.1"
port = 10000

# === Anahtar yükleme fonksiyonları ===
def load_private_key(path):
    with open(path, "rb") as f:
        return RSA.import_key(f.read())

def load_public_key(path):
    with open(path, "rb") as f:
        return RSA.import_key(f.read())

```

Figure 5: Loading files, settings for TCP connection and creating key loading functions for sender

Next, we created the function where the main part of the PGP process will take place.

```

# === PGP Şifreleme ve İmzalama ===
def prepare_pgp_package():
    snd_priv_key = load_private_key(SND_PRIV_KEY)
    recv_pub_key = load_public_key(RECV_PUB_KEY)

    with open(MESSAGE_FILE, "r", encoding="utf-8") as f:
        plaintext_message = f.read()

    # MD5 hash hesapla
    md5_hash = MD5.new()
    md5_hash.update(plaintext_message.encode('utf-8'))
    message_digest = md5_hash.hexdigest()

    # Hash'i gönderenin private key'i ile imzala
    signer = pkcs1_15.new(snd_priv_key)
    signature = signer.sign(MD5.new(plaintext_message.encode('utf-8'))

    # Mesaj + İmza birleştir
    combined_data = plaintext_message + "\n---SIGNATURE---\n" + base64.b64encode(signature).decode()

    # zlib ile sıkıştır
    compressed_data = zlib.compress(combined_data.encode('utf-8'))

    # IDEA için 128-bit (16 byte) rastgele anahtar
    idea_key = get_random_bytes(16)

    # IDEA ile şifrele
    idea_cipher = IDEA(idea_key)
    encrypted_data = idea_cipher.encrypt(compressed_data)

    # IDEA anahtarını alıcının public key'i ile RSA şifrele
    rsa_cipher = PKCS1_OAEP.new(recv_pub_key)
    encrypted_key = rsa_cipher.encrypt(idea_key)

    # Şifrelenmiş anahtar + şifrelenmiş veri birleştir
    final_data = encrypted_key + encrypted_data

```

```

# Base64 encode
ascii_message = base64.b64encode(final_data).decode()

# Paket bilgilerini kaydet
package = {
    "encrypted_key_size": len(encrypted_key),
    "encrypted_data_size": len(encrypted_data),
    "ascii_message": ascii_message,
    "original_size": len(plaintext_message),
    "compressed_size": len(compressed_data),
    "md5_hash": message_digest
}

with open(PACK_FILE, "w") as f:
    json.dump(package, f, indent=2)

print("[✓] PGP paketi hazırlandı!")
print(f"Final ASCII mesaj uzunluğu: {len(ascii_message)} karakter")

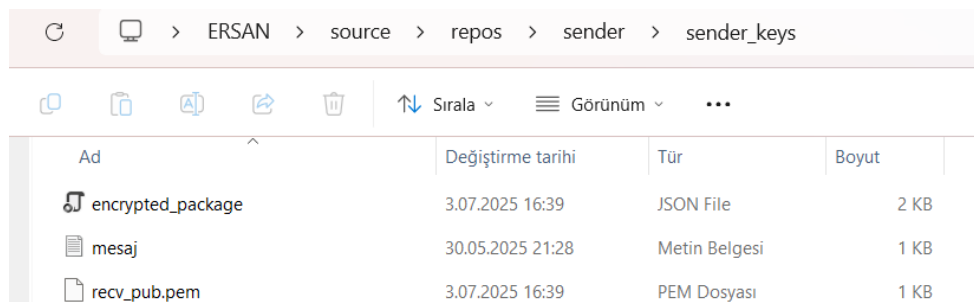
```

Figure 6: The main function of the PGP process

In this function:

- The plaintext message is read.
- MD5 hash is calculated.
- Hash is signed with sender's private key.
- Message + signature are combined.
- Compressed with "compress" function which belongs to zlib library of Python.
- Encrypted symmetrically with IDEA algorithm.
- IDEA key is encrypted with RSA with receiver's public key.
- Encrypted IDEA key + encrypted data are combined.
- Converted to ASCII with Base64 and a JSON packet is created.

The output of this function and the resulting files for sender is in Figure 7.



Ad	Değiştirme tarihi	Tür	Boyut
encrypted_package	3.07.2025 16:39	JSON File	2 KB
mesaj	30.05.2025 21:28	Metin Belgesi	1 KB
recv_pub.pem	3.07.2025 16:39	PEM Dosyası	1 KB

```
[✓] PGP paketi hazırlandı!  
Final ASCII mesaj uzunluğu: 1140 karakter
```

Figure 7: The output of the function and the resulting files for sender

Finally, we created a function to communicate with the receiver over TCP and our main function that will execute the sender side code.

```
# === TCP sunucu başlatılıyor ===  
def start_sender_server():  
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:  
        server.bind((host, port))  
        server.listen(1)  
        print(f"[ 🟡 ] PGP Sender dinleniyor: {host}:{port}")  
        print("Receiver'ın bağlanması bekleniyor...\n")  
  
        conn, addr = server.accept()  
        with conn:  
            print(f"[ 🟢 ] Receiver bağlandı: {addr}")  
  
            print("[ 📁 ] Sender'ın public key'i gönderiliyor...")  
            # Step 1: Sender'ın public key'ini gönder  
            with open(SND_PUB_KEY, "rb") as f:  
                conn.sendall(f.read() + b"<END_KEY>")  
  
            print("[ 📁 ] Receiver'ın public key'i alınıyor...")  
            # Step 2: rcv_pub.pem al  
            rcv_key_data = b""  
            while not rcv_key_data.endswith(b"<END_KEY>"):  
                rcv_key_data += conn.recv(1024)  
            rcv_key_data = rcv_key_data.replace(b"<END_KEY>", b"")  
  
            with open(RECV_PUB_KEY, "wb") as f:  
                f.write(rcv_key_data)  
            print("[✓] rcv_pub.pem alındı")  
  
            # Step 3: PGP paketini hazırla  
            prepare_gpg_package()  
  
            print("[ 📁 ] Şifrelenmiş paket gönderiliyor...")  
            # Step 4: encrypted_package.json gönder  
            with open(PACK_FILE, "rb") as f:  
                conn.sendall(f.read() + b"<END_JSON>")  
            print("[✓] encrypted_package.json gönderildi")
```

Figure 8: The TCP connection function for sender side

In this function:

- The TCP server is started on local host (127.0.0.1) and port 10000.
- The sender public key (snd_pub.pem) is sent.
- The receiver public key is received and saved as rcv_pub.pem.
- The message is prepared by calling prepare_gpg_package() function.
- The encrypted package (encrypted_package.json) is sent.

```
# === Çalıştır ===
if __name__ == "__main__":
    print("=== PGP SENDER ===")

    # Mesaj dosyasının varlığını kontrol et
    if not os.path.exists(MESSAGE_FILE):
        print(f"❌ Hata: {MESSAGE_FILE} dosyası bulunamadı!")
        exit(1)

    start_sender_server()
```

Figure 9: The main function for sender

The output of sender side code is in Figure 10.

```
[👤] PGP Sender dinleniyor: 127.0.0.1:10000
Receiver'ın bağlanması bekleniyor...

[🟢] Receiver bağlandı: ('127.0.0.1', 60624)
[📁] Sender'ın public key'i gönderiliyor...
[📁] Receiver'ın public key'i alınıyor...
[✓] recv_pub.pem alındı
[✓] PGP paketi hazırlandı!
    Final ASCII mesaj uzunluğu: 1140 karakter
[📁] Şifrelenmiş paket gönderiliyor...
[✓] encrypted_package.json gönderildi

🎉 PGP mesajı başarıyla gönderildi!
```

Figure 10: The output of sender side code

In the next stage, we moved on to writing the codes for the receiver side. First, we created a class for the implementation of the IDEA algorithm, as in the sender side.

```
import socket
import json
import os
import base64
import zlib
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from Crypto.Hash import MD5
from Crypto.Signature import pkcs1_15

# IDEA implementasyonu (sender ile aynı)
class IDEA:
    def __init__(self, key):
        """IDEA için 128-bit anahtar (16 byte)"""
        if len(key) != 16:
            raise ValueError("IDEA anahtarı 16 byte olmalı")
        self.key = key

    def encrypt(self, data):
        """Veriyi şifrele - basit XOR tabanlı benzetim"""
        encrypted = bytearray()
        key_len = len(self.key)
        for i, byte in enumerate(data):
            encrypted.append(byte ^ self.key[i % key_len])
        return bytes(encrypted)

    def decrypt(self, data):
        """Veriyi çöz - XOR işlemi simetrik"""
        return self.encrypt(data)
```

Figure 11: The IDEA class for receiver side

The operations performed here are the same as those performed by the sender.

Then, the files containing the public-private key pairs for the receiver, the output of decrypted message, the public key file of the sender to be received from the server, received encrypted message packet were loaded. In addition, the necessary settings were made to establish a TCP connection with the sender. Finally, key loading functions were created.

```
# === Dosya yolları ===
PRIV_KEY_FILE = "receiver_keys/recv_priv.pem"
PUB_KEY_FILE = "receiver_keys/recv_pub.pem"
SND_PUB_KEY_FILE = "receiver_keys/snd_pub.pem"
PACK_FILE = "receiver_keys/encrypted_package.json"
DECRYPT_OUT = "receiver_keys/dekripte_mesaj.txt"

# === TCP ayarlamaları ===
host = "127.0.0.1"
port = 10000

# === Anahtarları yükle ===
def load_private_key(path):
    with open(path, "rb") as f:
        return RSA.import_key(f.read())

def load_public_key(path):
    with open(path, "rb") as f:
        return RSA.import_key(f.read())
```

Figure 12: Loading files, settings for TCP connection and creating key loading functions for receiver

Next, we create the PGP decryption function.

```
# === PGP Çözümleme İşlemi ===
def decrypt_pgp_message():
    with open(PACK_FILE, "r") as f:
        package_data = json.load(f)

    recv_priv_key = load_private_key(PRIV_KEY_FILE)
    snd_pub_key = load_public_key(SND_PUB_KEY_FILE)

    # Base64 decode
    encrypted_combined = base64.b64decode(package_data["ascii_message"])

    # Şifrelenmiş anahtar ve veriyi ayır
    encrypted_key_size = package_data["encrypted_key_size"]
    encrypted_key = encrypted_combined[:encrypted_key_size]
    encrypted_data = encrypted_combined[encrypted_key_size:]

    # RSA ile IDEA anahtarını çöz
    rsa_cipher = PKCS1_OAEP.new(recv_priv_key)
    idea_key = rsa_cipher.decrypt(encrypted_key)

    # IDEA ile veriyi çöz
    idea_cipher = IDEA(idea_key)
    compressed_data = idea_cipher.decrypt(encrypted_data)

    # zlib ile aç
    decompressed_data = zlib.decompress(compressed_data).decode('utf-8')

    # Mesaj ve imzayı ayır
    parts = decompressed_data.split("\n---SIGNATURE---\n")
    if len(parts) != 2:
        raise ValueError("Mesaj formatı hatalı!")

    original_message = parts[0]
    signature_b64 = parts[1]
    signature = base64.b64decode(signature_b64)
```

```

# Mesajın MD5 hash'ini hesapla
md5_hash = MD5.new()
md5_hash.update(original_message.encode('utf-8'))
calculated_hash = md5_hash.hexdigest()
stored_hash = package_data["md5_hash"]
if calculated_hash == stored_hash:
    print("    ✓ MD5 hash doğrulandı!")
else:
    print("    ✗ MD5 hash uyuşmuyor!")

# İmzayı doğrula
try:
    verifier = pkcs1_15.new(snd_pub_key)
    verifier.verify(MD5.new(original_message.encode('utf-8')), signature)
    print("    ✓ Dijital imza geçerli!")
    signature_valid = True
except (ValueError, TypeError):
    print("    ✗ Dijital imza geçersiz!")
    signature_valid = False

# Çözülmüş mesajı kaydet
with open(DECRYPT_OUT, "w", encoding="utf-8") as f:
    f.write(original_message)

print("\n" + "="*60)
print("🔓 PGP MESAJI BAŞARIYLA ÇÖZÜLDÜ")
print("="*60)
print("\n📄 Mesaj İçeriği:")
print("-" * 60)
print(original_message)
print("-" * 60)

return signature_valid and (calculated_hash == stored_hash)

```

Figure 13: The PGP decryption function

In this function:

- The decoding is done with Base64.
- The encrypted IDEA key and the data are separated.
- The IDEA key is decrypted with the receiver's private key and RSA.
- The data is decrypted with IDEA.
- The data is opened with “decompress” function which belongs to zlib library of Python.
- The message and signature are separated.
- The MD5 hash is checked.
- The digital signature is verified with the sender's public key.
- The message is written to the file and displayed to the user.

Finally, we created a function to communicate with the sender over TCP and our main function that will execute the receiver side code.

```
# === TCP üzerinden sender'a bağlan ===
def connect_to_sender():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((host, port))
        print(f"[🔌] Sender'a bağlandı: {host}:{port}")

        print("[📡] Sender'ın public key'i alınıyor...")
        # Step 1: snd_pub.pem al
        sender_key_data = b""
        while not sender_key_data.endswith(b"<END_KEY>"):
            sender_key_data += s.recv(1024)
        sender_key_data = sender_key_data.replace(b"<END_KEY>", b"")

        with open(SND_PUB_KEY_FILE, "wb") as f:
            f.write(sender_key_data)
        print("[✓] snd_pub.pem alındı")

        print("[📡] Receiver'ın public key'i gönderiliyor...")
        # Step 2: rcv_pub.pem gönder
        with open(PUB_KEY_FILE, "rb") as f:
            s.sendall(f.read() + b"<END_KEY>")
        print("[✓] rcv_pub.pem gönderildi")

        print("[📡] Şifrelenmiş paket alınıyor...")
        # Step 3: encrypted_package.json al
        package_data = b""
        while not package_data.endswith(b"<END_JSON>"):
            package_data += s.recv(1024)
        package_data = package_data.replace(b"<END_JSON>", b"")

        with open(PACK_FILE, "wb") as f:
            f.write(package_data)
        print("[✓] encrypted_package.json alındı")

        print("\n" + "="*50)
        print("🔍 PGP ÇÖZÜMLEME BAŞLATILIYOR")
        print("="*50)

    # Step 4: PGP mesajını çöz
    success = decrypt_pgp_message()

    if success:
        print("\n🎉 Tüm güvenlik kontrolleri başarılı!")
    else:
        print("\n⚠️ Güvenlik kontrolleri başarısız! Mesaj tehlikeye girmiş olabilir.")
```

Figure 13: The TCP connection function for receiver side

In this function:

- Connect to sender via local host (127.0.0.1) and port 10000.
- Sender public key is received.
- Receiver public key is sent.
- Encrypted package is received (encrypted_package.json).
- decrypt_pgp_message() function is called.

↶

🖨

>

ERSAN

>

source

>

repos

>

receiver

>

receiver_keys

📁

📁

📄

🔗

🗑

↕

Sırala

▼

☰

Görünüm

▼

⋮

Ad	Değiştirme tarihi	Tür	Boyut
<div>📄</div> dekript_mesaj	3.07.2025 16:39	Metin Belgesi	1 KB
<div>📄</div> encrypted_package	3.07.2025 16:39	JSON File	2 KB
<div>📄</div> snd_pub.pem	3.07.2025 16:39	PEM Dosyası	1 KB

Figure 14: The resulting files for receiver side

```
# === Çalıştır ===
if __name__ == "__main__":
    print("=== PGP RECEIVER ===")
    connect_to_sender()
```

Figure 15: The main function for receiver side

The output of receiver side code is in Figure 16.

```
=== PGP RECEIVER ===
[🔗] Sender'a bağlandı: 127.0.0.1:10000
[📡] Sender'ın public key'i alınıyor...
[✓] snd_pub.pem alındı
[📡] Receiver'ın public key'i gönderiliyor...
[✓] recv_pub.pem gönderildi
[📡] Şifrelenmiş paket alınıyor...
[✓] encrypted_package.json alındı

=====
🔍 PGP ÇÖZÜMLEME BAŞLATILYOR
=====
✓ MD5 hash doğrulandı!
✓ Dijital imza geçerli!

=====
🔓 PGP MESAJI BAŞARIYLA ÇÖZÜLDÜ
=====

📧 Mesaj İçeriği:
-----
Merhaba Barış,

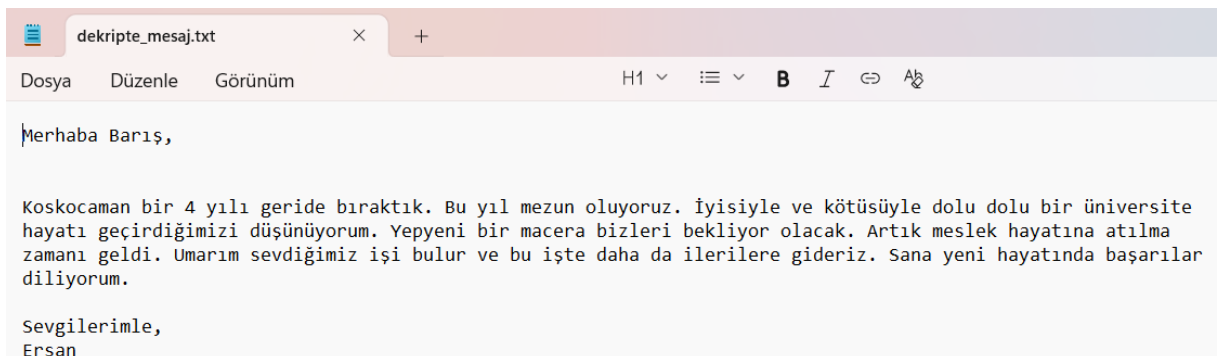
Koskocaman bir 4 yılı geride bıraktık. Bu yıl mezun oluyoruz. İyisiyle ve kötüsüyle dolu dolu bir üniversite hayatı geçirdiğimizi düşünüyorum. Yepyeni bir macera bizleri bekliyor olacak. Artık meslek hayatına atılma zamanı geldi. Umarım sevdiğimiz işi bulur ve bu işte daha da ilerilere gideriz. Sana yeni hayatında başarılar diliyorum.

Sevgilerimle,
Ersan
-----

🎉 Tüm güvenlik kontrolleri başarılı!
```

Figure 16: The output of receiver side code

Finally, the content of the decrypted message is in Figure 17.



The screenshot shows a text editor window titled 'dekripted_mesaj.txt'. The text content is as follows:

```
Merhaba Barış,

Koskocaman bir 4 yılı geride bıraktık. Bu yıl mezun oluyoruz. İyisiyle ve kötüsüyle dolu dolu bir üniversite hayatı geçirdiğimizi düşünüyorum. Yepyeni bir macera bizleri bekliyor olacak. Artık meslek hayatına atılma zamanı geldi. Umarım sevdiğimiz işi bulur ve bu işte daha da ilerilere gideriz. Sana yeni hayatında başarılar diliyorum.

Sevgilerimle,
Ersan
```

Figure 17: The content of the decrypted message

3) Results:

The secure email communication system based on the PGP protocol was successfully implemented. Key results include:

Sender Side:

- RSA key pairs were generated.
- The message was hashed using MD5 and signed with the sender's private key.
- The signed message was compressed and encrypted using a symmetric key (IDEA).
- The symmetric key was encrypted with the receiver's public RSA key.
- The final package was encoded in Base64 and sent over TCP.

Receiver Side:

- The package was decoded and separated.
- The symmetric key was decrypted using RSA, and the message was decrypted with IDEA.
- The message and signature were decompressed and verified.
- Message integrity and sender authenticity were confirmed.

All operations were completed without errors, and the message was successfully received and verified by the recipient.

4) Discussion:

This project demonstrated a simplified yet functional simulation of the PGP protocol. It showed how hybrid cryptography ensures:

- Confidentiality via symmetric encryption (IDEA),
- Authentication and integrity through RSA and MD5,
- Secure transmission using TCP sockets.

The system proved effective for secure peer-to-peer communication. For future improvements, a complete implementation of the IDEA algorithm and a user-friendly interface could be added to enhance realism and usability.

5) References:

- Andrew S. Tanenbaum, David J. Wetherall, *Computer Networks*, 5th Ed., Section 8.8.1: Pretty Good Privacy (PGP).
- <https://pypi.org/project/pycryptodome/>
- https://tedboy.github.io/python_stdlib/generated/zlib.html
- <https://www.techtarget.com/searchsecurity/definition/International-Data-Encryption-Algorithm>
- https://www.tutorialspoint.com/cryptography/md5_algorithm.htm

6) Task Distribution

In this project, tasks were effectively distributed among team members in accordance with individual competencies and project requirements. The work carried out within the scope of the project and our contributions to these studies are summarized as follows:

Ersan Ergin => writing sender - receiver side codes and helping to prepare the report

Barişcan İlter => preparation of the report and writing sender – receiver key generating codes

Addition: Our codes are in : <https://github.com/ersanergin48/Computer-Networks-2-Final-Project>