GELİŞMİŞ GÜVENLİ DOSYA TRANSFER SİSTEMİ: ŞİFRELEME, DÜŞÜK SEVİYELİ IP İŞLEME VE AĞ PERFORMANSI ANALİZİ

1. GİRİŞ

Günümüzde veri güvenliği, özellikle ağlar üzerinden dosya transferi sırasında büyük önem taşımaktadır. Bu proje, dosyaların güvenli ve bütünlüğü sağlanarak iletilmesini amaçlamaktadır. Projede ayrıca IP başlıklarının manuel düzenlenmesi ve ağ performansının ölçülmesi hedeflenmektedir.

2. TEKNİK DETAYLAR

2.1 Gerçekleştirilen Çalışmalar

- **İstemci-Sunucu İletişimi:** TCP protokolü kullanılarak iki taraf arasında güvenli iletişim sağlanmıştır.
- **Dosya Şifreleme:** İstemci tarafında AES algoritması (Fernet yapısı) kullanılarak dosya şifrelenmiştir.
- **Manuel Paketleme:** Şifreli veri 1024 baytlık parçalara bölünmüş, her pakete bir numara ve checksum değeri eklenmiştir.
- **Checksum Kontrolü:** Alıcı taraf, gelen verinin bütünlüğünü kontrol etmek için SHA-256 checksum değerini doğrulamaktadır.
- **Ağ Trafiği Analizi:** Wireshark kullanılarak gönderim sırasındaki trafik kaydedilmiş, şifreli veri transferinin başarıyla gerçekleştiği gözlemlenmiştir.

2.2 Kullanılan Teknolojiler ve Kullanım Sebepleri

Python Programlama Dili

- Neden kullandık?

Python, socket programlama, şifreleme işlemleri ve ağ işlemleri için çok güçlü ve hızlı geliştirme imkânı sağlar. Ayrıca çok sayıda hazır kütüphaneye sahip olduğu için proje sürecinde zaman tasarrufu sağlamıştır.

socket Kütüphanesi

- Neden kullandık?

Ağ üzerinden istemci ve sunucu arasında veri alışverişi yapmak için TCP bağlantıları kurmamız gerekti. Python'un socket kütüphanesi düşük seviyeli TCP/IP iletişimi kurmak için en uygun ve standart kütüphanedir.

cryptography Kütüphanesi (Fernet)

- Neden kullandık?

Gönderilecek dosyaların gizliliğini korumak için AES tabanlı simetrik şifreleme kullandık. Cryptography kütüphanesi, güvenli ve güncel şifreleme algoritmalarını Python'da kullanmayı kolaylaştırdığı için tercih edilmiştir.

hashlib Kütüphanesi

- Neden kullandık?

Gönderilen her verinin doğruluğunu ve bütünlüğünü sağlamak için paketlere SHA-256 algoritması ile checksum (özet bilgi) ekledik. Böylece veri bozulmaları veya paket kayıpları tespit edilebilmektedir.

struct Kütüphanesi

- Neden kullandık?

Verileri (paket numarası ve checksum) doğru bir biçimde paketlemek ve çözerken hatasız ayrıştırmak için kullandık. Ağ üzerinde sabit byte dizileri oluşturmak için struct modülü gereklidir.

time Kütüphanesi

- Neden kullandık?

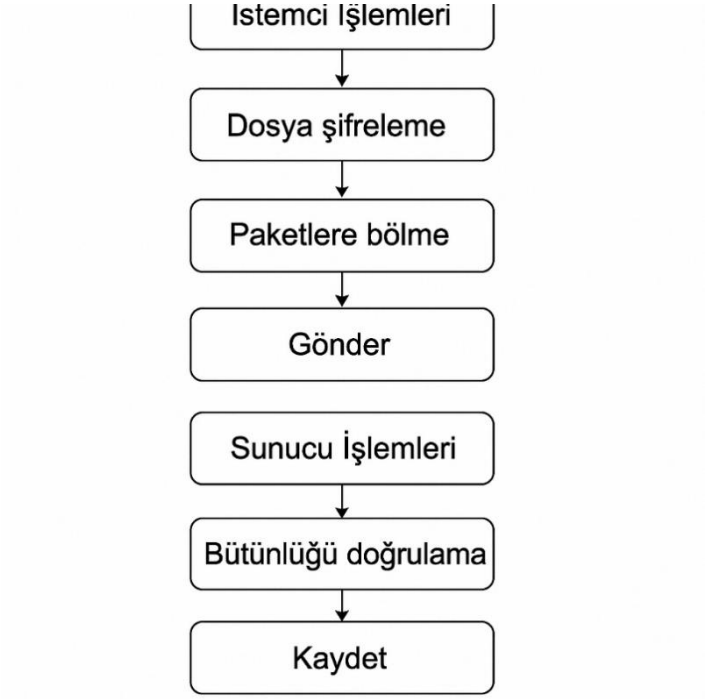
Paket gönderimleri arasında yapay ağ gecikmesi simülasyonu yapmak istedik. time.sleep() fonksiyonunu kullanarak her gönderim arasında küçük beklemeler ekleyerek gerçek dünya koşullarını daha iyi yansıttık.

Wireshark

- Neden kullandık?

Dosya transferi sırasında gönderilen TCP paketlerinin analizini yapmak için Wireshark kullanılmıştır. Paket içeriklerinin şifrelenmiş olması ve doğru aktarılması bu araçla doğrulanmıştır. Ayrıca, ilerleyen aşamada saldırı simülasyonları ve performans ölçümleri de Wireshark ile yapılacaktır.

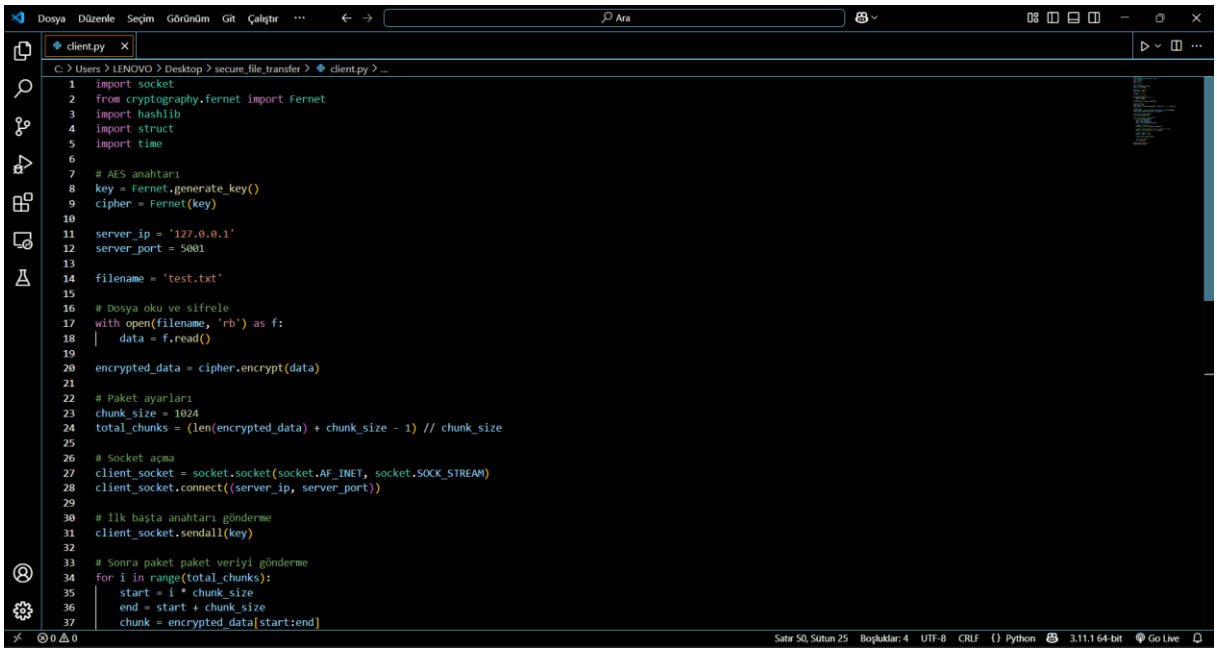
2.3 Proje Akış Diyagramı



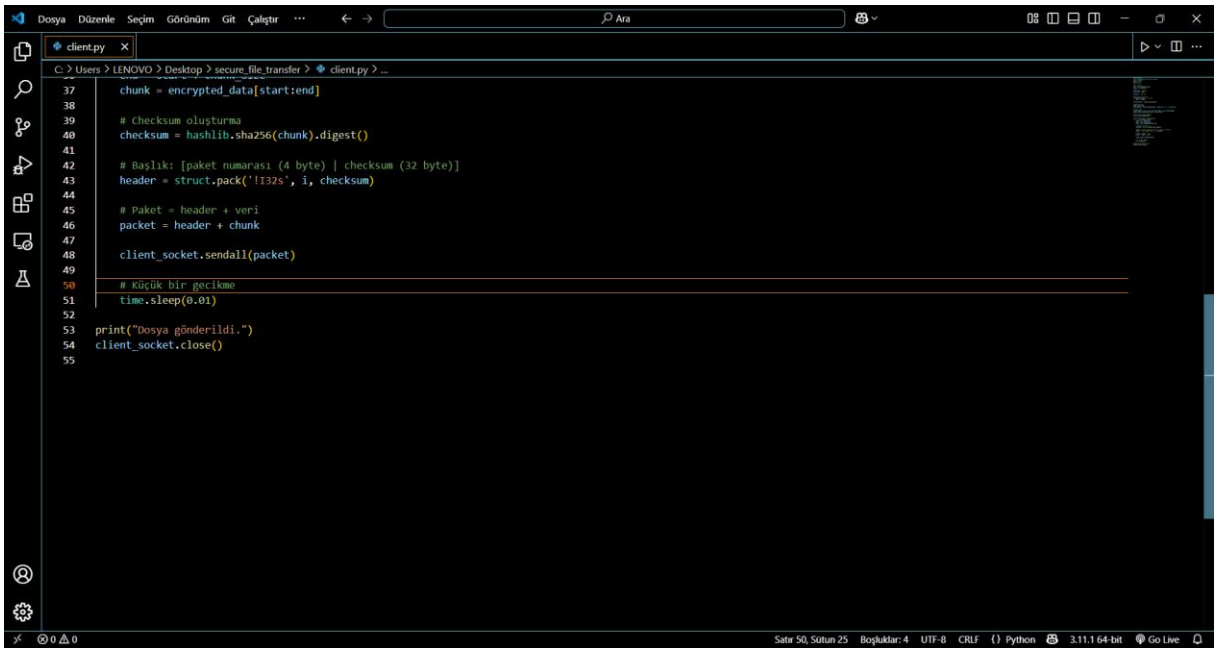
Şekil 1. Gelişmiş güvenli dosya transfer sisteminin istemci ve sunucu tarafındaki işlem akışı.

2.4 Kodların Ekran Görüntüleri ve Açıklamaları

2.4.1 İstemci (Client) Kodları



```python
import socket
from cryptography.fernet import Fernet
import hashlib
import struct
import time

# AES anahtarı
key = Fernet.generate_key()
cipher = Fernet(key)

server_ip = '127.0.0.1'
server_port = 5001

filename = 'test.txt'

# Dosya oku ve şifrele
with open(filename, 'rb') as f:
    data = f.read()

encrypted_data = cipher.encrypt(data)

# Paket ayarları
chunk_size = 1024
total_chunks = (len(encrypted_data) + chunk_size - 1) // chunk_size

# Socket açma
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((server_ip, server_port))

# İlk başta anahtarı gönderme
client_socket.sendall(key)

# Sonra paket paket veriyi gönderme
for i in range(total_chunks):
    start = i * chunk_size
    end = start + chunk_size
    chunk = encrypted_data[start:end]
```

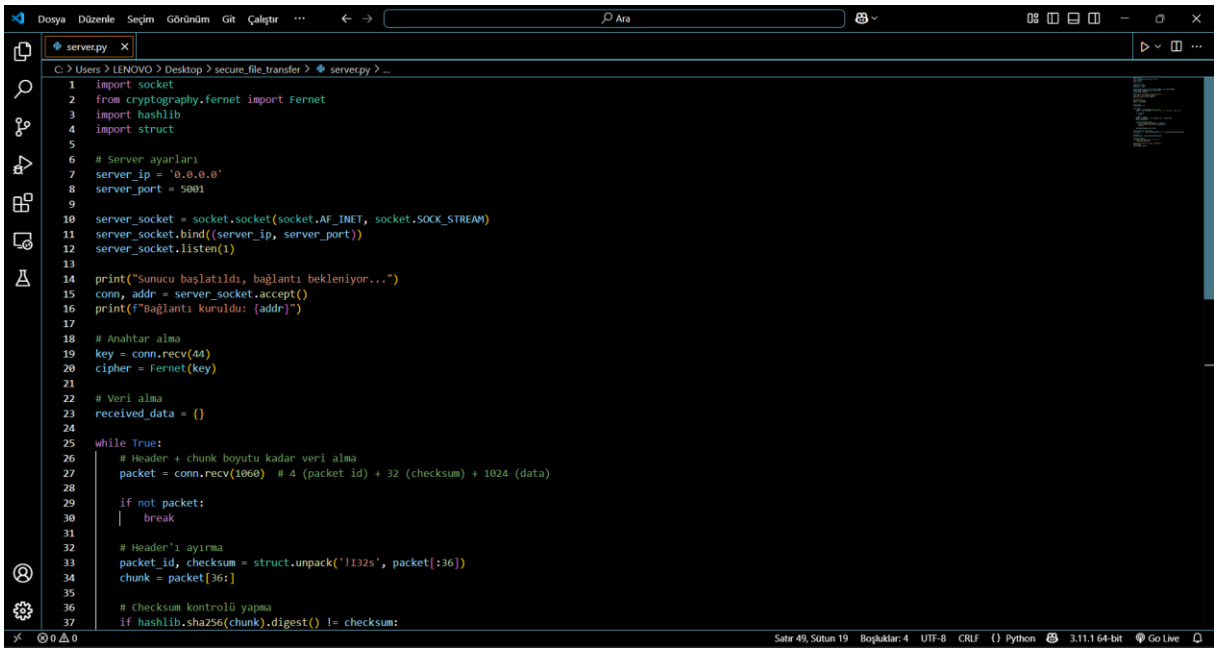


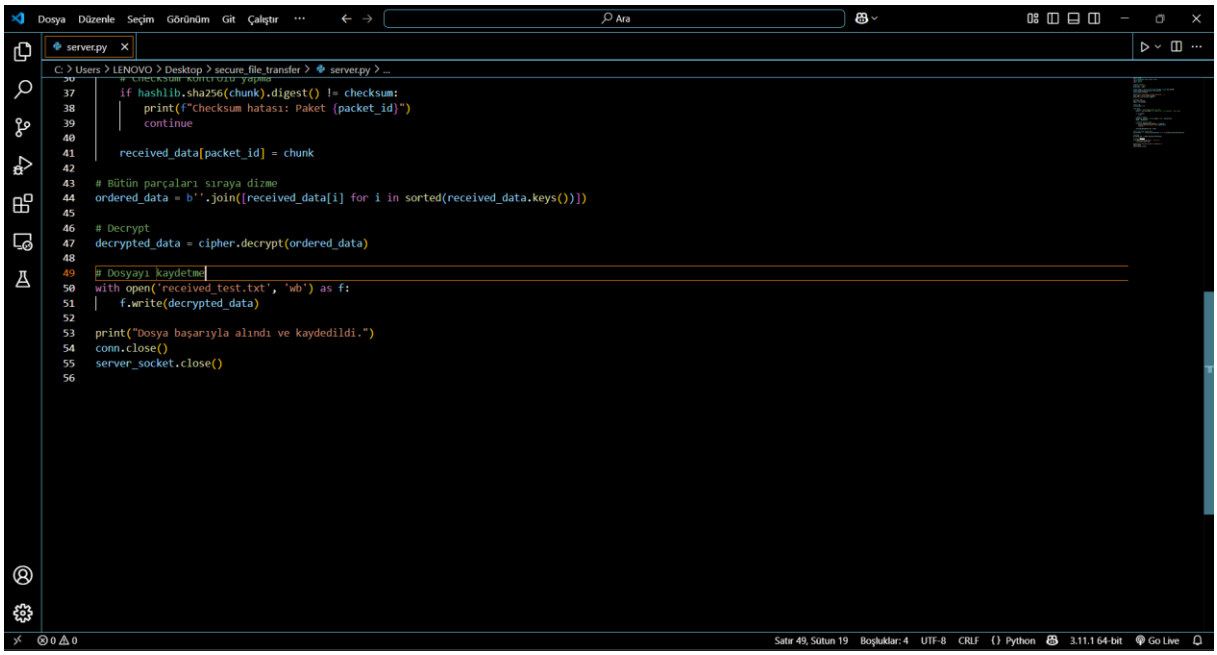
```python
    chunk = encrypted_data[start:end]

    # Checksum oluşturma
    checksum = hashlib.sha256(chunk).digest()

    # Başlık: [paket numarası (4 byte) | checksum (32 byte)]
    header = struct.pack('!I32s', i, checksum)

    # Paket = header + veri
    packet = header + chunk

    client_socket.sendall(packet)

    # Küçük bir gecikme
    time.sleep(0.01)

print("Dosya gönderildi.")
client_socket.close()
```

Açıklama:

Bu kod, istemci tarafında şifrelenmiş dosyayı TCP üzerinden sunucuya göndermek amacıyla yazılmıştır. Dosya AES algoritması ile şifrelenmiş, ardından 1024 baytlık parçalara bölünmüş ve her pakete bir numara ile checksum bilgisi eklenmiştir.

2.4.2 Sunucu (Server) Kodları



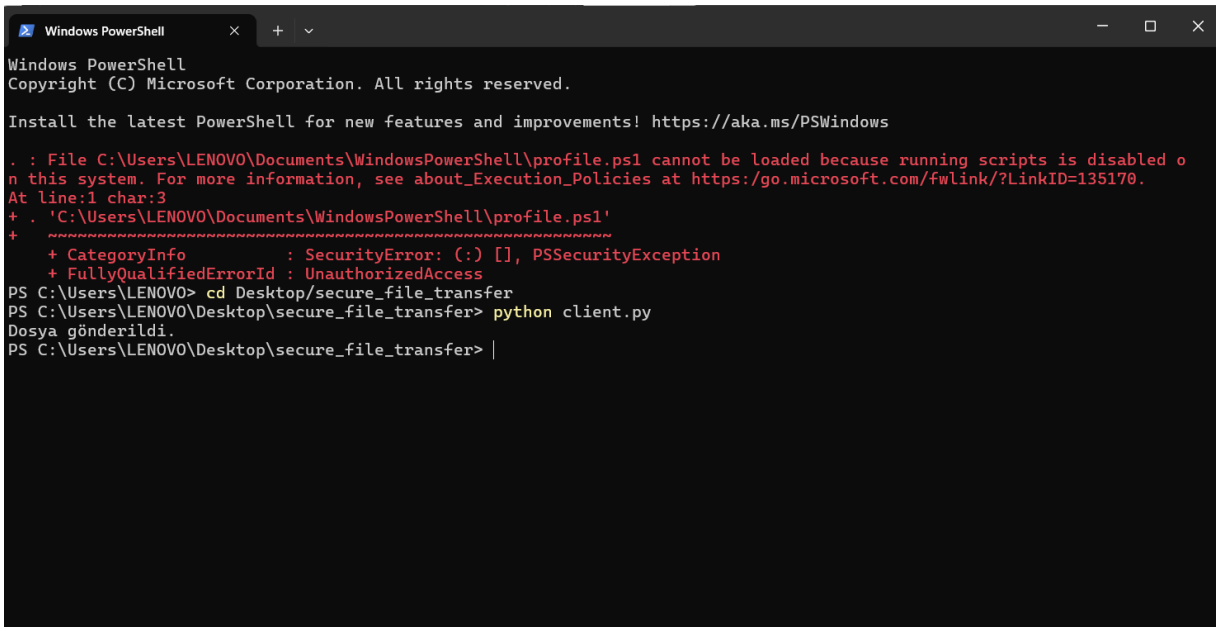


Açıklama:

Bu kod, sunucu tarafında gelen veriyi parçalar halinde alır, her parçanın checksum doğrulamasını yapar ve verileri doğru sırada birleştirerek şifre çözümünü gerçekleştirir. Dosya başarıyla kaydedildikten sonra işlem tamamlanır.

2.5 Kod Çalıştırma Sonuçları ve Açıklamaları

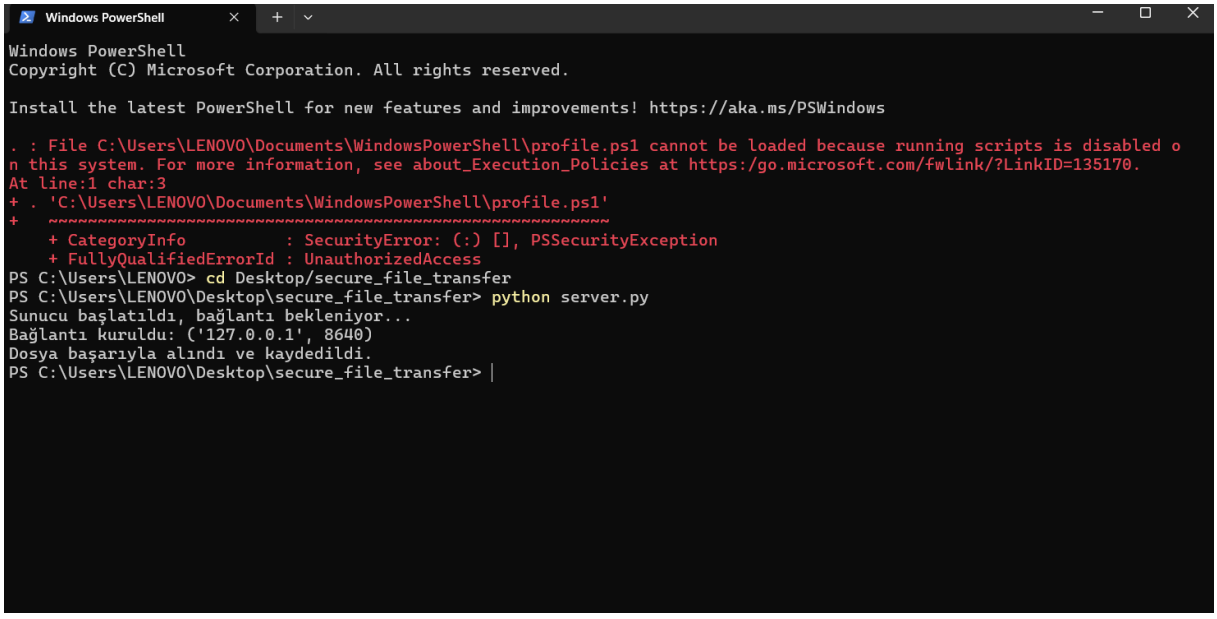
2.5.1 İstemci Tarafı Çıktısı



Açıklama:

İstemci tarafında dosya başarıyla şifrelenmiş ve parçalar halinde sunucuya gönderilmiştir. Çıktı mesajı dosya transferinin tamamlandığını göstermektedir

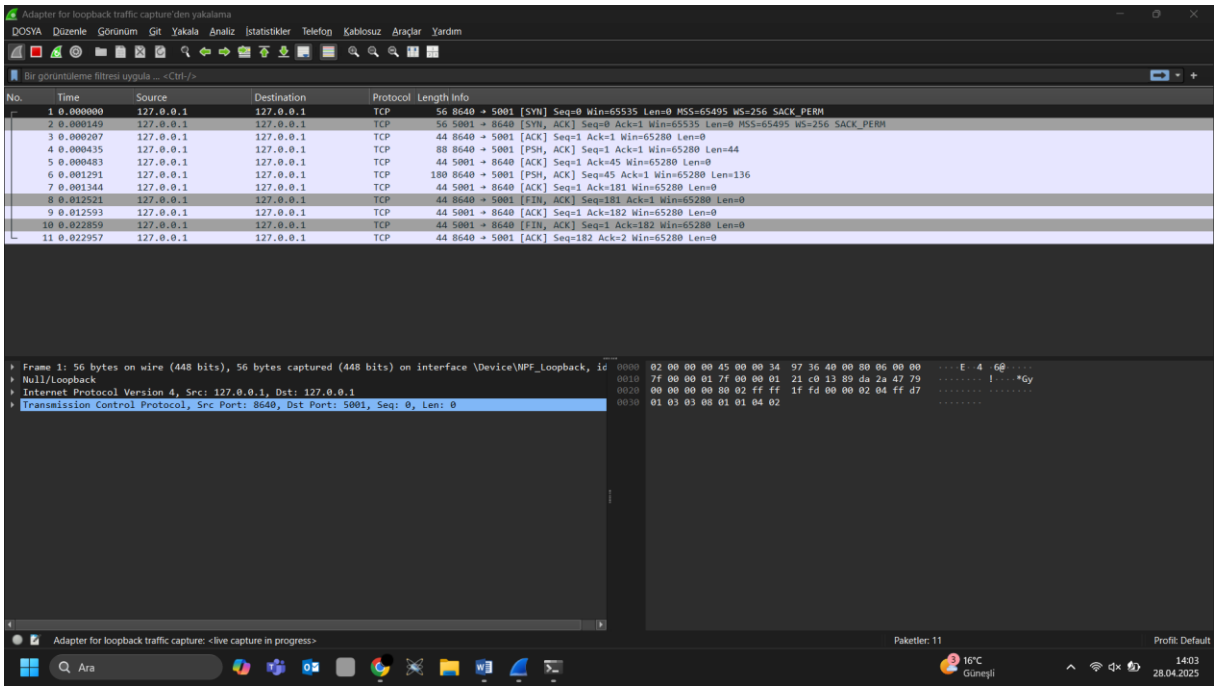
2.5.2 Sunucu Tarafı Çıktısı



Açıklama:

Sunucu tarafında bağlantı başarılı bir şekilde kurulmuş ve dosya doğru bir şekilde alınarak kaydedilmiştir. Herhangi bir checksum hatası olmadan tüm veriler başarıyla işlenmiştir.

2.6 Wireshark Trafik Analizi



Açıklama:

Wireshark kaydında TCP protokolü üzerinde transfer edilen şifreli veriler gözlemlenmiştir. Transfer edilen verilerin içerikleri okunamaz durumda olup şifrelemenin başarılı olduğu doğrulanmıştır.

3. SINIRLAMALAR VE GELECEKTEKİ İYİLEŞTİRMELER

3.1 Henüz Gerçekleştirilmemiş Çalışmalar

- IP başlıkları üzerinde manuel değişiklik (Flags, TTL, Checksum, Fragment Offset).
- iPerf kullanılarak ağ performans metriklerinin detaylı ölçümü.
- Man-in-the-Middle (MITM) saldırı senaryolarının test edilmesi.

3.2 Planlanan İyileştirmeler

- Hatalı paketlerin yeniden gönderilmesi için yeniden iletim mekanizması.
- Dinamik ağ uyum mekanizması (transfer hızının ağ koşullarına göre ayarlanması).
- Grafiksel kullanıcı arayüzü (GUI) geliştirilmesi.

4. SONUÇ

Bu aşamada, güvenli dosya transfer sisteminin temel fonksiyonları başarıyla gerçekleştirilmiştir. Dosyaların şifrelenmesi, parçalara ayrılması, paketlerin numaralandırılması ve bütünlüğünün sağlanması adımları başarıyla tamamlanmıştır. Ağ trafiği üzerinde şifreli verilerin transfer edildiği doğrulanmıştır. İlerleyen süreçte IP protokol seviyesi düzenlemeler ve gelişmiş ağ performans ölçümleri ile proje tamamlanacaktır.

5. KAYNAKÇA

- Python Software Foundation. (n.d.). Python Belgeleri. Erişim: https://docs.python.org/3/
- PyCA Cryptography. (n.d.). Cryptography Belgeleri. Erişim: https://cryptography.io/en/latest/
- Beej's Guide to Network Programming. (n.d.). Erişim: https://beej.us/guide/bgnet/html/
- RFC 791 - Internet Protocol. (1981). Erişim: https://datatracker.ietf.org/doc/html/rfc791
- Wireshark Foundation. (n.d.). Wireshark Kullanım Kılavuzu. Erişim: https://www.wireshark.org/