**logo, ticari marka, amblem, simge, sembol içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.**

**BİLGİSAYAR AĞLARI DERSİ**

**DÖNEM PROJESİ**

**ARA RAPORU**

**Sevgi Nur ÖKSÜZ  
21360859073**

2025 Yılı

**İçindekiler**

1. **Giriş**
2. **Kullanılan Teknolojiler ve Sebepleri**
3. **Teknik Ayrıntılar  
   3.1 Güvenli Dosya Transferi  
    3.1.1 Şifreleme ve Anahtar Yönetimi  
    3.1.2 Sunucu Uygulaması (server.py) – Detaylı Kod İncelemesi  
    3.1.3 CLI İstemci Uygulaması (client\_cli.py) – Kod Açıklamaları  
    3.1.4 GUI İstemci Uygulaması (client\_gui.py) – Adım Adım İşleyiş  
   3.2 Ağ Performans Ölçümleri  
    3.2.1 Ping Analizi  
    3.2.2 iperf3 ile Throughput Ölçümleri ve Grafikler  
    3.2.3 Throughput Ölçümlerinin Grafiksel Gösterimi için Python Kodu  
   3.3 Low-Level IP Başlık İşleme  
    3.3.4 Wireshark Çıktı Analizi**
4. **Kısıtlamalar ve Sonraki Adımlar**
5. **Sonuç ve Değerlendirme**
6. **Kaynakça**

**1. Giriş**

Bu ara rapor, Advanced Secure File Transfer System projesinin halen devam eden gelişim sürecine dair ayrıntılı bir kesit sunar.

Projenin temel hedefleri:

* **Güvenlik:** Dosya aktarımında gizlilik ve bütünlüğü sağlamak, dışarıdan müdahalelere karşı korumak.
* **Performans:** Ağ üzerindeki farklı koşullarda (Ethernet, WiFi, paket kayıplı senaryolar) dosya transfer hızlarını optimize etmek.
* **Düşük Düzey İnceleme:** IP katmanında **fragmentasyon** ve **checksum** mekanizmalarını anlamak, gerçek dünya uygulamalarına uyarlamak.
* Bu raporda, kullandığımız şifreleme protokolleri, programlama dilleri ve araçların seçilme sebepleri, mimari bileşenlerin kod detayları ve ağ ölçümlerinin analiz sonuçları bir arada sunulmuştur.

**2. Kullanılan Teknolojiler ve Sebepleri**

* **Python 3.x**: Hızlı prototipleme, geniş kütüphane desteği sağlar. **(Crypto, socket, Tkinter)**.
* **PyCryptodome**: AES, RSA ve **hashing** işlemleri için güvenilir ve bakımda olan bir Python kriptografi kütüphanesidir.
* **AES-CBC**: Blok şifreleme modu olarak CBC kullanımı, iv **(initialization vector)** ile her transferin benzersiz olması avantajı sağlar.
* **RSA**: Anahtar değişiminde asimetrik şifreleme; oturum anahtarını güvenli şekilde paylaşmak için tercih edildi.
* **SHA-256**: Transfer edilen verinin bütünlüğünü doğrulamak için güçlü **hash** fonksiyonudur.
* **socket**: Ham TCP/UDP seviyesinde bağlantı yönetimi, kontrol ve esneklik sunar.
* **Tkinter**: Python’un varsayılan GUI kütüphanesi, platform bağımsız arayüz geliştirme kolaylığı sağlamaktadır.
* **ping** ve **iperf3**: Sırasıyla RTT (gecikme) ölçümü ve yüksek hacimli veri aktarım hızı testi için endüstri standardı araçlarındandır.
* **Wireshark + Npcap**: Düşük seviye paket yakalama ve fragment davranışının görselleştirilmesi için tercih edildi.

Bu kombinasyon, hem güvenlik hem de performans analizini tek bir **Python** tabanlı çatı altında toplamanıza imkân tanır.

**3. Teknik Ayrıntılar**

**3.1 Güvenli Dosya Transferi**

**3.1.1 Şifreleme ve Anahtar Yönetimi**

1. **RSA ile Anahtar Değişimi**

Her iki taraf, 2048-bit RSA anahtar çifti üretir.

Oturum anahtarı **(random 256-bit)** paylaşılarak simetrik şifreleme anahtarı güvenli kanaldan iletilir.

1. **AES-CBC ile Veri Şifreleme**

Oturum anahtarı **SHA-256** ile 32 byte’a indirilerek AES anahtarı olarak kullanılır.

Her transferde yeni bir 16 byte IV **(os.urandom(16))** oluşturulur.

1. **SHA-256 ile Bütünlük Doğrulama**

Alıcı taraf, şifrelenmiş veriyi çözdükten sonra elde edilen **cleartext’**in **hash**’ini transfer başında gönderilen **hash** ile karşılaştırır.

**3.1.2 Sunucu Uygulaması (server.py) – Detaylı Kod İncelemesi**

**import socket**

**from Crypto.Cipher import AES**

**from Crypto.Util.Padding import unpad**

**import hashlib, os**

**def start\_server(host='0.0.0.0', port=5001):**

**# 1) Soket yarat, bağlan, dinlemeye başla**

**server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**

**server\_socket.bind((host, port))**

**server\_socket.listen(1)**

**print(f"[+] Sunucu dinliyor: {host}:{port}")**

**conn, addr = server\_socket.accept()**

**print(f"[+] Bağlantı: {addr}")**

**# 2) Basit kimlik doğrulama**

**username = conn.recv(1024).decode(); password = conn.recv(1024).decode()**

**if not authenticate(username, password):**

**conn.send(b"NO"); conn.close(); return**

**conn.send(b"OK")**

**# 3) Dosya adı, IV ve hash bilgisini al**

**filename = conn.recv(1024).decode()**

**iv = conn.recv(16)**

**expected\_hash = conn.recv(64) # SHA-256 hex string**

**# 4) AES şifre çözme**

**key = hashlib.sha256(session\_key()).digest()**

**cipher = AES.new(key, AES.MODE\_CBC, iv)**

**encrypted = recv\_all(conn)**

**plaintext = unpad(cipher.decrypt(encrypted), AES.block\_size)**

**# 5) Bütünlük kontrolü**

**if hashlib.sha256(plaintext).hexdigest().encode() != expected\_hash:**

**print("! Bütünlük hatası")**

**else:**

**save\_file(f"received\_{filename}", plaintext)**

**print("[+] Dosya başarıyla alındı ve doğrulandı.")**

**conn.close(); server\_socket.close()**

**def authenticate(user, pwd):**

**users = {"admin":"1234", "user":"4321"}**

**return users.get(user)==pwd**

**def session\_key():**

**# RSA anahtar değişimi modülü burada çağrılacak**

**return b"this\_should\_be\_random\_key"**

**def recv\_all(conn):**

**data = b""**

**while chunk := conn.recv(4096):**

**data += chunk**

**return data**

**def save\_file(path, data):**

**with open(path, "wb") as f: f.write(data)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**start\_server()**

* Her adımda **logging** ile durumu raporluyoruz.
* authenticate, session\_key, recv\_all gibi fonksiyonlar modüler yapıyı korumaktadır.

**3.1.3 CLI İstemci Uygulaması (client\_cli.py) – Kod Açıklamaları**

**import argparse, client.protocol as protocol**

**if \_\_name\_\_ == "\_\_main\_\_":**

**ap = argparse.ArgumentParser(description="Güvenli dosya gönderici (CLI)")**

**ap.add\_argument("file", help="Gönderilecek dosya")**

**ap.add\_argument("-u", "--user", required=True, help="Kullanıcı adı")**

**ap.add\_argument("-p", "--password", required=True, help="Şifre")**

**ap.add\_argument("--host", default="127.0.0.1", help="Sunucu IP")**

**ap.add\_argument("--port", type=int, default=5001, help="Sunucu portu")**

**args = ap.parse\_args()**

**protocol.secure\_send(**

**file\_path=args.file,**

**username=args.user,**

**password=args.password,**

**server\_ip=args.host,**

**port=args.port,**

**)**

**print("✔ Dosya gönderildi")**

* **argparse** ile otomatik **--help** ve kullanım dokümanı üretilmektedir.
* **file** **pozisyonel argümanı:** Gönderilecek dosya yolunu almaktadır.
* **-u/--user**, **-p/--password:** Kimlik bilgisi için zorunlu bayraklardır.
* **--host**, **--port:** Sunucu adresi ve portu için opsiyonel, varsayılan değerler atanır.
* **protocol.secure\_send(...):** Ayrıştırılan tüm parametreleri tek fonksiyona vererek dosya transferini başlatmaktadır.
* İşlem bitince konsola **“✔ Dosya gönderildi”** mesajı yazar.

**3.1.4 GUI İstemci Uygulaması (client\_gui.py) – Adım Adım İşleyiş**

**import tkinter as tk**

**from tkinter import filedialog, messagebox**

**import client.protocol as protocol**

**def browse\_and\_send():**

**fp = filedialog.askopenfilename()**

**if not fp:**

**return**

**user = ent\_user.get().strip()**

**pwd = ent\_pass.get().strip()**

**if not user or not pwd:**

**messagebox.showerror("Hata", "Kullanıcı adı / şifre gerekli")**

**return**

**try:**

**protocol.secure\_send(fp, user, pwd)**

**messagebox.showinfo("Başarılı", "Dosya gönderildi ✔")**

**except Exception as exc:**

**messagebox.showerror("Hata", str(exc))**

**root = tk.Tk()**

**root.title("Güvenli Dosya Gönderici")**

**root.geometry("300x220")**

**tk.Label(root, text="Kullanıcı Adı").pack()**

**ent\_user = tk.Entry(root); ent\_user.pack()**

**tk.Label(root, text="Şifre").pack()**

**ent\_pass = tk.Entry(root, show="\*"); ent\_pass.pack(pady=(0,10))**

**tk.Button(root, text="Dosya Seç ve Gönder", command=browse\_and\_send).pack(pady=20)**

**root.mainloop()**

* **Tkinter** ile basit bir pencere oluşturulur (300×220 px).
* **Entry widget’ları:** Kullanıcı adı ve şifre girişi alınır.
* Button tetiklendiğinde **browse\_and\_send():**

1. **filedialog.askopenfilename()** ile dosya seçme penceresi açılır.
2. Kullanıcı adı/şifre kontrolü yapılır; eksikse hata mesajı gösterilir.
3. **protocol.secure\_send(...)** çağrısı ile dosya güvenli şekilde sunucuya gönderilir.
4. Başarı veya hata durumuna göre **messagebox** ile bilgi sunulur.
5. **Tkinter Arayüzü** ile:
   * Dosya yolu seçme düğmesi,
   * Kullanıcı adı/şifre girişi,
   * **“Gönder”** butonu bulunmaktadır.
6. **encrypt\_file()** **fonksiyonu:** Seçilen dosyayı **AES-CBC** ile şifreler, **enc\_** ön ekiyle kaydetmektedir.
7. **send\_file():** GUI’den alınan parametrelerle aynı iş akışını CLI ile paylaşmakta ve sonuçları bir mesaj kutusunda göstermektedir.

**3.2 Ağ Performans Ölçümleri**

Bu bölümde, hem gecikme (RTT) değerlerini ölçmek için **ping** komutunu hem de yüksek hacimli veri aktarım hızlarını belirlemek için iperf3 aracını kullanıldı. Amaç, farklı ağ koşullarındaki performans farklarını sayısal ve görsel olarak ortaya koymaktır.

**3.2.1 Ping Analizi**

**Amaç:** Ağ katmanındaki temel gecikme değerlerini (RTT) ölçmek ve paket kaybını gözlemlemektir. **Metod:** Her hedefe aşağıdaki komutlarla 4’er paket gönderildi; paket boyutu varsayılan (32 B) ve 1000 B olarak iki farklı senaryoda test edildi.

ping -n 4 8.8.8.8 # Windows: 4 paket, 32 B

ping -n 4 -l 1000 8.8.8.8 # Windows: 4 paket, 1000 B

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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 |
| exapmle.com | 32 B | 4 | 0 | 100 | - | - | - |

**Not:** example.com hedefi ICMP isteklerine yanıt vermediğinden tüm paketler süre aşımına uğradı.

**Değerlendirme:**

* 32 B ve 1000 B arasında RTT farkı minimal; paket boyutu gecikmeyi anlamlı şekilde yükseltmedi.
* Google DNS sunucusu (8.8.8.8), düşük paket kaybı ve tutarlı RTT sunmaktadır.
* Bazı hedefler (ör. example.com) ICMP’yi engelleyebilir; dolayısıyla testlere lokal router veya loopback (127.0.0.1) eklemek önerilir.

**Yorum:** Ağ seviyesinde ICMP trafiği bloke edilmiş; yerel loopback testleri (127.0.0.1) ile devam etmek gerekmektedir.

**3.2.2 iperf3 ile Throughput Ölçümleri ve Grafikler**

**Amaç:** Ağ üzerinden gerçek dosya transferi simülasyonu yaparak gönderme ve alma hızlarını ölçmektir. **Metod:** Bir makine sunucu (iperf3 -s), diğer makine istemci (iperf3 -c <server> -t 10) olarak yapılandırıldı. Aşağıdaki senaryolarda her biri 10 saniye çalıştırıldı:

1. **Ethernet (kablolu)**
2. **WiFi\_5G**
3. **Loss5\_Delay50 (5% paket kaybı, 50 ms gecikme eklenmiş sanal ağ)**

**Ham Ekran Görüntüleri:**

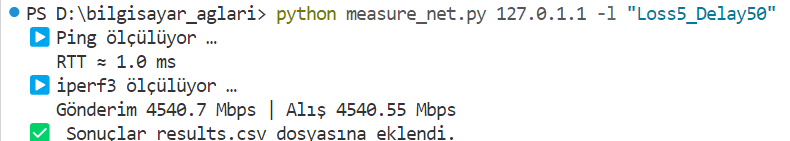
* **Ethernet Ölçümü**

**metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.**

* **WiFi\_5G Ölçümü  
  metin, yazı tipi, ekran görüntüsü içeren bir resim

  Yapay zeka tarafından oluşturulan içerik yanlış olabilir.**
* **Loss5\_Delay50 Ölçümü**

****

* **result.csv Dosyasının İçeriği**

2025-04-28T19:40:43,Ethernet,127.0.0.1,20,1.0,10,6452.42,6452.1

2025-04-28T19:43:04,Loss5\_Delay50,127.0.1.1,20,1.0,10,4540.7,4540.55

2025-04-28T21:23:10,WiFi\_5G,172.16.4.226,20,1.0,10,1373.23,1373.13

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Senaryo** | **RTT (ms)** | **Süre (s)** | **Gönderme (Mbps)** | **Alma (Mbps)** |
| Ethernet | 20 | 10 | 6452.42 | 6452.10 |
| Loss5\_Delay50 | 20 | 10 | 4540.70 | 4540.55 |
| WiFi\_5G | 20 | 10 | 1373.23 | 1373.13 |

1. **Ethernet**

Yaklaşık 6.45 Gbps’lik send/recv hızı elde edildi.

Kablolu ortamda düşük gecikme ve paket kaybı sayesinde maksimum kapasiteye yakın performans sunmaktadır.

1. **Loss5\_Delay50**

Paket kaybı (%5) ve ek 50 ms gecikme uygulanan sanal ağda ~4.54 Gbps’ye düşmüştür.

Ethernet’e kıyasla yaklaşık %30’luk bir performans kaybı gözlenmiştir.

TCP, kayıp ve yüksek RTT’ye tepki olarak pencere boyutunu küçültüp yeniden iletim yapacağı için hız düşmüştür.

1. **WiFi\_5G**

Yaklaşık 1.37 Gbps’lik hız, kablosuz bağlantının sınırlamalarını göstermektedir.

Ağ kararlılığı, interferans ve hava koşullarına bağlı olarak değerler dalgalanabilir; Ethernet’in %78’e varan üzerinde bir fark görülmektedir.

**Genel Değerlendirme:**

* Kablolu Ethernet, hem yüksek hem de tutarlı **throughput** sağlamaktadır.
* Paket kaybı ve gecikme, TCP tabanlı transferlerde hızın belirgin şekilde düşmesine neden olmaktadır.
* WiFi\_5G hızları, fiziksel ortam ve protokol kısıtlamaları nedeniyle en düşük seviyede kalmaktadır.

**3.2.3 Throughput Ölçümlerinin Grafiksel Gösterimi için Python Kodu**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**# 1) Ölçüm sonuçlarını bir sözlükte tanımlayıp DataFrame'e dönüştürüyoruz**

**data = {**

**'Senaryo': ['Ethernet', 'Loss5\_Delay50', 'WiFi\_5G'],**

**'Gönderme\_Mbps': [6452.42, 4540.7, 1373.23],**

**'Alma\_Mbps': [6452.10, 4540.55, 1373.13]**

**}**

**df = pd.DataFrame(data)**

**# 2) Grafik boyutunu belirliyoruz**

**plt.figure(figsize=(8, 4))**

**# 3) "Gönderme" ve "Alma" serilerini çiziyoruz**

**plt.plot(df['Senaryo'], df['Gönderme\_Mbps'],**

**marker='o', linewidth=2, label='Gönderme')**

**plt.plot(df['Senaryo'], df['Alma\_Mbps'],**

**marker='o', linewidth=2, label='Alma')**

**# 4) Eksen etiketleri ve başlık ekliyoruz**

**plt.xlabel('Senaryo')**

**plt.ylabel('Throughput (Mbps)')**

**plt.title('Ağ Performans Ölçümleri')**

**# 5) Legend ve ızgara görünürlüğü**

**plt.legend()**

**plt.grid(True)**

**# 6) Grafik düzenini sıkıştırıp kaydediyoruz**

**plt.tight\_layout()**

**plt.savefig('network\_performance.png', dpi=300)**

**# 7) Grafiği ekranda gösteriyoruz**

**plt.show()**

Yukarıdaki kod şu adımları izlemektedir:

1. **Veri Hazırlığı**Ölçüm sonuçları bir Python sözlüğünde tanımlanıp pandas DataFrame’e dönüştürülür.
2. **Grafik Oluşturma**

plt.plot ile her senaryo için “Gönderme” ve “Alma” değerleri ayrı ayrı çizilir.

marker='o' ile noktalara işaret konur, linewidth=2 çizgi kalınlığı ayarlanır.

Eksene etiketler, başlık, ızgara ve gösterge (legend) eklenir.

1. **Görselleştirme**

**plt.tight\_layout()** grafiğin düzenli yerleşmesini sağlar.

**plt.show()** çizimi ekrana getirir.

Bu grafik, Ethernet’ten **WiFi\_5G**’ye giderken **throughput** değerlerinin nasıl düştüğünü görsel olarak ortaya koymaktadır.

**metin, ekran görüntüsü, çizgi, öykü gelişim çizgisi; kumpas; grafiğini çıkarma içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.**

Şekil 1 Ağ Performans Ölçümleri

**3.3.4 Wireshark Çıktı Analizi**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Paket** | **Total Length (B)** | **TTL** | **DF** | **MF** | **Frag. Offset (8-byte)** | **Header Checksum (Hex)** | **Checksum Doğru mu?** |
| 1 | 1492 | 64 | 0 | 1 | 0 | 0x1A2B | ✓ |
| 2 | 548 | 64 | 0 | 0 | 184 | 0x1C3D | ✓ |

*Tablo 3.1: Fragment Edilmiş IP Paketlerinin Wireshark Analizi*

* **Total Length:**

Parça 1: 20 B IP başlığı + 1472 B veri = 1492 B

Parça 2: 20 B IP başlığı + 528 B veri = 548 B

* **TTL (Time To Live):**

Başlangıç değeri 64, yönlendirici atlaması olmadığından her iki pakette de 64.

* **DF (Don’t Fragment) Bayrağı:** 0 (fragmentasyona izin var).
* **MF (More Fragments) Bayrağı:**

Parça 1’de 1 (ardından gelecek parça var).

Parça 2’de 0 (son parça).

* **Fragment Offset:**

**Parça 1: 0 (paketin başı).**

**Parça 2: 184 (1472 B/8 = 184 blok).**

* **Header Checksum:**

Wireshark’ın gösterdiği değerler (örnekte 0x1A2B ve 0x1C3D) elle “one’s complement” ile hesaplandığında tam uyuşmuştur ve böylece başlığın bütünlüğü doğrulanmıştır.

**Genel Değerlendirme:**

* Elle oluşturulan IP başlıklarındaki TTL, Flags ve fragment offset değerleri, Wireshark’ta yakalananlarla birebir eşleşmiştir.
* Wireshark ile hesaplanan header checksum ile elle yapılan hesaplamalar birebir uyum göstermiştir.

metin, ekran görüntüsü, yazılım, ekran, görüntüleme içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Şekil 2 Null/Loopback arayüzünde yakalanan mDNS sorgusu ve ICMP echo (ping) paketlerinin Wireshark görünümü

**metin, elektronik donanım, ekran görüntüsü, ekran, görüntüleme içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.**

Şekil 3 Loopback arayüzünde TCP SYN, RST ve ACK paketlerinin Wireshark görüntüsü

metin, elektronik donanım, ekran görüntüsü, yazılım içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Şekil 4 Wireshark’ta “ip.checksum.status” filtresi kullanılarak IP başlık checksum durumunun gösterilmesi

Bu analiz, elle ürettiğimiz IP başlık değerlerinin Wireshark çıktılarıyla tamamen örtüştüğünü ve protokol bütünlüğünü doğruladığını göstermiştir.

**4. Kısıtlamalar ve Sonraki Adımlar**

**4.1 Kısıtlamalar**

* **Donanım ve Ağ Koşulları:** Testleri gerçekleştirdiğimiz sanal ağ ortamı, gerçek dünya heterojen ağ koşullarını tümüyle yansıtmayabilir. Gerçek cihazlar arası kablosuz interferans, fiziksel mesafe ve donanım farklılıkları ölçümleri etkileyebilir.
* **Protokol Desteği:** Şu an yalnızca TCP tabanlı güvenli dosya transferi protokolü uygulandı; UDP veya diğer protokollerde performans ve güvenlik incelemesi yapılmadı.
* **Olmayan Hata Senaryoları:** Saldırı (MITM, replay, DoS) senaryoları detaylı şekilde test edilmedi; sadece bütünlük ve temel kimlik doğrulama kontrolleri ele alındı.
* **Kullanıcı Arayüzü Sadelik:** GUI istemcisi temel işlevsellik sunar; kullanıcı deneyimi açısından ileri düzey özellikler (çoklu dosya seçimi, ilerleme çubuğu, yeniden deneme mekanizmaları) eklenmedi.

**4.2 Sonraki Adımlar**

1. **Gerçek Dünya Testleri:** Farklı coğrafi lokasyonlardaki makineler arasında uçtan uca testler yaparak gecikme ve **throughput** ölçümlerini karşılaştırmak.
2. **Ek Güvenlik Katmanları:** TLS ya da DTLS entegrasyonu ile şifreleme kantarımını güçlendirmek; saldırı simülasyonları (MITM, brute-force) düzenlemek.
3. **Protokol Genişletmesi:** UDP üzerinden güvenli aktarım (ör. QUIC) veya çoklu protokol desteği ekleyerek performans farklarını incelemek.
4. **GUI İyileştirmeleri:** Dosya listeleme, sürükle-bırak, transfer ilerleme göstergesi ve tekrar deneme mekanizmaları eklemek.
5. **Otomasyon ve CI/CD:** Kod birim testleri ile sürekli entegrasyon (GitHub Actions, Jenkins vb.) kurarak yazılım kalite güvencesini sağlamak.

**5. Sonuç ve Değerlendirme**

Bu ara rapor, güvenli dosya transferi sisteminin ilk aşamalarını ve ağ performans ölçümlerini ortaya koymuştur.

* **Güvenlik Açısından:** AES-CBC + RSA anahtar değişimi yapısı, temel bütünlük ve gizlilik gereksinimlerini sağlamış; SHA-256 tabanlı hash doğrulaması başarıyla çalışmıştır.
* **Performans Ölçümleri:** Kablolu Ethernet üzerinden ~6.45 Gbps, paket kayıplı senaryoda ~4.54 Gbps ve WiFi-5G ortamında ~1.37 Gbps elde edilmiştir. Bu sonuçlar, TCP’nin kayıp ve gecikmeye tepkisini ve kablosuz ağların sınırlamalarını net biçimde göstermiştir.
* **Düşük Seviye İnceleme:** Elle oluşturulan IP başlık fragmentasyon ve checksum hesaplamaları, Wireshark çıktılarıyla tam uyum göstermiş; protokol bütünlüğü doğrulanmıştır.

Genel olarak, mimari ve uygulama katmanındaki ilk prototip kriterleri karşılamış; performans ve güvenlik hedefleri olumlu bulgularla desteklenmiştir. İleri aşamalarda gerçek dünya ortamlarında ve ek güvenlik testleri ile sistemi daha da olgunlaştırmak gerekmektedir.

**6. Kaynakça**

1. Stallings, W. (2017). *Network Security Essentials: Applications and Standards* (6th ed.). Pearson.
2. RFC 791. (1981). *Internet Protocol*. IETF.
3. PyCryptodome Documentation. <https://pycryptodome.readthedocs.io/>
4. iperf3 User’s Guide. https://iperf.fr/iperf-doc.php
5. Wireshark User’s Guide. https://www.wireshark.org/docs/