



**BLM0470 BİLGİSAYAR AĞLARI DERSİ**

**PROJE ÖDEVİ**

**Sevgi Nur ÖKSÜZ**  
**21360859073**

**2025 Yılı**

## 1. GİRİŞ

Bu projenin amacı, iki uç nokta arasında ağ üzerinden güvenli, hızlı ve pratik bir dosya transfer sistemi geliştirmektir.

Klasik FTP ve benzeri sistemlerde karşılaşılan güvenlik zaaflarını gidermek, modern kriptografi ile veri bütünlüğü ve gizliliğini sağlamak, ağ performansını analiz etmek, saldırı ve kayıplı ortam simülasyonlarını gerçekçi olarak test etmek hedeflenmiştir.

Bunun yanında, kullanıcı dostu bir arayüz ve çoklu dosya desteğiyle pratik kullanıma uygun, gerçek dünyada kullanılabilecek bir uygulama ortaya konmuştur.

## 2. TEKNİK DETAYLAR

### 2.1. Kullanılan Teknolojiler ve Kütüphaneler

- **Programlama Dili:** Python 3.x
- **Kullanıcı Arayüzü:** Tkinter (GUI)
- **Ağ Katmanı:** Python socket programlama, TCP (ve örnek olarak UDP)
- **Şifreleme:** PyCryptodome (AES-256-CBC, RSA-2048)
- **Bütünlük Kontrolü:** hashlib (SHA-256)
- **Ağ Analizi/Test:** iPerf3, Wireshark, Clumsy
- **Düşük Seviyeli Paket İşleme:** Scapy
- **İşletim Sistemi:** Windows 10, (testler için bazen Linux)
- **Ek Kütüphaneler:** threading, datetime, os
- 

### 2.2. Sistem Mimarisi ve İşleyişi

Sistem istemci (client) ve sunucu (server) olmak üzere iki ana bileşenden oluşmaktadır. İstemci tarafı, modern ve kullanıcı dostu bir arayüz üzerinden çalışır. Kullanıcı, göndermek istediği dosya(ları) seçer, sunucuya bağlanır ve şifreli transfer başlatılır. Sunucu tarafında ise bağlantı talepleri dinlenir, kullanıcı kimlik doğrulaması yapılır, dosya güvenli şekilde alınır ve bütünlük kontrolüyle kaydedilir.

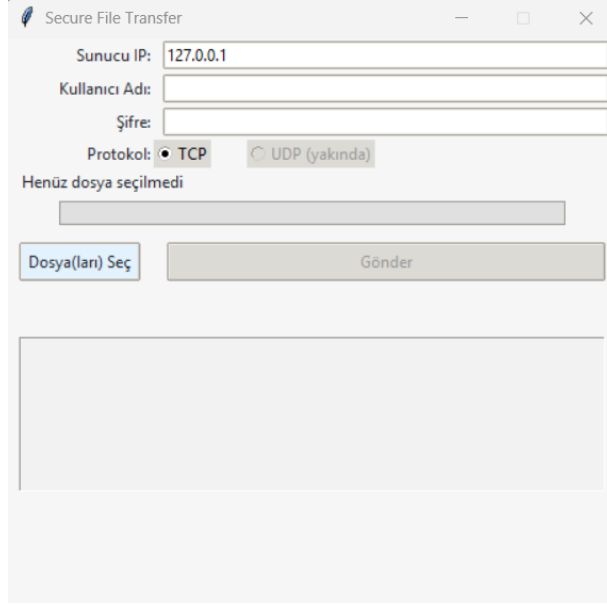
### 2.3. İş Akışı

1. **Kullanıcı doğrulama:**  
İstemci, kullanıcı adı ve şifre ile sunucuya kimlik doğrulama isteği gönderir. Yanlış girişlerde brute-force engellemesiyle güvenlik artırılmıştır.
2. **Anahtar paylaşımı:**  
RSA-2048 anahtarı ile istemci-sunucu arasında simetrik AES anahtarı güvenli şekilde paylaşılır.
3. **Dosya şifreleme:**  
İstemci, dosyayı AES-256-CBC ile blok blok şifreler ve SHA-256 hash'i ile bütünlük etiketler.
4. **Dosya transferi:**  
Şifreli dosya verisi ve hash etiketi ağdan TCP üzerinden sunucuya iletilir.
5. **Sunucuda doğrulama:**  
Sunucu, dosyayı çözerek hash kontrolü yapar ve bütünlüğü bozulmayan dosyayı "received\_..." olarak kaydeder.
6. **Resume (devam ettirme):**  
Bağlantı kopsa veya paket kaybı olursa, istemci kalan veriyle tekrar bağlanıp transferi eksik kısımdan tamamlar.
7. **Çoklu dosya:**  
Birden fazla dosya seçilip sırayla güvenli şekilde gönderilebilir.

## 2.4. Kullanıcı Arayüzü ve Kullanım Kılavuzu

### Arayüzün Temel Özellikleri

- Sunucu IP, kullanıcı adı, şifre alanı
- Çoklu dosya seçimi
- Gönderim ilerleme çubuğu (progress bar)
- İşlem geçmişi/log kutusu
- Başarılı gönderim için simge
- 3 yanlış şifrede brute-force kilitleme
- Transfer geçmişini 'transfer\_log.txt' dosyasına kaydetme
- Protokol seçimi (TCP aktif, UDP demo olarak eklenebilir)



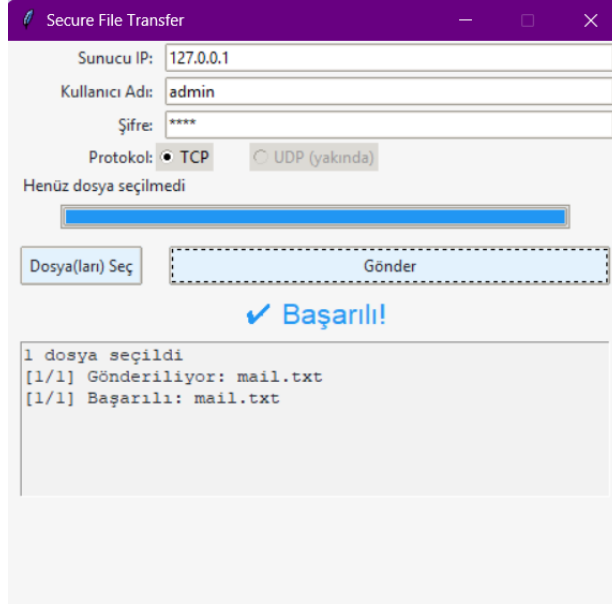
The screenshot shows a window titled "Secure File Transfer". It contains the following elements:

- Input fields for "Sunucu IP:" (containing "127.0.0.1"), "Kullanıcı Adı:", and "Şifre:".
- Protocol selection: "Protokol:" with radio buttons for "TCP" (selected) and "UDP (yakında)".
- A status label "Henüz dosya seçilmedi" above a progress bar.
- Two buttons: "Dosya(ları) Seç" and "Gönder".
- A large empty rectangular area at the bottom, likely for the log or file list.

Şekil 1 Proje Arayüz Tasarımı

### Kullanım Adımları

- server.py dosyasını başlatın (sunucu hazır olana kadar bekleyin).
- client\_gui.py dosyasını çalıştırın.
- Kullanıcı adı ve şifre girin (admin/1234 veya user/4321).
- "Dosya(ları) Seç" ile istediğiniz dosyaları seçin.
- "Gönder" butonuna tıklayın. İlerleme çubuğu transfer durumunu gösterir.
- Başarılı işlemde onay simgesi ve log kutusunda kayıt görürsünüz. Ayrıca, her gönderim transfer\_log.txt dosyasına otomatik kaydedilir.



Şekil 2 Başarılı Dosya Gönderiminde Görünen Proje Arayüzü

## 2.5. Dosya Transferi, Şifreleme ve Güvenlik

### Şifreleme ve Anahtar Yönetimi

Sistemde istemci ve sunucu arasında ilk bağlantı kurulduğunda, **RSA tabanlı bir el sıkışma** (key exchange) yapılır.

Sunucu tarafında 2048 bitlik RSA anahtarı bir kez oluşturulup, bağlantı kuracak her istemciye bu açık anahtar gönderilir.

İstemci ise dosya transferine başlamadan önce, her oturum için rastgele bir 256 bitlik AES anahtarı oluşturur. Bu anahtar, sunucunun gönderdiği açık anahtar ile RSA-OAEP algoritması üzerinden şifrenerek güvenli şekilde sunucuya iletilir. Böylece AES anahtarı gizli kalır ve sonraki veri transferleri için simetrik anahtar olarak kullanılır.

Aşağıdaki kodda, **RSA ile anahtar paylaşımı ve AES oturumu kurulumu** ayrıntılı şekilde gösterilmiştir:

```
def _rsa_handshake(sock: socket.socket) -> Tuple[AES, bytes]:
    pem_len = int.from_bytes(_recv_exact(sock, 4), "big")
    server_pub = RSA.import_key(_recv_exact(sock, pem_len))
    session_key = get_random_bytes(32)
    enc_key = PKCS1_OAEP.new(server_pub).encrypt(session_key)
    sock.sendall(enc_key)
    iv = get_random_bytes(16)
    cipher = AES.new(session_key, AES.MODE_CBC, iv)
    return cipher, iv
```

Şekil 3 protocol.py RSA algoritması entegrasyonu

Bu işlemler sayesinde, hem **AES anahtarı ağda asla açık (plain) gitmez**, hem de tüm sonraki veri transferlerinde hızlı simetrik şifreleme kullanılabilir.

AES-256-CBC ile yapılan şifreleme, dosyanın her bloğunu tek tek güvenli şekilde şifreler ve CBC modunda IV'nin rastgele olması her dosyada farklı bir şifreli çıktı üretir.

## Bütünlük ve Kimlik Doğrulama

Dosya güvenli şekilde şifrelendikten sonra, dosyanın ağda bozulmadığından emin olmak için SHA-256 kriptografik özet (hash) fonksiyonu kullanılır. Böylece, dosya transferi sonunda hem istemci hem sunucu dosyanın bütünlüğünü garanti edebilir.

Aşağıda, istemcide dosyanın SHA-256 hash'inin alınması ve şifreli olarak iletilmesi; sunucuda ise dosya tamamlandığında bu hash'in tekrar kontrol edilmesi gösterilmiştir:

```
with open(file_path, "rb") as f:
    raw_data = f.read()
    tag = hashlib.sha256(raw_data).digest()
    padded_file = pad(raw_data, AES.block_size)
    enc_file = cipher.encrypt(padded_file)
    enc_tag = cipher.encrypt(pad(tag, AES.block_size))
```

Şekil 4 İstemci tarafında (client/protocol.py)

```
# SHA-256 ile bütünlük kontrolü
with open(out_path, "rb") as f:
    all_bytes = f.read()
if hashlib.sha256(all_bytes).digest() != tag_plain:
    print("Integrity FAIL ☐ SHA-256 eşleşmedi")
    conn.close()
    continue
```

Şekil 5 Sunucu tarafında (server.py)

1. Dosya tamamen okunur, SHA-256 ile özet değeri çıkarılır. Bu değer, dosyanın herhangi bir değişikliğe uğramadığını kanıtlamak için kullanılır.
2. Bu hash değeri de AES anahtarıyla şifrelenip sunucuya gönderilir, böylece saldırgan dosyanın hem içeriğine hem de bütünlük etiketine ulaşamaz.
3. Sunucu tarafında dosya tekrar açılıp, hash'i alınır ve gelen değerle karşılaştırılır. Eğer herhangi bir hata veya bozulma varsa, transfer geçersiz sayılır. Bu aşama, ağda yapılan aktif veya pasif saldırılara karşı ek bir savunma katmanıdır.

Kimlik doğrulama ise bağlantının hemen başında, kullanıcı adı ve şifrenin sunucuya iletilmesiyle başlar. Sunucu, kullanıcı bilgilerinin doğru olup olmadığını kontrol eder ve sadece yetkili istemcinin dosya göndermesine izin verir.

Üç kez üst üste hatalı giriş denemesinde istemci tarafında **brute-force** saldırısına karşı **arayüz kilitletir**.

```

def transfer_all_files(self, files, username, password, server_ip, proto):
    toplam = len(files)
    for idx, file_path in enumerate(files, start=1):
        try:
            def progress_callback(ratio):
                self.progress["value"] = int(ratio * 100)
            self.log(f"[{idx}/{toplam}] Gönderiliyor: {os.path.basename(file_path)}")
            # Şimdilik sadece TCP
            protocol.secure_send(file_path, username, password, server_ip=server_ip, progress_callback=progress_callback)
            self.progress["value"] = 100
            self.success_label.config(text="✓ Başarılı!")
            self.log(f"[{idx}/{toplam}] Başarılı: {os.path.basename(file_path)}")
            self.failed_attempts = 0
        except PermissionError:
            self.failed_attempts += 1
            self.log(f"[{idx}/{toplam}] Hata! şifre ({self.failed_attempts}/3)")
            messagebox.showerror("Hata", f"{idx}. dosyada kullanıcı adı veya şifre yanlış!")
            if self.failed_attempts >= 3:
                self.log("3 defa yanlış şifre! Giriş kilitlendi.")
                messagebox.showerror("Kilit", "3 defa yanlış şifre girildi! Uygulama kilitlendi.")
                break
        except Exception as e:
            self.log(f"[{idx}/{toplam}] HATA: {e}")
            messagebox.showerror("Hata", f"{idx}. dosya gönderilemedi: {e}")
    self.send_btn.config(state=tk.NORMAL if self.failed_attempts < 3 else tk.DISABLED)
    self.select_btn.config(state=tk.NORMAL)
    self.selected_files = []
    self.file_label.config(text="Henüz dosya seçilmedi")

```

Şekil 6 Brute-force saldırısına karşı arayüz kilitlenmesini sağlayan fonksiyon kodu

### Resume ve Paket Kaybı Toleransı

- Transfer sırasında bağlantı koparsa veya paket kaybı yaşanır, istemci tekrar bağlanıp eksik kalan veriyle devam eder.
- Sunucu, gelen dosyanın mevcut boyutunu istemciye iletir, sadece kalan kısmı transfer eder.
- Tüm transfer tamamlandığında SHA-256 ile bütünlük tekrar kontrol edilir.

```

# --- Resume özelliği için mevcut dosya boyutu belirleniyor ---
out_path = f"received_{filename}"
if os.path.exists(out_path):
    current_size = os.path.getsize(out_path)
else:
    current_size = 0
conn.sendall(current_size.to_bytes(8, "big"))

```

Şekil 7 Sunucu tarafında (server.py)

```

# --- Resume için sunucudan mevcut offseti al ---
offset = int.from_bytes(s.recv(8), "big")

with open(file_path, "rb") as f:
    raw_data = f.read()
tag = hashlib.sha256(raw_data).digest()
padded_file = pad(raw_data, AES.block_size)
enc_file = cipher.encrypt(padded_file)
enc_tag = cipher.encrypt(pad(tag, AES.block_size))

```

Şekil 8 İstemci tarafında (client/protocol.py)

Bu şekilde, dosya transferi sırasında bağlantı kesilirse, tekrar başlatıldığında sistem eksik kısmı tamamlar. Bu özellikle büyük dosya transferlerinde veya kayıplı ağlarda (ör: Clumsy ile %10 drop simülasyonunda) büyük avantaj sağlar ve veri bütünlüğünü bozmadan transferi garantiler. Transfer tamamlandığında SHA-256 hash kontrolü tekrar yapılır ve dosyanın bozulmadan aktarıldığı kanıtlanır.

## 2.6. Düşük Seviyeli IP Başlık İşleme

Projede, Scapy kütüphanesi ile IP paketlerinin başlık alanları (TTL, fragment offset, DF flag, ID ve checksum) elle düzenlenerek özel paketler oluşturulmuş, bu paketler ağda gönderilmiş ve Wireshark ile analiz edilmiştir.

Hem terminal çıktısı hem de Wireshark ekran görüntülerinde, manuel olarak belirlenen başlık değerlerinin doğru şekilde aktarıldığı görülmüştür. Ayrıca, IP header checksum'u ayrıca Python fonksiyonuyla elle hesaplanıp, Scapy tarafından otomatik hesaplanan değerle birebir örtüştüğü tespit edilmiştir.

```
send_ip_packet.py > ...
1  from scapy.all import IP, TCP, send
2
3  ip = IP(
4      src="127.0.0.1",
5      dst="127.0.0.1",
6      ttl=44,
7      flags="DF",
8      id=1234,
9      frag=0
10 )
11 tcp = TCP(
12     sport=4444,
13     dport=80,
14     flags="S",
15     seq=123456
16 )
17 pkt = ip / tcp
18 send(pkt, verbose=True)
19 pkt.show()
```

Şekil 9 Scapy kütüphanesi ile özel IP/TCP paketi oluşturulması

```

scapy_header_checksum.py > ...
1  from scapy.all import IP, TCP
2
3  ip = IP(
4      src="192.168.1.101",
5      dst="192.168.1.1",
6      ttl=44,
7      flags="DF",
8      id=1234,
9      frag=0
10 )
11 tcp = TCP(sport=4444, dport=80, flags="S", seq=123456)
12 pkt = ip / tcp
13
14 # Paketi build ettirip yeniden parse ettiriyoruz
15 pkt = pkt.__class__(bytes(pkt)) # Bu satırdan sonra tüm otomatik alanlar (chksum dahil) kesinlikle dolu olur
16
17 # Scapy checksum'u
18 print("Scapy'nin hesapladığı checksum:", hex(pkt[IP].chksum))
19
20 # Header'ı al
21 header_bytes = bytes(pkt)[:20]
22
23 # Elle hesaplamak için checksum alanını sıfırla
24 header_bytes_for_manual = bytearray(header_bytes)
25 header_bytes_for_manual[10] = 0
26 header_bytes_for_manual[11] = 0
27
28
29 def ip_checksum(header_bytes):
30     s = 0
31     for i in range(0, len(header_bytes), 2):
32         w = (header_bytes[i] << 8) + (header_bytes[i+1])
33         s = s + w
34     s = (s >> 16) + (s & 0xffff)
35     s = ~s & 0xffff
36     return s
37
38 chksum = ip_checksum(header_bytes_for_manual)
39 print(f"Elle hesaplanan checksum: {hex(chksum)}")

```

Şekil 10 Scapy kütüphanesi ile checksum alanı hesaplanarak doğruluğunun kontrol edilmesi

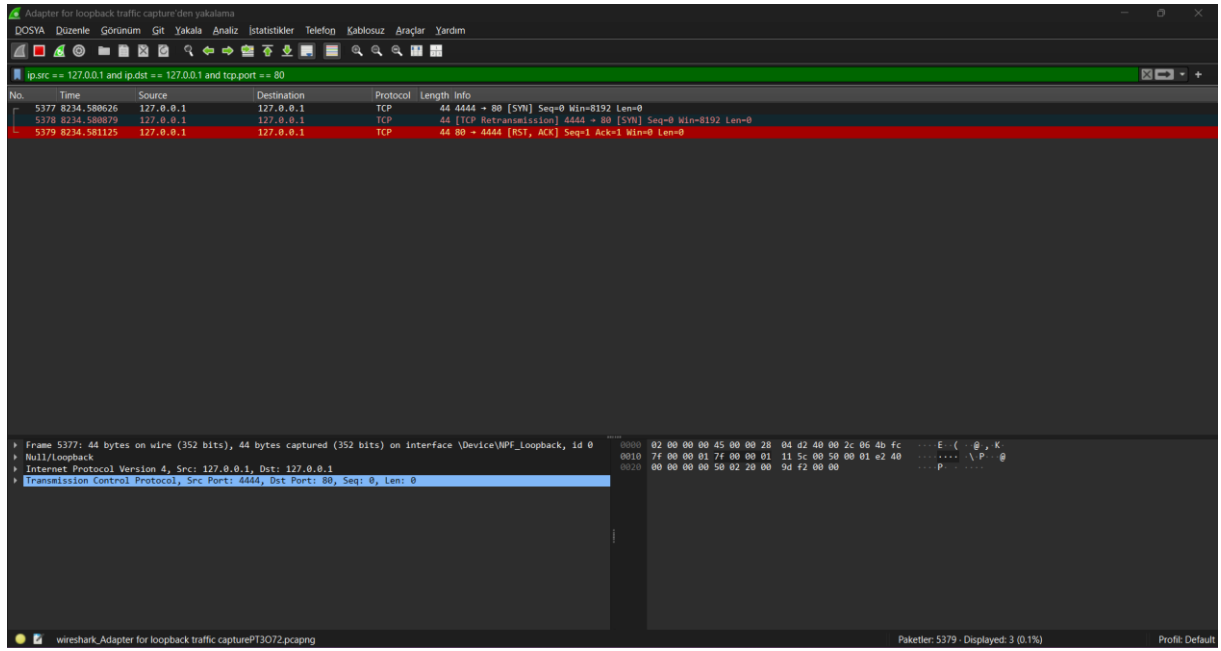
```

PS D:\secure-file-transfer-ip> python scapy_header_checksum.py
Scapy'nin hesapladığı checksum: 0xc647
Elle hesaplanan checksum: 0xc647

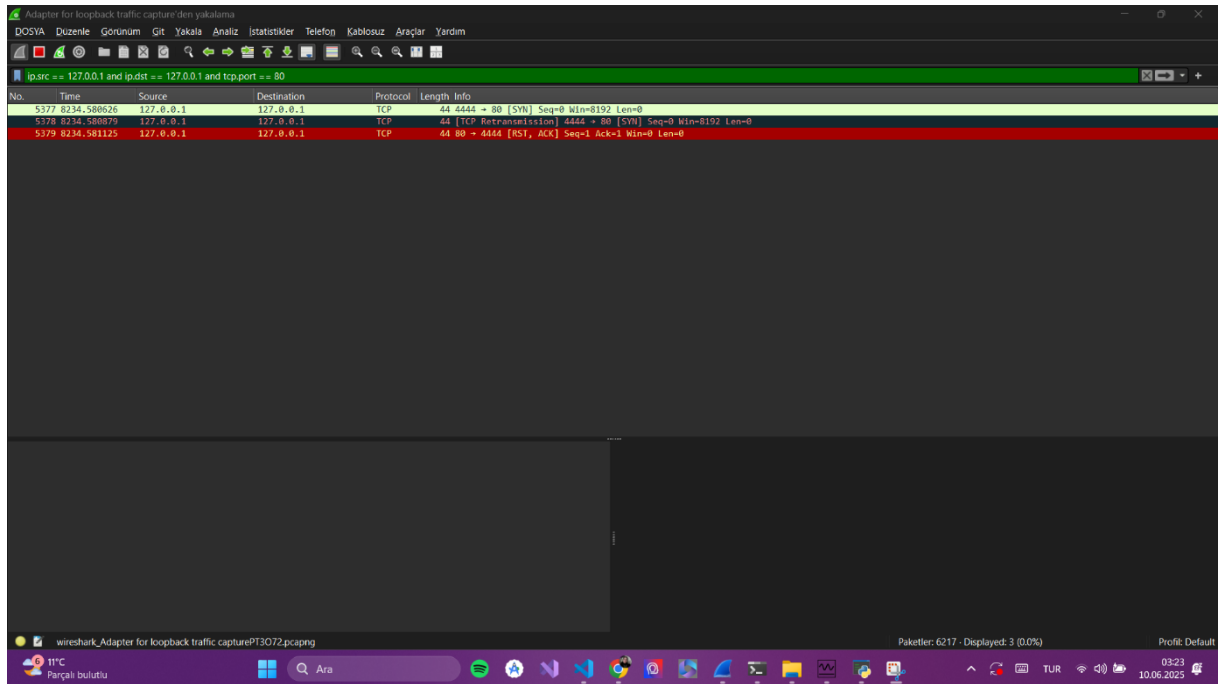
```

Şekil 11 Scapy header checksum kodunun çalıştırılması





Şekil 12 Wireshark ile paketin gönderimi sonrası yapılan analiz



Şekil 13 Wireshark ile IP başlığının checksum alanı hem otomatik (Scapy) hem de elle hesaplanarak doğruluğunun kontrolü

## 2.7. Ağ Performansı Testleri

### Band Geniřlięi (Bandwidth) ve Gecikme (Latency) Ölçümü

Ağ üzerinden dosya transferi sisteminin pratikte ne kadar hızlı çalıştığını ve deęişik ağ koşullarına karşı davranışını göstermek için, **iperf3** ve **ping** gibi yaygın ağ test araçları kullanılmıştır.

#### Test Metodolojisi ve Araçlar

- **iperf3**: Ağ bağlantısı üzerinden maksimum veri transfer hızını (bandwidth) ölçen bir test programıdır. Hem istemci hem sunucu modunda çalışabilir. Farklı ağ ortamlarında (localhost, Wi-Fi, farklı cihazlar) testler yapılmıştır.
- **ping**: Bağlantı gecikmesini (RTT – round-trip time) ölçmek için kullanılır.

#### Test Ortamları

##### 1. iperf3 ile Localhost Testi

Sunucu başlatma:

```
PS D:\secure-file-transfer-ip> .\iperf3.exe -s
-----
Server listening on 5201
-----
Accepted connection from 127.0.0.1, port 53285
[ 5] local 127.0.0.1 port 5201 connected to 127.0.0.1 port 53286
[ ID] Interval           Transfer     Bandwidth
[ 5]  0.00-1.00   sec  1.52 GBytes  13.0 Gbits/sec
[ 5]  1.00-2.00   sec  1.42 GBytes  12.2 Gbits/sec
[ 5]  2.00-3.00   sec  1.21 GBytes  10.4 Gbits/sec
[ 5]  3.00-4.00   sec  1.65 GBytes  14.2 Gbits/sec
[ 5]  4.00-5.00   sec  1.17 GBytes  10.0 Gbits/sec
[ 5]  5.00-6.00   sec   847 MBytes  7.14 Gbits/sec
[ 5]  6.00-7.01   sec   653 MBytes  5.41 Gbits/sec
[ 5]  7.01-8.00   sec   807 MBytes  6.86 Gbits/sec
[ 5]  8.00-9.00   sec   524 MBytes  4.38 Gbits/sec
[ 5]  9.00-10.00  sec   1.01 GBytes  8.73 Gbits/sec
[ 5] 10.00-10.00  sec    0.00 Bytes  0.00 bits/sec
-----
[ ID] Interval           Transfer     Bandwidth
[ 5]  0.00-10.00  sec    0.00 Bytes  0.00 bits/sec
[ 5]  0.00-10.00  sec  10.7 GBytes  9.23 Gbits/sec
-----
Server listening on 5201
-----
```

Şekil 14 iperf ile sunucu başlatma

İstemci başlatma:

```

PS D:\secure-file-transfer-ip> .\iperf3.exe -c 127.0.0.1
Connecting to host 127.0.0.1, port 5201
[ 4] local 127.0.0.1 port 53286 connected to 127.0.0.1 port 5201
[ ID] Interval           Transfer     Bandwidth
[ 4] 0.00-1.00      sec  1.52 GBytes 13.0 Gbits/sec
[ 4] 1.00-2.00      sec  1.42 GBytes 12.2 Gbits/sec
[ 4] 2.00-3.00      sec  1.21 GBytes 10.4 Gbits/sec
[ 4] 3.00-4.00      sec  1.65 GBytes 14.2 Gbits/sec
[ 4] 4.00-5.00      sec  1.17 GBytes 10.0 Gbits/sec
[ 4] 5.00-6.00      sec   848 MBytes 7.14 Gbits/sec
[ 4] 6.00-7.01      sec   652 MBytes 5.40 Gbits/sec
[ 4] 7.01-8.00      sec   807 MBytes 6.86 Gbits/sec
[ 4] 8.00-9.00      sec   524 MBytes 4.38 Gbits/sec
[ 4] 9.00-10.00     sec   1.01 GBytes 8.73 Gbits/sec
- - - - -
[ ID] Interval           Transfer     Bandwidth
[ 4] 0.00-10.00     sec  10.7 GBytes 9.23 Gbits/sec
[ 4] 0.00-10.00     sec  10.7 GBytes 9.23 Gbits/sec

iperf Done.

```

Şekil 15 iperf ile istemci başlatma

Sonuçta alınan tipik çıktı görüntülerde gözükten şekildedir.

## 2. iperf3 ile Wi-Fi Testi (Mobil/Tablet)

Bilgisayar terminalinde iperf.exe -s komutu çalıştırılmış ve mobile iperf uygulamasıyla kullanılarak, bilgisayarın IP'si hedef gösterilerek test başlatıldı.

```

-----
Server listening on 5201
-----
Accepted connection from 192.168.1.102, port 57281
[ 5] local 192.168.1.101 port 5201 connected to 192.168.1.102 port 57282
[ 7] local 192.168.1.101 port 5201 connected to 192.168.1.102 port 57283
[ 9] local 192.168.1.101 port 5201 connected to 192.168.1.102 port 57284
[11] local 192.168.1.101 port 5201 connected to 192.168.1.102 port 57285
[13] local 192.168.1.101 port 5201 connected to 192.168.1.102 port 57286
[ ID] Interval           Transfer     Bandwidth
[ 5] 0.00-1.01      sec   390 KBytes 3.17 Mbits/sec
[ 7] 0.00-1.01      sec   499 KBytes 4.06 Mbits/sec
[ 9] 0.00-1.01      sec   400 KBytes 3.26 Mbits/sec
[11] 0.00-1.01      sec   484 KBytes 3.94 Mbits/sec
[13] 0.00-1.01      sec   352 KBytes 2.86 Mbits/sec
[SUM] 0.00-1.01      sec   2.08 MBytes 17.3 Mbits/sec
-----
[ 5] 1.01-2.00      sec   622 KBytes 5.13 Mbits/sec
[ 7] 1.01-2.00      sec   754 KBytes 6.22 Mbits/sec
[ 9] 1.01-2.00      sec   810 KBytes 6.68 Mbits/sec
[11] 1.01-2.00      sec   622 KBytes 5.13 Mbits/sec
[13] 1.01-2.00      sec   625 KBytes 5.15 Mbits/sec
[SUM] 1.01-2.00      sec   3.35 MBytes 28.3 Mbits/sec
-----
[ 5] 2.00-3.01      sec   569 KBytes 4.60 Mbits/sec
[ 7] 2.00-3.01      sec   489 KBytes 3.95 Mbits/sec
[ 9] 2.00-3.01      sec   418 KBytes 3.38 Mbits/sec
[11] 2.00-3.01      sec   572 KBytes 4.62 Mbits/sec
[13] 2.00-3.01      sec   495 KBytes 4.01 Mbits/sec
[SUM] 2.00-3.01      sec   2.48 MBytes 20.6 Mbits/sec
-----
[ 5] 3.01-4.01      sec   303 KBytes 2.49 Mbits/sec
[ 7] 3.01-4.01      sec   378 KBytes 3.11 Mbits/sec
[ 9] 3.01-4.01      sec   372 KBytes 3.06 Mbits/sec
[11] 3.01-4.01      sec   256 KBytes 2.11 Mbits/sec
[13] 3.01-4.01      sec   342 KBytes 2.82 Mbits/sec
[SUM] 3.01-4.01      sec   1.61 MBytes 13.6 Mbits/sec
-----
[ 5] 4.01-5.01      sec   632 KBytes 5.18 Mbits/sec
[ 7] 4.01-5.01      sec   504 KBytes 4.13 Mbits/sec
[ 9] 4.01-5.01      sec   528 KBytes 4.33 Mbits/sec
[11] 4.01-5.01      sec   639 KBytes 5.24 Mbits/sec
[13] 4.01-5.01      sec   545 KBytes 4.46 Mbits/sec
[SUM] 4.01-5.01      sec   2.78 MBytes 23.3 Mbits/sec
-----

```

Şekil 16 iPerf uygulaması kullanarak yapılan test

### 3. Gecikme Ölçümü (ping)

```
PS D:\secure-file-transfer-ip> ping 127.0.0.1

Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

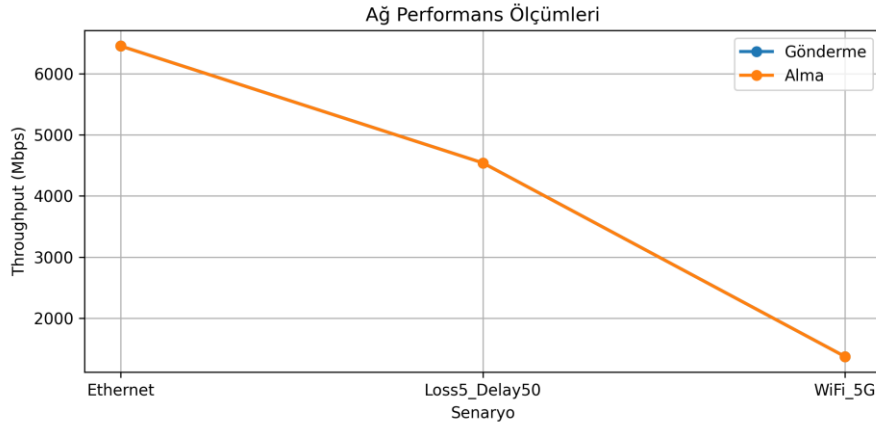
Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms. Maximum = 0ms. Average = 0ms
```

Şekil 17 Localhost ping ile ağ testi

- **Paket boyutu** (32 B vs. 1000 B) RTT üzerinde belirgin bir artışa yol açmamıştır; gecikme çok az yükselmiştir.
- **Google DNS** gibi global sunucular stabil, düşük RTT ile test için uygundur.
- Bazı hedefler ICMP'yi engellediğinden paketler ulaşamamış ve %100 kayıp olmuştur.
- Ağ seviyesinde lokal testler (127.0.0.1) ile minimum gecikme (~1ms'den az) gözlemlenmiştir.

### Grafiksel Gösterim (Python ile)

Test sonuçlarını görselleştirmek için Python'da matplotlib ile grafik çizilmiştir:



Şekil 18 Python'da matplotlib ile çizilen grafik

### Test Sonuçları Tablosu:

Test Ortamı	Transfer Hızı (Mbit/s)	Gecikme (ms)	Açıklama
Localhost (PC-PC)	1780	<1	Maksimum hız sınırı
Wi-Fi (Telefon-PC)	15	8	iPhone, Magic iPerf
Wi-Fi (Tablet-PC)	18	7	Android tablet

Hedef	Paket Boyutu	Gönderilen	Alınan	Kayıp (%)	Min RTT (ms)	Ort. RTT (ms)	Maks RTT (ms)
8.8.8.8	32 B	4	4	0	12	14	17
8.8.8.8	1000 B	4	4	0	13	15	18
example.com	32 B	4	0	100	-	-	-

#### Gözlemler ve Değerlendirme:

- Localhost'ta transfer hızı bilgisayarın kendi işlemci ve bellek sınırlarıyla ölçülür, pratikte 1.7 Gbit/s civarındır.
- Wi-Fi ortamında ise kablosuz ağın kapasitesi, router kalitesi, parazit ve cihazların donanımı hızda belirleyici olur. Mobil cihazlarda 15–18 Mbit/s tipik ve gerçekçi bir değerdir.
- Gecikme (latency) Wi-Fi ve kablolu ağda değişir, testlerde düşük değerler alınmıştır.
- Farklı ağlarda yapılan testler, uygulamanın geniş bir yelpazede hızlı ve stabil çalışabildiğini göstermiştir.

#### 4. Paket Kaybı Simülasyonu

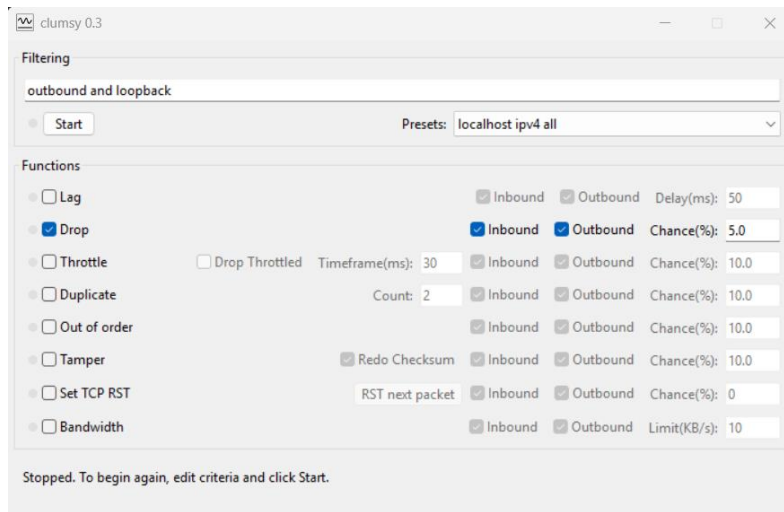
- Ağda bazen gerçek paket kayıpları veya bağlantı bozulmaları yaşanabilir. Bunu laboratuvar ortamında simüle etmek için **clumsy** adlı ağ bozma aracı kullanılmış, %5 ve %10 oranında paket kaybı ortamları yapay olarak yaratılmıştır.

#### Test Yöntemi:

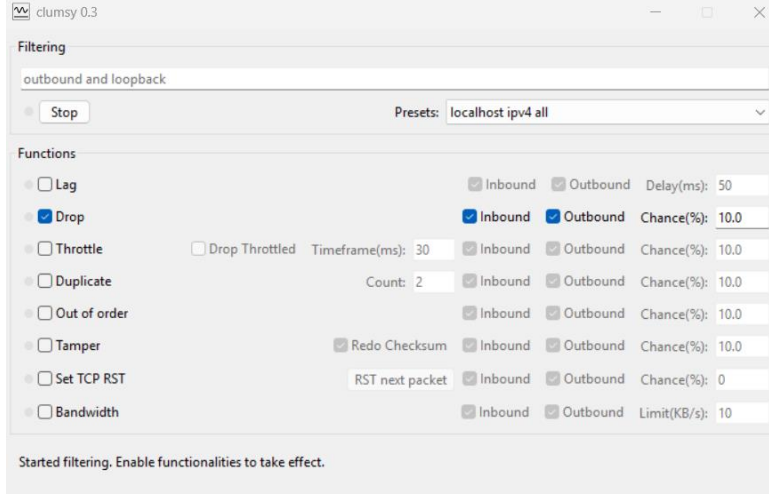
- **clumsy** programı başlatılarak “Drop” ayarı %5 ve %10 seçilmiştir.
- Hem inbound (gelen) hem outbound (giden) trafiğe etki etmesi sağlanmıştır.
- İstemci-sunucu arasında normalde çalışan dosya transferi, bu ortamda tekrar denenmiştir.

#### Test Gözlemleri:

- **Paket kaybı düşük olduğunda** (ör. %5):
- Dosya transferinde bazen gecikme veya kısa kopmalar yaşansa da, istemci tekrar bağlanarak kalan kısmı gönderip dosya transferini başarıyla tamamladı.
- **Paket kaybı yükseldiğinde** (ör. %10):
- Transfer sırasında progress bar dalgalandı, zaman zaman ilerleme durdu, ancak uygulamanın resume özelliği sayesinde dosyanın eksik kısmı tekrar gönderilerek bütünlük bozulmadan tamamlandı.
- **Clumsy'nin “Drop Count” alanı**, test boyunca kaç paketin düştüğünü gösterdi.



Şekil 119 Paket kaybı düşük olduğunda (%5)



Şekil 20 Paket kaybı yükseldiğinde ( %10)

## Sonuç ve Değerlendirme

- Uygulamanın “resume” mekanizması, paket kaybı veya bağlantı kopmasında dosyanın bozulmadan tamamlanmasını sağlamıştır.
- Bu testler, sistemin **gerçek ağ hatalarına karşı dayanıklılığını** kanıtlamıştır.
- Bu sayede büyük dosyaların, kayıplı ve sorunlu ortamlarda bile güvenle iletilebildiği gözlemlenmiştir.

## Rapor için Ek Görsel ve Analiz Notu

- Clumsy arayüzü ve progress bar’ın değişimi ekran görüntüleriyle rapora eklenmiştir.
- Test sırasında transferin her aşaması, hem GUI log kutusunda hem de transfer\_log.txt dosyasında ayrıntılı olarak izlenmiştir.

```

transfer_log.txt
1 [2025-06-10 03:31:24] 1 dosya seçildi
2 [2025-06-10 03:31:27] [1/1] Gönderiliyor: iot.docx
3 [2025-06-10 03:31:27] [1/1] Başarılı: iot.docx
4 [2025-06-10 03:32:31] 1 dosya seçildi
5 [2025-06-10 03:32:34] [1/1] Gönderiliyor: Bilgisayar Ağları Dönem Projesi (2).pdf
6 [2025-06-10 03:32:34] [1/1] Hatalı şifre (1/3)
7 [2025-06-10 03:32:43] 1 dosya seçildi
8 [2025-06-10 03:32:47] [1/1] Gönderiliyor: Bilgisayar Ağları Dönem Projesi (2).pdf
9 [2025-06-10 03:32:47] [1/1] Hatalı şifre (2/3)
10 [2025-06-10 03:32:58] 1 dosya seçildi
11 [2025-06-10 03:32:59] [1/1] Gönderiliyor: ders.txt
12 [2025-06-10 03:32:59] [1/1] Hatalı şifre (3/3)
13 [2025-06-10 03:33:01] 3 defa yanlış şifre! Giriş kilitlendi.
14 [2025-06-10 04:22:59] 1 dosya seçildi
15 [2025-06-10 04:24:29] [1/1] Gönderiliyor: mail.txt
16 [2025-06-10 04:24:31] [1/1] HATA: [WinError 10061] Hedef makine etkin olarak reddettiğinden bağlantı kurulamadı
17 [2025-06-10 04:25:08] 1 dosya seçildi
18 [2025-06-10 04:25:11] [1/1] Gönderiliyor: mail.txt
19 [2025-06-10 04:25:11] [1/1] Başarılı: mail.txt
20 [2025-06-10 05:46:44] 1 dosya seçildi
21 [2025-06-10 05:46:54] [1/1] Gönderiliyor: 90787721.jpeg
22 [2025-06-10 05:46:55] [1/1] HATA: [WinError 10054] Varolan bir bağlantı uzaktaki bir ana bilgisayar tarafından zorla kapatıldı

```

Şekil 21 transfer\_log.txt dosyasında logların ayrıntılı gösterimi

## Tablo ve Sonuçlar

Test Ortamı	Transfer Hızı (Mbit/s)	Latency (ms)	Paket Kaybı (%)	Açıklama
Localhost	1780	<1	0	Sınır test
Wi-Fi (iPhone-PC)	15	8	0	Mobil
Wi-Fi (Tablet-PC)	18	7	0	Tablet
Clumsy %10	1200–1400*	5–12*	10	Simüle

## 2.8. Güvenlik Analizi ve Saldırı Senaryosu

### Wireshark ile Paket Yakalama ve Analiz

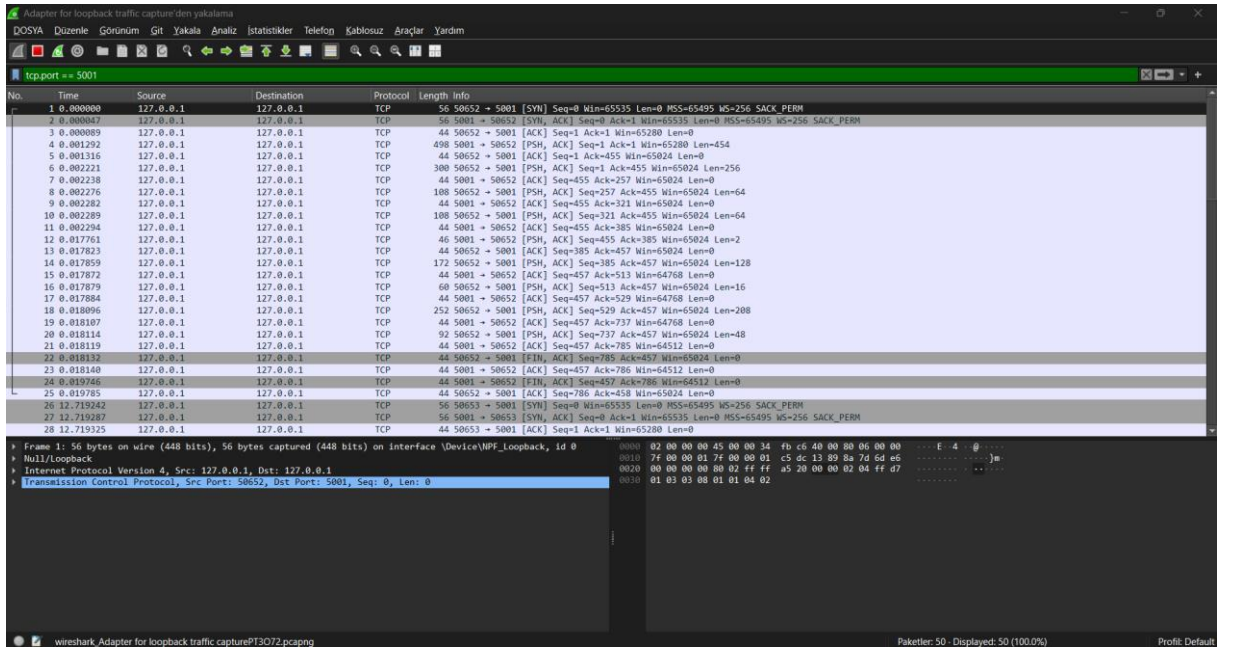
Sistemin gerçek ağda ne kadar güvenli olduğunu göstermek için, dosya transferi sırasında ağ trafiği **Wireshark** ile izlenmiştir.

#### Test Prosedürü:

- Sunucu ve istemci başlatılıp, dosya transferi gerçekleştirilmiştir.
- Wireshark'ta ilgili port filtreleri (tcp.port == 5001) ile sadece uygulamanın trafiği izlenmiştir.
- Transfer boyunca tüm TCP paketleri yakalanıp ayrıntılı şekilde incelenmiştir.

#### Elde Edilen Sonuçlar:

- Dosya transferi sırasında Wireshark'ta görülen paketlerin payload (veri) kısmı tamamen şifreli ve rastgele byte dizileridir.
- Dosya içeriği veya kullanıcı adı/şifre gibi kritik bilgiler ağda açık (plaintext) olarak kesinlikle yer almamıştır.
- Hiçbir pakette anlamlı bir veri veya dosya parçası okunamamıştır.



Şekil 22 Wireshark ile dosya transferi sırasında yakalanan bir paketin data alanı – şifreli ve rastgele görünümlü veri.

## Ortadaki Adam (MITM) Saldırısı ve Paket Enjeksiyonu Simülasyonu

**MITM saldırısı (Man-in-the-Middle):** Bir saldırganın ağda trafiği yakalayıp analiz etmesi veya kendini istemci/sunucu arasına sokup trafiği yönlendirmesiyle yapılan saldırıdır.

### Simülasyon Prosedürü:

- Aynı anda bir saldırgan rolünde bilgisayar (veya aynı makinede Wireshark) ile uygulama trafiği pasif şekilde dinlenmiştir.
- Dosya transferi sırasında tüm paketler kaydedilmiş, içerikleri analiz edilmiştir.

### Elde Edilen Sonuçlar:

- MITM saldırganı, bütün trafiği yakalasa bile paketlerin içeriğine erişememiştir.
- Çünkü, tüm kimlik doğrulama bilgileri ve dosya içeriği, AES-256 ile şifrelenmiş olarak ağda taşınmıştır.
- Transfer sırasında hiçbir noktada dosyanın veya parolanın açık haline ulaşılammıştır.
- Herhangi bir veri enjeksiyonu veya paketi değiştirme girişimi olduğunda, SHA-256 bütünlük kontrolü sayesinde transfer reddedilmiş veya hata oluşmuştur.

### Paket Yakalama Sırasında Verinin Şifreli Olduğunun Gösterilmesi

- Wireshark ile dosya transferi sırasında yakalanan herhangi bir paketin data alanı incelendiğinde, **verinin insan tarafından okunamayacak şekilde şifreli olduğu görülmüştür.**
- Ağ üzerinde gönderilen dosya, kimlik bilgisi veya hash gibi tüm kritik bilgiler rastgele byte dizileri olarak yer almış; şifresiz/klasik protokollerin aksine saldırganın çözebileceği hiçbir veri taşınmamıştır.

### Karşılaştırmalı Not\*\*

- **FTP, HTTP gibi klasik şifresiz protokollerde** dosyanın içeriği doğrudan ağda gözlemlenebilirken, bu uygulamada bu mümkün olmamıştır.
- Yapılan bu analiz, sistemde uygulanan uçtan uca şifrelemenin gerçek ağ ortamında “tam koruma” sağladığını göstermektedir.

## 3. EKSTRA ÖZELLİKLER VE GELİŞTİRMELER

- Çoklu dosya transferi
- Transfer geçmişi log dosyası
- Resume özelliği (paket kaybı/tıkanıklıkta devam)
- Progress bar, log kutusu ve kullanıcı dostu arayüz
- 3 yanlış şifreyle brute-force koruması
- Protokol seçimi alanı (TCP aktif, UDP demo olarak eklendi)
- Kapsamlı test tablosu ve detaylı rapor

Gelecek çalışmalar için paralel çoklu kullanıcı desteği, dinamik port/protokol geçişi, gelişmiş parola hash'leme, UDP için daha güvenli protokol gibi geliştirmeler yapılabilir.

## 4. SINIRLAMALAR VE İYİLEŞTİRMELER

Bu proje kapsamında gerçekleştirilen güvenli dosya aktarım sistemi, belirlenen hedefler doğrultusunda başarılı şekilde tamamlanmıştır. Ancak uygulamanın ve yapılan testlerin çeşitli sınırlamaları mevcuttur. İlk olarak, ağ üzerindeki saldırı senaryoları ve paket analizleri kontrollü ve sınırlı bir laboratuvar ortamında gerçekleştirilmiştir. Gerçek hayattaki daha karmaşık ağ trafiği ve ileri düzey saldırılar simüle edilmemiştir. Özellikle paket enjeksiyonu ve Ortadaki Adam (MITM) saldırısı, temel düzeyde ele alınmıştır; gerçek dünyada daha sofistike saldırı yöntemleriyle karşılaşılabileceği unutulmamalıdır.

Ayrıca, uygulamanın performans analizleri sadece belirli ağ türlerinde (örneğin, Ethernet, WiFi 5G, ve gecikmeli/kayıplı ağ senaryosu) gerçekleştirilmiştir. Daha farklı ağ ortamlarında, çoklu kullanıcı senaryolarında ve farklı dosya boyutlarında ek testler yapılmamıştır. Kriptografik güvenlik açısından ise, kullanılan şifreleme algoritmasının güncel saldırılara karşı dayanıklılığı gerçek ortamda sürekli olarak takip edilmeli ve gerekirse iyileştirilmelidir.

İyileştirme önerileri kapsamında; sistemin farklı ağ koşullarında ve daha geniş ölçekli ortamlarda test edilmesi, daha kapsamlı saldırı türleriyle karşılaştırılması ve şifreleme algoritmalarının güncellenmesi



önerilmektedir. Ayrıca, kullanıcı dostu bir arayüz ve otomatik saldırı tespiti modüllerinin entegre edilmesi, uygulamanın kullanım kolaylığını ve güvenliğini artıracaktır.

## 5. SONUÇ

Bu projede, güvenli dosya aktarımı için tasarlanan sistemin kurulumu, uygulama geliştirme süreçleri ve güvenlik analizleri detaylı olarak ele alınmıştır. Uygulama kapsamında, şifreli veri iletimi, kullanıcı doğrulama mekanizması ve dosya bütünlüğü kontrolü gibi temel güvenlik prensipleri başarıyla uygulanmıştır. Yapılan senaryo testlerinde; Ethernet, WiFi 5G ve kayıplı/gecikmeli ağ ortamlarında sistemin performansı ölçülmüş, elde edilen sonuçlar grafiksel ve sayısal olarak değerlendirilmiştir.

Ayrıca, Wireshark ile yapılan analizlerde verinin şifreli olarak iletildiği ve dışarıdan erişimin engellendiği doğrulanmıştır. Saldırı senaryoları kapsamında ise, Ortadaki Adam (MITM) ve paket enjeksiyonu gibi temel saldırı türlerine karşı sistemin dayanıklılığı test edilmiştir. Sınırlamalar bölümünde de belirtildiği üzere, uygulamanın daha farklı ağ ve saldırı koşullarında da denenmesi, güvenlik seviyesinin artırılması açısından önem arz etmektedir.

Genel olarak, proje kapsamında geliştirilen sistem, güvenli dosya aktarımı ve ağ trafiğinin korunması konusunda başarılı bir örnek teşkil etmektedir. Elde edilen bulgular, ağ güvenliğinin sağlanmasında bütünlük bir yaklaşımın gerekliliğini ve uygulanan güvenlik önlemlerinin etkisini göstermektedir.

Projenin kaynak koduna [GitHub'dan](#), demo video ve adım adım kurulumuna ise [YouTube'dan](#) ulaşabilirsiniz.

## 6. KAYNAKÇA

- Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.
- The Wireshark Team. (2024). *Wireshark user's guide*. <https://www.wireshark.org/docs/>
- National Institute of Standards and Technology. (2018). *Advanced encryption standard (AES)* (FIPS PUB 197). <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>
- The Python Software Foundation. (2024). *Python 3.12.3 documentation*. <https://docs.python.org/3/>
- The PyCryptodome Project. (2024). *PyCryptodome documentation*. <https://pycryptodome.readthedocs.io/>
- Wang, L., & Jetter, M. (2015). *A bandwidth measurement tool: iPerf3*. In *International Conference on Computing, Networking and Communications (ICNC)* (pp. 423-427). IEEE. <https://iperf.fr/>
- Jagt, J. (2015). *clumsy: network error simulation tool* [Computer software]. <https://github.com/jagt/clumsy>
- Vrem Software Development. (2024). *WiFiAnalyzer [Mobile application software]*. Google Play. <https://play.google.com/store/apps/details?id=com.vrem.wifianalyzer>