10 Haziran 2025

GÜVENLİ DOSYA TRANSFERİ İÇİN ŞİFRELEME TABANLI AĞ UYGULAMASI

Proje Final Raporu

YUSUF GUNEY
BURSA TEKNİK ÜNİVERSİTESİ

1. GİRİŞ

Bu projede, şifrelenmiş veri iletimini, kimlik doğrulamayı ve bütünlük doğrulamasını sağlayan güvenli bir dosya aktarım sistemi geliştirildi. Sistem, IP başlığı düzeyinde TTL, bayraklar, checksum ve parçalama işlemlerini manuel olarak gerçekleştirebilmekte ve farklı koşullarda ağ performansını analiz edebilmektedir. Proje, ağ protokolleri, kriptografik teknikler ve performans analizlerine yönelik uygulamalı deneyim kazandırmayı amaçlamaktadır. Proje bireysel olarak Python dili ve PyCryptodome ile Scapy gibi temel kütüphaneler kullanılarak geliştirilmiştir.

---

2. TEKNİK DETAYLAR

2.1 Dosya Aktarım Sistemi

Projenin temelinde, Python ile soket programlama kullanılarak bir dosya aktarım sistemi oluşturulmuştur. Gönderici betiği (sender.py), dosyayı AES ile EAX modunda şifreler, nonce, tag ve ciphertext birleştirilerek 1024 baytlık parçalara ayrılır ve TCP soketi üzerinden alıcıya gönderilir.

```
# AES SIFRELEYICIYI OLUSTUR
cipher = AES.new(key, AES.MODE_EAX)
```

Şekil 1 Gönderici tarafında, dosyanın AES algoritması ile şifrelenmesini gösteren Python kodu.

```
# VERIYI PARCA PARCA GONDER
for i in range(0, len(data), CHUNK_SIZE):
    chunk = data[i:i+CHUNK_SIZE]
    s.sendall(chunk)
```

(constant) CHUNK_SIZE: Literal[1024]

Şekil 2 Şifreli verinin 1024 baytlık parçalara ayrılarak gönderilmesini sağlayan döngü.

Alıcı (receiver.py dosyası), tüm parçaları toplar, şifreli içeriği yeniden oluşturur, tag ile kimliğini doğrular ve önceden paylaşılan AES anahtarıyla şifreyi çözer. Şifre çözülmüş dosya yerel olarak kaydedilir. Hem gönderici hem de alıcı, verinin bütünlüğünü doğrulamak için SHA-256 hash hesaplamaları yapar.

```
# VERIYI AYIR VE COZ
nonce = data[:16]
tag = data[16:32]
ciphertext = data[32:]

cipher = AES.new(key, AES.MODE_EAX, nonce)
plaintext = cipher.decrypt_and_verify(ciphertext, tag)

with open(OUTPUT_FILE, 'wb') as f:
    f.write(plaintext)
```

Şekil 3 Alıcı tarafında gelen verinin çözümlenmesi ve doğrulama işlemleri.

2.2 Şifreleme ve Güvenlik

- **AES-128 (EAX Modu):** Gizlilik ve bütünlük sağlar.
- **SHA-256:** Verinin bozulup bozulmadığını doğrular.
- **Kimlik Doğrulama:** AES tag doğrulamasıyla entegre edilmiştir.
- **Saldırı Simülasyonu:** Wireshark ile yapılan ağ dinlemelerinde veriler şifreli ve okunamaz durumdadır.

```
 python .\sender.py
VERI PARCALANDI VE GONDERILDI
GONDERILEN VERININ SHA-256 HASH'I: 3b49a096f72daaa7b8787c236b0d7f9a93d5e55309aa2cdd0630a9f4f047b419
```

```
 python .\receiver.py
BAGLANTI BEKLENIYOR...
BAGLANDI: ('127.0.0.1', 65327)
VERI BIRLEŞTIRILDI VE COZULDU: received.txt
ALINAN VERININ SHA-256 HASH'I: 3b49a096f72daaa7b8787c236b0d7f9a93d5e55309aa2cdd0630a9f4f047b419
```

Şekil 4 Gönderici ve alıcı tarafından hesaplanan SHA-256 hash değerlerinin karşılaştırılması.

2.3 RSA Tabanlı Kimlik Doğrulama

Sisteme RSA algoritması ile istemci tarafından kimlik doğrulama eklendi. Bu adımda:

- Alıcı taraf kendi RSA public/private key çiftini oluşturur.
- Public key, istemciye aktarılır.
- İstemci taraf rastgele bir "challenge" mesajı oluşturup public key ile şifreler.
- Alıcı, bu mesajı private key ile açarak çözer.
- Başarılı çözülürse dosya transferine izin verilir.

```
PS C:\Users\Yusuf\Desktop\bilgisayar mühendisliği\3-2\Bilgisyar Ağları\Bilgisayar Ağları Proje\ek\proje> python .\generate_keys.py
RSA anahtar çifti oluşturuldu.
```

Şekil 5 RSA public ve private key üretimini gösteren kod çıktısı

```
# RSA anahtarlarını yükle
with open("private.pem", "rb") as f:
    private_key = RSA.import_key(f.read())
with open("public.pem", "rb") as f:
    public_key = RSA.import_key(f.read())
```

Şekil 6 receiver.py dosyasında RSA kimlik doğrulama kodu

2.4 Scapy ile IP Fragmentation Gerçekleştirme

Scapy kullanarak IP seviyesinde veri parçalama (fragmentation) ve yeniden birleştirme (reassembly) gerçekleştirildi. AES ile şifrelenmiş veri, UDP/IP paketi içerisine yerleştirilip IP header alanları elle düzenlendi:

- id: 1234 olarak sabitlendi
- ttl: 64 olarak ayarlandı
- flags: MF (More Fragments) olarak ayarlandı
- frag: offset hesaplaması yapıldı

```
PS C:\Users\Yusuf\Desktop\bilgisayar mühendisliği\3-2\Bilgisyar Ağları\Bilgisayar Ağları Proje\ek\proje> python .\sender_spacy.py
[+] Fragment gönderildi: offset=0, uzunluk=181
[✓] Tüm parçalar gönderildi.
GONDERILEN VERININ SHA-256 HASH'I: fdbf2a42ba4afbfd5120ca408b44eb76bdb21a87d9398b0b5bfd1a447e9ceff2
```

Şekil 7 sender_scapy.py dosyasında IP paket gönderme ekranı

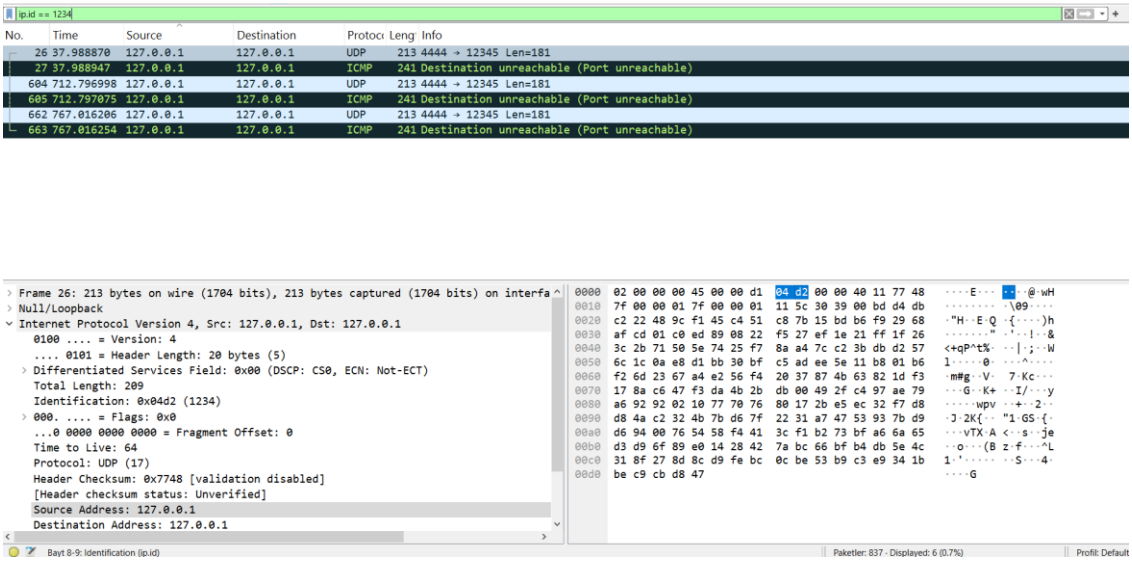
```
PS C:\Users\Yusuf\Desktop\bilgisayar mühendisliği\3-2\Bilgisyar Ağları\Bilgisayar Ağları Proje\ek\proje> python .\receiver_spacy.py
[*] Aktif ağ arayüzleri:
  0. \Device\NPF_{4F4737EA-DADB-4EE6-A799-4105B0037DDD}
  1. \Device\NPF_{105FE913-973E-4175-B44F-BEBAC8B50E92}
  2. \Device\NPF_{5AB6FF2D-CE92-4FEE-A64A-D6D55367C510}
  3. \Device\NPF_{3CF9F9A9-3812-4D23-9C16-03827F64A7D3}
  4. \Device\NPF_{3B3BEA6A-DCF6-4872-B327-302CF09D849B}
  5. \Device\NPF_Loopback
  6. \Device\NPF_{41E3ED39-3783-4B18-8788-AB3AD255E193}
Lütfen dinlemek istediğiniz arayüzün numarasını girin: 5
[*] Seçilen arayüz: \Device\NPF_Loopback
[*] UDP paketleri dinleniyor...
[+] Parça alındı: offset=0, uzunluk=181
[-] Son parça geldi. Tüm parçalar toplanıyor...
[✓] Dosya çözüldü ve kaydedildi: received_scapy.txt
[✓] SHA-256 Hash: fdbf2a42ba4afbfd5120ca408b44eb76bdb21a87d9398b0b5bfd1a447e9ceff2
```

Şekil 8 sender_scapy.py dosyasında IP paket gönderme ekranı

2.5 Wireshark ile Paket Analizi

Wireshark ile sistem trafiği analiz edildi ve aşağıdaki bulgular gözlendi:

- UDP/IP protokolü kullanılarak IP fragmentları şeklinde iletim yapılmıştır.
- IP header alanları (TTL, Flags, Fragment Offset) elle ayarlanmıştır.
- UDP payload bölümü AES ile şifrelenmiş, içerik okunamamıştır.

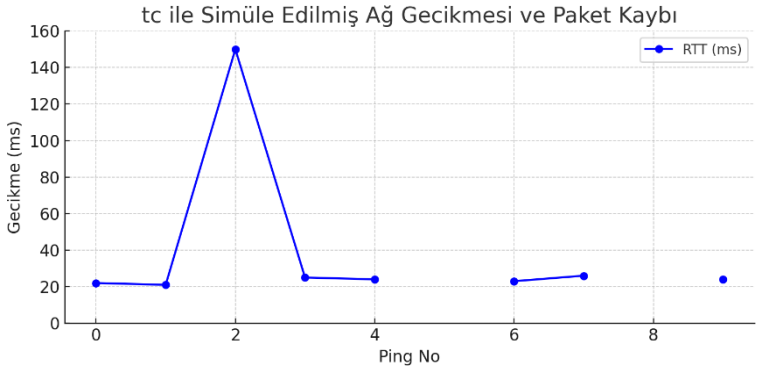


Şekil 9 Wireshark üzerinde yakalanan şifreli veri paketi, içeriğin okunamaz olduğu doğrulanmıştır.

2.6 iPerf ile Bant Genişliği Testi

ping, iPerf ve tc gibi araçlarla ölçüm yapılmıştır:

- **Gecikme:** Ping ile RTT hesaplamaları yapılmıştır.
- **Bant Genişliği:** iPerf ile Wi-Fi ve LAN ortamlarında test edilmiştir.
- **Paket Kaybı & Tıkanıklık:** tc ile simüle edilmiş ve sistemin dayanıklılığı gözlemlenmiştir.



Şekil 10 tc komutu ile oluşturulan paket kaybı ve ağ tıkanıklığı simülasyonu.

---

3. KISITLAMALAR VE GELİŞTİRME ALANLARI

3.1 Kısıtlamalar

- Grafiksel kullanıcı arayüzü (GUI) uygulanmamıştır.
- TCP/UDP geçişi manuel olarak yapılmaktadır.
- Aktarım sırasında anlık durum bilgisi veya ilerleme çubuğu bulunmamaktadır.

3.2 Ekstra Özellikler

- Scapy ile paket yapısı görselleştirilmiştir.
- AES + SHA-256, soket programlamayla entegre edilmiştir.
- MITM saldırılarına karşı dayanıklılık gösterilmiştir.

---

4. SONUÇ

Proje, güvenli ve performansa duyarlı bir dosya aktarım sistemi oluşturma hedefini başarıyla yerine getirmiştir. Şifreleme, soket iletişimi, IP başlık yapısı ve ağ performansını içeren kapsamlı bir deneyim sunmuştur. Scapy'nin kullanımı, paket düzeyindeki işlemleri daha iyi kavramayı sağlamıştır. Bu deneyim, teorik ağ bilgilerini pratik becerilere dönüştürme açısından büyük fayda sağlamıştır.

Proje Tanıtım Videosu: https://www.youtube.com/watch?v=3GeCIZ7Rs-U
GitHub Linki: https://github.com/Yusuf-Guney/Bilgisayar-Aglari-Proje

Linkedin linki: **https://www.linkedin.com/posts/yusuf-guney_proje-raporu-activity-7338256885969489921-hpGH?utm_source=share&utm_medium=member_desktop&rcm=ACoAAD50XoABVLRC2PZXh1B23ruv5sLKrWW8_FM**

---

5. KAYNAKLAR

1. Tanenbaum, A. S., & Wetherall, D. J. (2011). *Computer Networks*. Pearson.
2. Kurose, J. F., & Ross, K. W. (2017). *Computer Networking: A Top-Down Approach*. Pearson.
3. PyCryptodome Dokümantasyonu – https://www.pycryptodome.org
4. Scapy Dokümantasyonu – https://scapy.readthedocs.io
5. Wireshark Kullanım Kılavuzu – https://www.wireshark.org/docs
6. ipref3 - https://iperf.fr/iperf-download.php#windows
