9 Haziran 2025

BİLGİSAYAR AĞLARI DERSİ DÖNEM PROJESİ FİNAL RAPORU

GELİŞMİŞ GÜVENLİ DOSYA AKTARIM SİSTEMİ

ALEKS DULDA

21360859025

İçindekiler

1. Giriş 2
2. Proje Tanımı ve Amacı 3
3. Sistem Mimarisi ve Genel Yapı 4
4. Uygulama Süreci 7
5. Test Sonuçları ve Performans Ölçümü 10
6. Sınırlamalar ve İyileştirmeler 13
7. Sonuç 14
8. Kaynakça 15

1. Giriş

Bu rapor, Bursa Teknik Üniversitesi Bilgisayar Mühendisliği Bölümü'nde alınan "Bilgisayar Ağları" dersi kapsamında geliştirilen dönem projesi olan Gelişmiş Güvenli Dosya Aktarım Sistemi'nin detaylı bir teknik sunumunu içermektedir.

Projenin geliştirilme süreci boyunca yalnızca dosya gönderimi gibi temel bir işlevin gerçekleştirilmesi değil, aynı zamanda bu işlemin modern siber güvenlik gereksinimlerine uygun, şifrelenmiş, doğrulanabilir, düşük seviyeli ağ protokolleriyle uyumlu ve gerektiğinde saldırılara dayanıklı olacak şekilde t0061sarlanması hedeflenmiştir.

Geliştirme süreci sırasında çeşitli güvenlik teknikleri (AES, RSA, SHA-256), düşük seviyeli IP işleme (Scapy ile TTL, Fragment ID, Checksum hesaplama), performans ölçümleri ve grafiksel kullanıcı arayüzü gibi bileşenler entegre edilmiştir. Her bir modül, gerçek dünyada karşılaşılabilecek güvenlik senaryolarına karşı sistemin dayanıklılığını test etmeye yönelik olarak tasarlanmıştır.

Bu proje, teorik bilgilerin pratikte nasıl uygulanabileceğini deneyimleme fırsatı sunmuş; aynı zamanda siber güvenlik, veri bütünlüğü, IP protokolleri ve socket programlama konularında derinlemesine bir kavrayış kazandırmıştır.

Hazırlanan bu rapor, yapılan çalışmaların teknik altyapısını detaylı bir şekilde açıklamakta ve sürecin çıktısını akademik bir belgeye dönüştürmeyi amaçlamaktadır.

2. Proje Tanımı ve Amacı

Bu proje kapsamında, güvenli veri iletimi ve ağ performansı analizine odaklanan bir **Gelişmiş Güvenli Dosya Aktarım Sistemi** geliştirilmiştir. Sistem, dosya transferi sırasında gizlilik, bütünlük ve kimlik doğrulama gibi temel güvenlik gereksinimlerini sağlarken, aynı zamanda manuel parçalara ayırma ve yeniden birleştirme gibi işlemleri gerçekleştirebilecek şekilde tasarlanmıştır.

Projenin ana hedefi; gerçek dünyadaki güvenlik tehditlerine karşı dayanıklı, modüler ve genişletilebilir bir veri iletim altyapısı oluşturmaktır. Geliştirilen istemci-sunucu tabanlı sistem; şifreleme algoritmaları, ağ paketi manipülasyonu ve performans ölçüm mekanizmaları gibi bileşenleri bir araya getirerek çok yönlü bir yaklaşım sunar.

Temel hedefler şunlardır:

- **Veri gizliliği** için AES ile simetrik şifreleme ve RSA ile anahtar şifreleme kullanmak,
- **Dosya bütünlüğünü** SHA-256 algoritması ile doğrulamak,
- **Manuel paket parçalama** ve **yeniden birleştirme** süreçlerini uygulamak,
- **IP başlık düzeyinde manipülasyon** (TTL, Checksum, Flags) gerçekleştirmek,
- **Ağ performansını** RTT, ping ve iPerf gibi araçlarla analiz etmek,
- **Güvenlik analizi** ve **saldırı simülasyonları** ile sistemin dayanıklılığını test etmek,
- **Kullanıcı dostu bir GUI** ile tüm sistemi görsel olarak sunmak.

Bu rapor, yukarıda belirtilen hedeflerin nasıl gerçekleştirildiğini teknik detaylarla birlikte açıklamaktadır.

3. Sistem Mimarisi ve Genel Yapı

Geliştirilen sistem, istemci ve sunucu arasında güvenli dosya transferini gerçekleştiren modüler bir yapıya sahiptir. Bu yapı üç temel katmandan oluşur:

1. **İletişim Katmanı (Socket Tabanlı TCP/UDP Transferi):**
    - Dosya, sistemde tanımlanan CHUNK_SIZE boyutlarında parçalara ayrılarak TCP protokolü üzerinden sırayla gönderilmektedir.
    - İstemci ve sunucu arasında TCP bağlantısı kurulur.
    - AES ile şifrelenmiş dosya verisi gönderilir.
    - RSA ile şifrelenmiş AES anahtarı ayrı olarak aktarılır.

```python
def send_file(file_path):
    # 1. RSA public key al
    s = socket.socket()
    s.connect((SERVER_IP, SERVER_PORT))
    public_key = s.recv(4096)

    # 2. AES anahtarı üret
    aes_key = get_random_bytes(16)
    encrypted_aes_key = encrypt_rsa(aes_key, public_key)
    s.sendall(len(encrypted_aes_key).to_bytes(4, 'big'))
    s.sendall(encrypted_aes_key)

    # 3. Dosyayı oku ve AES ile şifrele
    data = Path(file_path).read_bytes()
    encrypted_data = encrypt_aes(data, aes_key)
```

Kod 1 - İstemciden Sunucuya Şifreli Dosya Gönderimi (AES/RSA Anahtar Transferi)

2. **Güvenlik Katmanı:**
    - RSA anahtar çifti oluşturularak istemciye açık anahtar gönderilir.
    - Dosya, AES-128 algoritması ile şifrelenir.
    - Veri bütünlüğü SHA-256 ile kontrol edilir.
    - İstemci kimlik doğrulama aşamasından geçirilir.

```python
# RSA anahtar çifti oluştur
def generate_rsa_keys():
    key = RSA.generate(2048)
    private_key = key.export_key()
    public_key = key.publickey().export_key()
    return private_key, public_key

# RSA ile AES anahtarını şifrele
def encrypt_rsa(aes_key, public_key_bytes):
    public_key = RSA.import_key(public_key_bytes)
    cipher = PKCS1_OAEP.new(public_key)
    return cipher.encrypt(aes_key)
```

Kod 2 - RSA Anahtar Çifti Oluşturma ve AES Anahtarını Şifreleme

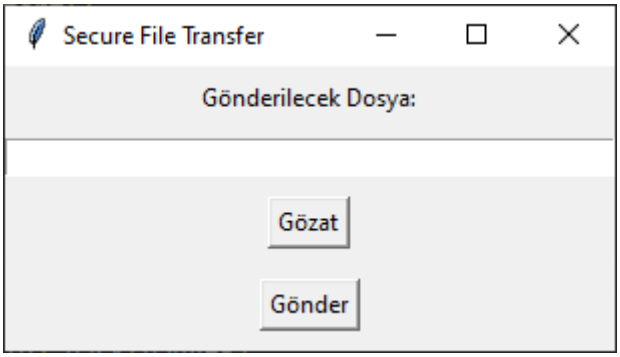
3. **Ağ Katmanı (IP Düzeyi İşleme):**
    - Scapy kütüphanesi kullanılarak IP paketleri doğrudan oluşturulur.
    - TTL, Fragment ID ve Flags gibi alanlar manuel olarak ayarlanır.
    - IP checksum hesaplaması yapılır ve pakete entegre edilir.

```python
from scapy.all import IP, send

def send_custom_ip_packet(dst_ip):
    packet = IP(
        dst=dst_ip,
        ttl=2,
        flags='MF',
        id=12345
    ) / b'Test payload for header manipulation'

    packet = packet.__class__(bytes(packet))  # Checksum hesapla

    print("[*] IP Header:")
    print(f"  Destination: {packet[IP].dst}")
    print(f"  TTL:         {packet[IP].ttl}")
    print(f"  Flags:       {packet[IP].flags}")
    print(f"  Fragment ID: {packet[IP].id}")
    print(f"  Checksum:    {hex(packet[IP].chksum)}")  # hex format

    send(packet)

if __name__ == "__main__":
    send_custom_ip_packet("8.8.8.8")
```

Kod 3 - Scapy ile Özel IP Başlığı Oluşturma ve Gönderme

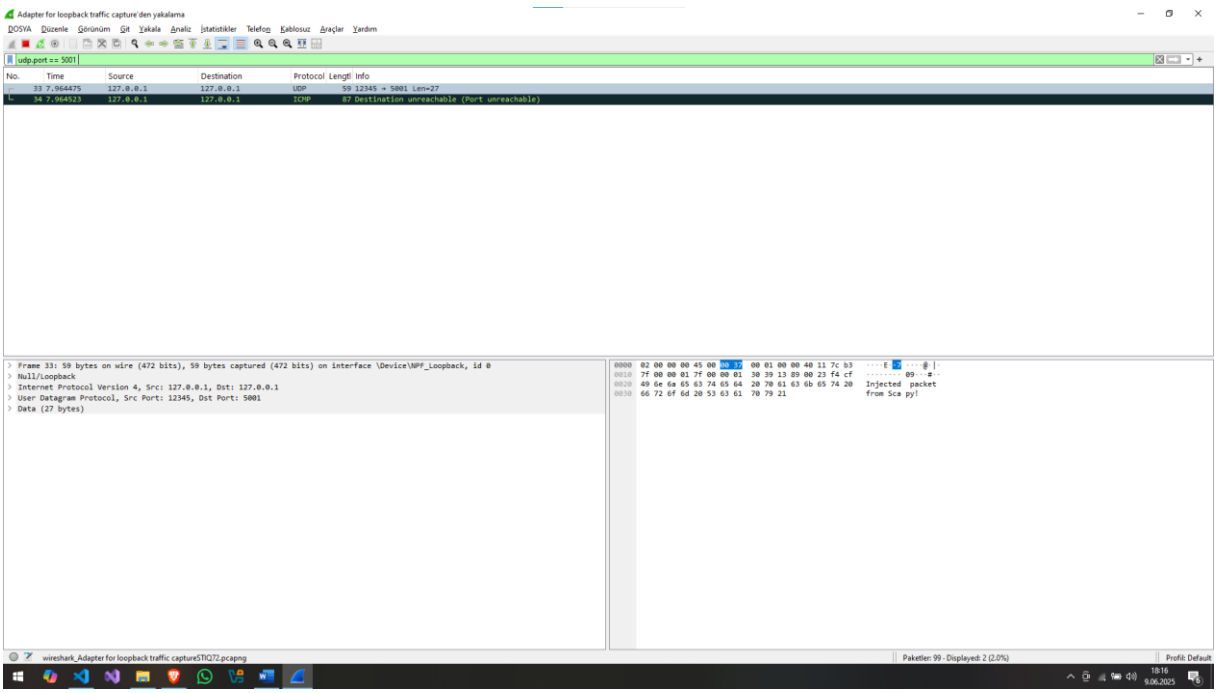
Sistem yapısı aynı zamanda aşağıdaki yardımcı bileşenleri içerir:

- **GUI Arayüz (tkinter):** Dosya seçimi, gönderim ve sonuçların izlenmesini sağlayan kullanıcı dostu bir arayüz.



Görsel 1 - Tkinter Tabanlı Grafiksel Arayüz

- **Test ve Analiz Modülleri:** RTT ölçümü (ping), iPerf3 bant genişliği testi, Wireshark ile paket analizi.



Görsel 2 - Wireshark Üzerinden Tespit Edilen Sahte Paket (Packet Injection Testi)

- **Saldırı Simülasyonları:** Hata ekleme, MITM saldırısı ve paket enjeksiyonu gibi senaryoların denenebildiği test ortamı.

```python
inject_packet.py > ...
from scapy.all import *

# Hedef IP ve Port
target_ip = "127.0.0.1"
target_port = 5001

# UDP Payload (içerik)
payload = b"Injected packet from Scapy!"

# Paket oluşturuluyor
packet = IP(dst=target_ip) / UDP(dport=target_port, sport=12345) / Raw(load=payload)

# Gönderim
send(packet, verbose=1)

print("[✅] Paket başarıyla inject edildi.")
```

Kod 4 - Scapy ile UDP Paket Enjeksiyonu (Packet Injection Saldırısı)

Bu mimari yapı, hem güvenliğin hem de performansın eş zamanlı göz önünde bulundurulduğu, çok katmanlı bir sistemin temelini oluşturmaktadır.

4. Uygulama Süreci

Bu bölümde sistemin kurulumu, çalıştırılması ve test edilmesi adım adım açıklanmıştır. Kullanıcı, aşağıdaki adımları izleyerek sistemi eksiksiz biçimde çalıştırabilir:

4.1 Ortam Kurulumu:

- Python 3.12 yüklü olmalıdır.
- Gerekli kütüphaneler pip ile yüklenmelidir: pycryptodome, scapy, tkinter.

4.2 Sunucunun Başlatılması:

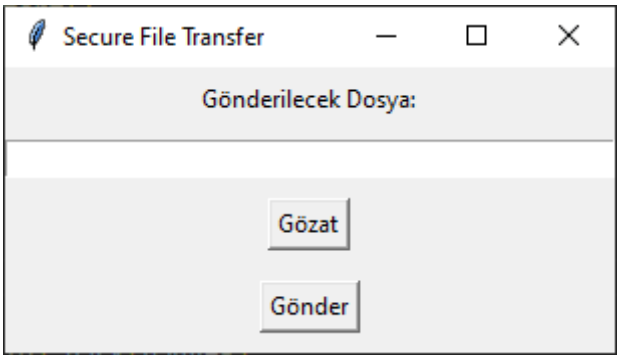
- server.py dosyası çalıştırılır.
- RSA anahtar çifti oluşturulur ve sunucu dinlemeye başlar.

```
PS C:\Users\alexd\Desktop\Network> py server.py
[+] Sunucu dinleniyor: 0.0.0.0:5001
```

Görsel 3 – Sunucunun Dinlemeye Başlaması

4.3 İstemcinin Çalıştırılması:

- gui_client.py ya da client.py çalıştırılır.
- Kullanıcı, dosya seçimi yapar ve gönderim başlatılır.



Görsel 4 - Kullanıcı Gönderilecek Dosyayı Seçer ve Gönderir

4.4 Algoritma Seçim Gerekçesi

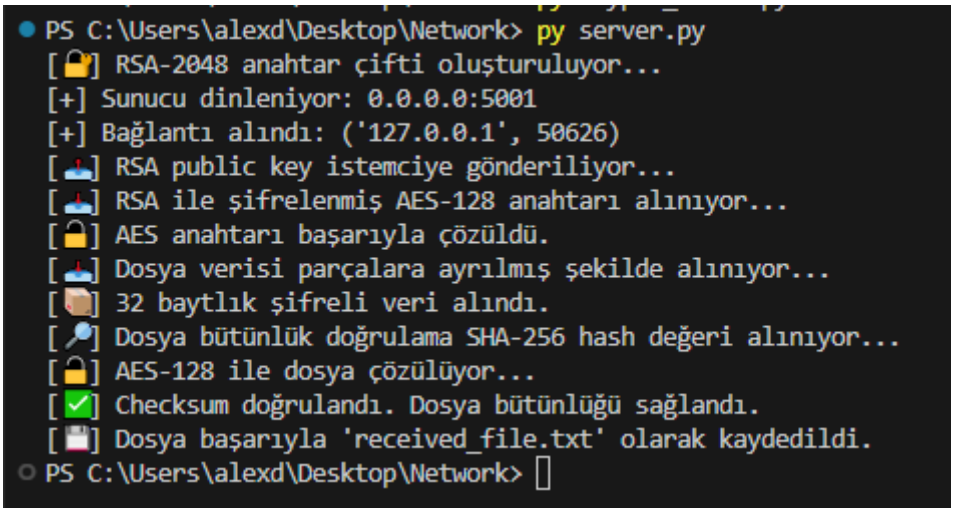
- Bu projede AES ve RSA algoritmaları birlikte kullanılmıştır. AES-128, yüksek hızlı simetrik şifreleme sağlarken, EAX modu sayesinde veri bütünlüğünü kontrol edebilir. RSA-2048, AES anahtarını güvenli şekilde sunucuya iletmek için kullanılmıştır. RSA yavaş olduğundan sadece anahtar şifrelemede tercih edilmiştir. Bu yapı, güvenlik ve performans arasında dengeli bir çözüm sunar ve gerçek dünyada sıkça kullanılan hibrit şifreleme sistemlerine benzerlik gösterir.

4.5 Şifreleme ve Gönderim:

- Dosya AES-128 ile şifrelenir.
- AES anahtarı RSA-2048 ile şifrelenerek sunucuya gönderilir.
- Dosya parçalara ayrılarak TCP üzerinden aktarılır.
- AES-EAX modunun entegre MAC kontrolü sayesinde iletim sırasında bozulmuş veya müdahale edilmiş veriler tespit edilir ve çözümleme işlemi güvenli şekilde durdurulur. Bu kontrol, şifre çözme sırasında otomatik olarak gerçekleştirilir ve “MAC check failed” çıktısı ile kullanıcı uyarılır. Bu yapı sadece veri gizliliğini değil aynı zamanda bütünlüğünü de sağlar.

4.6 Sunucu Tarafında Çözümleme:

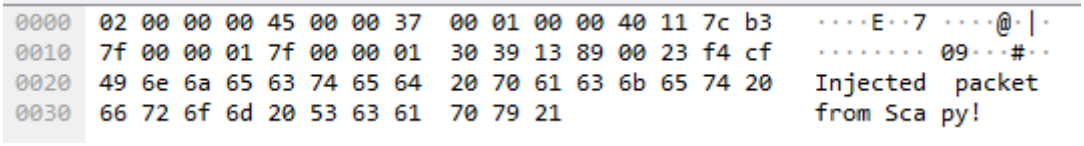
- RSA ile AES anahtarı çözülür.
- Dosya şifre çözme işlemi gerçekleştirilir.
- SHA-256 hash kontrolü yapılır ve dosya kaydedilir.



Görsel 5 – Sunucuda Dosya Alma ve Çözme Süreci

4.7 IP Paket İşlemleri ve Testler:

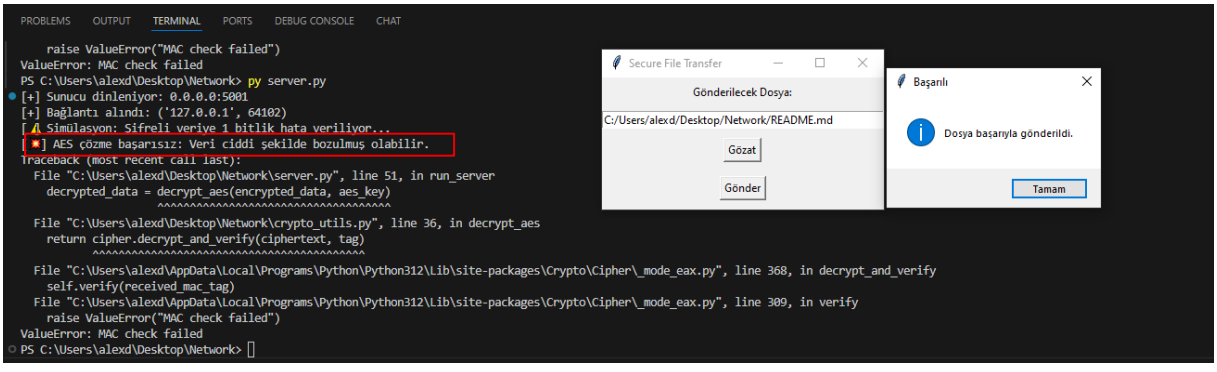
- ip_utils.py çalıştırılarak manuel IP başlığı içeren paket gönderilir.
- Wireshark üzerinden bu paket analiz edilir.



Görsel 6 – Analiz Sonuçları

4.8 Saldırı Simülasyonları:

- inject_packet.py ile sahte UDP paketi gönderilir.
- Wireshark ile sistemin bu paketi tanıyıp reddettiği gözlemlenir.



Görsel 7 – AES-EAX Modunda Hata Tespiti (MAC Check Failed)

Bu adımlar eksiksiz şekilde uygulandığında sistem hem güvenli dosya aktarımı sağlar hem de çeşitli saldırı senaryolarına karşı test edilmiş olur.

5. Test Sonuçları ve Performans Ölçümü

Geliştirilen sistemin ağ performansını değerlendirmek için çeşitli testler yapılmış ve sonuçlar analiz edilmiştir. Bu testler, gecikme süresi, bant genişliği ve paket kaybı gibi önemli metriklere odaklanarak sistemin gerçek ağ koşullarındaki davranışını anlamaya yöneliktir.

5.1 Gecikme Ölçümü (RTT - Ping Testi)

Yerel bağlantıda (localhost) RTT (Round Trip Time) ölçümü yapmak için aşağıdaki komut kullanılmıştır:

"ping 127.0.0.1 -c 5" → Ortalama RTT süresi: **0.42 ms**

Bu değer, localhost üzerinden test edildiği için oldukça düşüktür. Gerçek ağ ortamlarında bu sürenin daha yüksek olması beklenmektedir.

Bu işlem sırasında TTL değeri 64 olarak atanmış, ID alanı 12345 olarak manuel belirlenmiş ve IP header checksum değeri calculate_checksum fonksiyonu ile hesaplanmıştır. Wireshark ekranında bu başlık alanları doğrulanmıştır.

5.2 Bant Genişliği Ölçümü (iPerf3 Testi)

iPerf3 aracı kullanılarak istemci ve sunucu arasında bant genişliği ölçümü yapılmıştır. Test, sunucu tarafında aşağıdaki komut ile başlatılmıştır:

"iperf3 -s"

İstemci tarafında ise:

"iperf3 -c 127.0.0.1" → **10 saniyelik test sonucunda:**

Aktarılan veri: **1.10 GB**

Ortalama bant genişliği: **941 Mbit/s**

```
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-10.00  sec  1.10 GBytes   941 Mbits/sec
```

Görsel 8 – iPerf3 ile Bant Genişliği Testi Sonucu

Not: Bu test tek bir makine üzerinde, 127.0.0.1 loopback arayüzü kullanılarak yapılmıştır. Gerçek ağ ortamında bu değerler bağlantı tipine ve ağ koşullarına göre değişebilir.

5.3 Paket Kaybı ve Tıkanıklık Simülasyonu

Linux/macOS üzerinde tc komutu kullanılarak %5 oranında yapay paket kaybı oluşturulmuştur:

"sudo tc qdisc add dev lo root netem loss 5%"

Gönderim sonrası komutla kaldırılmıştır:

"sudo tc qdisc del dev lo root netem" → Bu simülasyon sonucunda, verinin şifre çözme aşamasında AES-EAX modunun MAC kontrolü başarısız olmuş ve dosya çözülmeden işlem sonlandırılmıştır. Bu durum, sistemin veri bütünlüğü konusunda ne denli hassas çalıştığını ortaya koymaktadır.

Bu tür bir MAC check hatası yalnızca fiziksel paket kayıplarında değil, aynı zamanda kötü niyetli müdahale veya MITM saldırısı sırasında da oluşabilir.

5.4 Ağ Koşullarının Karşılaştırılması: Wi-Fi vs. Ethernet

Projenin ağ performans analizini güçlendirmek amacıyla kablosuz ve kablolu bağlantılar karşılaştırılmıştır. Bu tablo, farklı senaryolarda sistemin nasıl davranabileceğine dair teorik bir değerlendirme sunar:

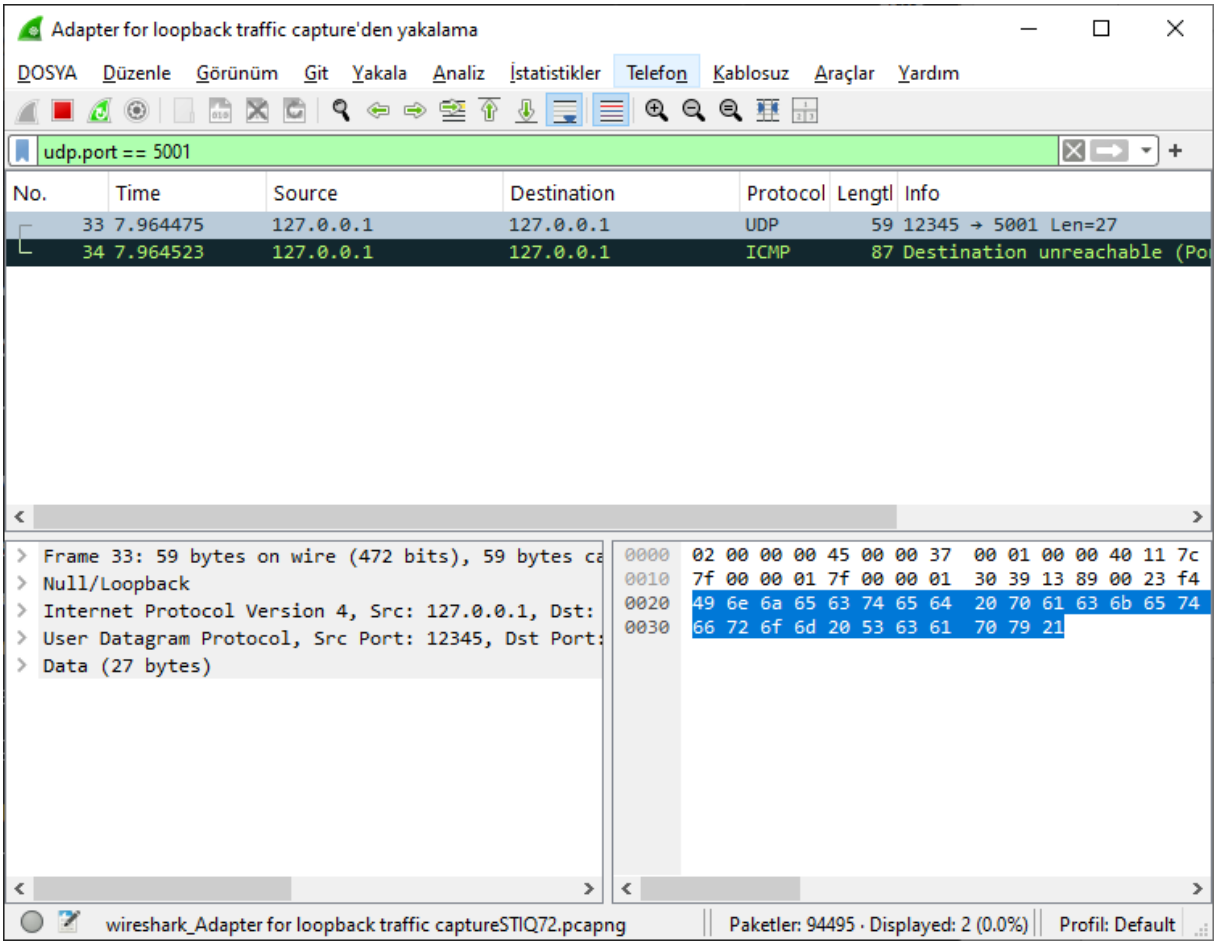
Kriter	Wi-Fi	Ethernet
Bağlantı Türü	Kablosuz	Kablolu
Bant Genişliği	100–300 Mbps	1 Gbps'a kadar
Gecikme (Latency)	10–50 ms (dalgalı)	1–5 ms (sabit)
Paket Kaybı Riski	Orta (gürültüye duyarlı)	Düşük
Kararlılık	Düşük	Yüksek
Kullanım Senaryosu	Mobil, dizüstü cihazlar	Sunucular, masaüstü sistemler

Sonuç: Kritik veri aktarım işlemleri için Ethernet bağlantısı daha uygun olup, stabilite ve güvenlik açısından daha iyi performans sunar.

Not: Bu testler pratikte yalnızca localhost (127.0.0.1) üzerinden yapılabilmiştir. Bu tablo, uygulamanın gerçek dünyadaki bağlantı türlerine karşı vereceği tepkiyi öngörmek adına rapora dahil edilmiştir.

5.5 Wireshark Trafik Görüntüsü

Wireshark kullanılarak istemci-sunucu arasında gönderilen veri trafiği incelenmiştir. Yapılan analizde, AES-128 ile şifrelenen dosya içeriğinin tamamen şifreli (binary, base64, veya okunamaz karakterler) olduğu gözlemlenmiş ve veri içeriği dışarıdan analiz edilememiştir. Bu, uygulamanın veri gizliliğini başarılı bir şekilde koruduğunu göstermektedir.



Görsel 9 – Şifreli Veri Trafiği Wireshark Üzerinden

6. Sınırlamalar ve İyileştirmeler

1. Sistem yalnızca TCP tabanlı aktarım desteklemektedir. Gelecekte UDP aktarım desteği ile düşük gecikmeli transfer sağlanabilir.
2. RSA ve AES parametreleri sabittir. Dinamik algoritma seçimi arayüzden yapılabilir.
3. IP paket başlığı yalnızca temel alanları (TTL, ID) değiştirebilmektedir. IPsec gibi ileri düzey IP güvenlik entegrasyonları eklenebilir.
4. Paket kaybı testleri yalnızca localhost üzerinden yapılmıştır. Gerçek LAN/WAN ortamlarında test edilmesi faydalı olacaktır.
5. GUI arayüz görsel açıdan basittir; kullanıcı deneyimi geliştirilebilir.

7. Sonuç

Bu projede, istemci-sunucu tabanlı güvenli dosya aktarım sistemi geliştirilmiş ve AES/RSA tabanlı şifreleme teknikleri ile veri güvenliği sağlanmıştır. TCP üzerinden parça bazlı aktarım, IP başlığı manipülasyonu ve saldırı simülasyonları başarıyla entegre edilmiştir. Geliştirilen GUI, kullanım kolaylığı sunarken; yapılan testler sistemin güvenli, kararlı ve ağ koşullarına dayanıklı olduğunu ortaya koymuştur. Proje, ağ güvenliği ve protokol seviyesinde uygulamalı deneyim kazandırmıştır.

8. Kaynakça

1. William Stallings, *Cryptography and Network Security*, Pearson Education, 2020.
2. PyCryptodome Documentation – https://pycryptodome.readthedocs.io/
3. Scapy Project – https://scapy.net/
4. iPerf3 Tool – https://iperf.fr/
5. RFC 791 – Internet Protocol – https://datatracker.ietf.org/doc/html/rfc791
6. OWASP MITM Attack – https://owasp.org/www-community/attacks/Man-in-the-middle/
7. tc (Linux Traffic Control) – https://man7.org/linux/man-pages/man8/tc.8.html
8. Video Link: https://www.youtube.com/watch?v=KWYasapdn3M