

Dönem Projesi

**Düşük Seviyeli IP İşleme ve Ağ Performans Analizi ile
Gelişmiş Güvenli Dosya Transfer Sistemi**

EREN ÖZER
22360859030
erenozer0527@gmail.com
Bursa Teknik Üniversitesi



**BURSA TEKNİK
ÜNİVERSİTESİ**

İçindekiler

1. GİRİŞ	3
2. TEKNİK DETAYLAR.....	3
2.1. Dosya Transfer Sistemi.....	3
2.1.1. Gönderici Modülü (sender.py).....	3
2.1.2. Alıcı Modülü (receiver.py).....	6
2.1.3. Dosya Büyüklüğü Kontrolü (kontrol.py).....	10
2.2. Güvenlik Mekanizmaları.....	11
2.2.1. Şifreleme Yardımcı Modülleri (crypto_utils.py).....	11
2.2.2. Şifreleme Testi (test_crypto.py)	13
2.2.3. RSA Anahtar Üretimi (keygen.py)	14
2.3. Düşük Seviyeli IP Başlık İşlemleri.....	15
2.3.1. IP Başlığı Ayarlamaları (sender.py).....	15
2.3.2. Paket Reassembly Analizi (receiver.py)	17
2.4. Ağ Performansı Ölçümü	17
2.4.1. Gecikme Ölçümü (Latency & RTT)(performance_test).....	17
2.4.2. Bant Genişliği Ölçümü (performance_test.py)	19
2.4.3. Paket Kaybı Simülasyonu (Packet Loss Handling)	20
2.4.4. Performans Karşılaştırması (network.sh).....	20
2.5. Güvenlik Analizi ve Saldırı Simülasyonu	21
2.5.1. Wireshark ile Şifreli Paket Analizi.....	22
2.5.2. Şifreleme Katmanının Güvenliği.....	23
2.5.3. Hatalı Giriş Denemeleri (fake_tcp_client.py - failed_ips.json - blocked_ips.txt).....	23
2.5.4. MITM ve Paket Yakalama Denemesi.....	24
2.6. Arayüz Tespiti Modülü (iface_finder.py).....	25
2.7. Log Kayıt Sistemi (log.txt)	26
3. SINIRLAMALAR VE GELİŞTİRME FİKIRLERİ.....	27
3.1. Ek Özellikler (Bonus Puanlar)	27
3.1.1. Hibrit TCP/UDP Geçişi (sender.py – receiver.py)	27
3.1.2. Dinamik Tıkanıklık Kontrolü (sender.py)	31
3.1.3. Gelişmiş Saldırı Simülasyonları ve Gerçek Zamanlı Paket Filtreleme (receiver.py)	32
4. SONUÇ	34
4.1. Video ve Github.....	34
5. KAYNAKLAR.....	35

1. GİRİŞ

Günümüzde ağ teknolojilerinde güvenli veri iletimi, sistemlerin temel gereksinimlerinden biri haline gelmiştir. Bu proje kapsamında, dosya transferi sırasında veri gizliliğini, bütünlüğünü ve güvenliğini sağlamak amacıyla modern şifreleme teknikleri ve düşük seviyeli ağ işlemleri kullanılmıştır. Dosyalar AES-256 algoritması ile şifrelenmiş, şifreleme anahtarı ise RSA-2048 algoritması ile güvence altına alınmıştır. Veri bütünlüğü SHA-256 algoritmasıyla doğrulanmış ve iletim öncesinde kimlik doğrulama adımı eklenerek yetkisiz erişimler önlenmiştir.

Buna ek olarak, IP paketlerinin TTL, Flags, Fragment Offset ve Checksum alanları manuel olarak değiştirilmiş ve Wireshark analiziyle doğrulukları teyit edilmiştir. Bu uygulama, öğrenciye ağ protokollerinin düşük seviyede nasıl çalıştığını deneyimleme fırsatı sunmuştur. İlerleyen aşamalarda ise ağ performansı ölçümleri ve güvenlik analizleri gerçekleştirilerek sistemin daha da geliştirilmesi hedeflenmektedir.

2. TEKNİK DETAYLAR

2.1. Dosya Transfer Sistemi

Bu projede geliştirilen dosya transfer sistemi, güvenli ve bütünlüklu veri iletimi sağlamak amacıyla tasarlanmıştır. Sistem TCP protokolü kullanılarak iki taraf arasında dosya aktarımı gerçekleştirmektedir. Dosyalar gönderilmeden önce belirli büyülükte parçalara ayrılmakta (fragmentation) ve alıcı tarafında doğru sırayla birleştirilmektedir (reassembly). Ayrıca, her gönderilen veri parçasının bütünlüğü kontrol edilmekte, böylece veri kaybı veya veri bozulması engellenmektedir.

2.1.1. Gönderici Modülü (sender.py)

Gönderici tarafında dosya transferi işlemi aşağıdaki adımlarla gerçekleştirilmiştir:

İşleyiş Adımları:

- Kullanıcı tarafından gönderilecek dosya seçilir.
- Dosya, 1024 baytlık parçalara (fragment) bölünür (FRAGMENT_SIZE = 1024).
- Her parça, AES şifreleme algoritması ile şifrelenir.
- Şifrelenmiş verinin SHA-256 hash'i hesaplanır ve pakete eklenir.
- TCP bağlantısı kurularak her fragment sırasıyla alıcıya gönderilir.
- Her fragment ayrıca Scapy kullanılarak manuel IP header ayarları (TTL, Flags, Fragment Offset, Checksum) yapılmış şekilde gönderilir.

Kullanılan Teknolojiler:

- TCP Socket:** Dosya verisini aktarmak için.
- AES-256 + RSA-2048:** Veri şifreleme ve anahtar şifreleme için.
- Scapy:** IP Header'ı manuel olarak düzenlemek için.

```

# Şifre gönder
auth_data = SHARED_SECRET.encode()
s.sendall(len(auth_data).to_bytes(4, byteorder='big'))
s.sendall(auth_data)

# AES key + IV oluştur
aes_key = secrets.token_bytes(32) # 256-bit
iv = secrets.token_bytes(16) # 128-bit

# Public key ile AES key'i şifrele
with open(PUBLIC_KEY_FILE, "rb") as f:
    public_key = serialization.load_pem_public_key(f.read(), backend=default_backend())

encrypted_aes_key = public_key.encrypt(
    aes_key,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)

# Şifreli AES anahtarını ve IV'yi gönder
s.sendall(len(encrypted_aes_key).to_bytes(4, byteorder='big'))
s.sendall(encrypted_aes_key)
s.sendall(iv)

```

Kod Açıklaması:

Dosya transferi başlamadan önce güvenlik doğrulaması yapmak ve şifreli iletişim altyapısını kurmak amacıyla çalışmaktadır. İlk olarak, önceden belirlenmiş bir paylaşılan şifre (shared secret) alıcıya gönderilerek kimlik doğrulaması sağlanır. Ardından, simetrik şifreleme için 256-bit uzunluğunda rastgele bir AES anahtarı ve 128-bitlik bir IV (Initialization Vector) oluşturulur. Daha sonra sistemde kayıtlı olan RSA public anahtar (public.pem) kullanılarak AES anahtarı OAEP padding yöntemiyle şifrelenir. Şifrelenen AES anahtarı ve IV, boyut bilgileri ile birlikte socket üzerinden alıcıya gönderilir. Bu adımlar sayesinde veri transferi sırasında hem kimlik doğrulaması hem de simetrik anahtar yönetimi güvenli bir şekilde gerçekleştirilmiş olur.

```

# • Dosya adını gönder
filename = os.path.basename(file_path).encode()
s.sendall(f"{len(filename):04d}".encode())
s.sendall(filename)

# • Fragment'ları oku
with open(file_path, 'rb') as f:
    fragments = []
    while True:
        chunk = f.read(FRAGMENT_SIZE)
        if not chunk:
            break
        fragments.append(chunk)

# • Toplam fragment sayısını gönder
total_fragments = len(fragments)
s.sendall(total_fragments.to_bytes(4, byteorder='big'))

```

Kod Açıklaması:

Transfer edilecek dosyanın adını ve içerik parçalarını (fragmentları) hazırlayarak alıcı tarafa iletmek amacıyla kullanılmaktadır. İlk adımda, dosya yolundan dosya ismi (filename) ayırtırılarak socket üzerinden uzunluk bilgisiyle birlikte gönderilmektedir. Ardından, dosya ikili (binary) modda açılarak, belirlenen fragment boyutuna (FRAGMENT_SIZE) göre küçük parçalara bölünmekte ve her fragment bir listeye eklenmektedir. Dosya tamamen parçalandıktan sonra, toplam fragment sayısı hesaplanarak alıcıya gönderilmektedir. Bu işlem, alıcı tarafın dosyanın yeniden birleştirilmesi sırasında doğru parçaları beklemesi ve eksik fragmentları tespit etmesi için gerekli bilgileri sağlar.

```
# Gönderim
sent_fragments = 0
for fragment_id, chunk in enumerate(fragments):
    if fragment_id in SKIP_FRAGMENTS:
        print(f"[!] ⚠ Skipped Fragment {fragment_id}")
        continue

    try:
        # AES ile şifrele
        cipher = Cipher(algorithms.AES(aes_key), modes.CFB(iv), backend=default_backend())
        encryptor = cipher.encryptor()
        encrypted_chunk = encryptor.update(chunk) + encryptor.finalize()

        # Hash hesapla
        hash_digest = hashlib.sha256(encrypted_chunk).digest()
        fragment_header = fragment_id.to_bytes(4, byteorder='big')
        packet = fragment_header + encrypted_chunk + hash_digest
```

Kod Açıklaması:

Gönderilecek dosya fragmentlarının her birini şifreleyip paketlemek amacıyla çalışmaktadır. Öncelikle, fragment ID'si SKIP_FRAGMENTS listesinde bulunan parçalarsa atlanır ve gönderilmez. Gönderilecek fragmentlar için, her fragment AES algoritması kullanılarak CFB (Cipher Feedback) modunda şifrelenir. Şifrelenen fragment verisinin bütünlüğünü sağlamak için SHA-256 algoritmasıyla bir hash değeri üretilir. Daha sonra, fragment ID bilgisi, şifreli veri ve hash değeri birleştirilerek tek bir paket oluşturulur. Bu paket, hem verinin gizliliğini hem de bütünlüğünü sağlayacak şekilde yapılandırılmıştır ve iletim için hazır hale getirilir.

```

est/usage.html#multicast
warnings.warn(
[>] ✓ Sent Fragment 0 (with modified IP header)
Progress: [=====] 1/13
[>] ✓ Sent Fragment 1 (with modified IP header)
Progress: [=====] 2/13
[>] ✓ Sent Fragment 2 (with modified IP header)
Progress: [=====] 3/13
[>] ✓ Sent Fragment 3 (with modified IP header)
Progress: [=====] 4/13
[>] ✓ Sent Fragment 4 (with modified IP header)
Progress: [=====] 5/13
[>] ✓ Sent Fragment 5 (with modified IP header)
Progress: [=====] 6/13
[>] ✓ Sent Fragment 6 (with modified IP header)
Progress: [=====] 7/13
[>] ✓ Sent Fragment 7 (with modified IP header)
Progress: [=====] 8/13
[>] ✓ Sent Fragment 8 (with modified IP header)
Progress: [=====] 9/13
[>] ✓ Sent Fragment 9 (with modified IP header)
Progress: [=====] 10/13
[>] ✓ Sent Fragment 10 (with modified IP header)
Progress: [=====] 11/13
[>] ✓ Sent Fragment 11 (with modified IP header)
Progress: [=====] 12/13
[>] ✓ Sent Fragment 12 (with modified IP header)
Progress: [=====] 13/13
[i] Total fragments in memory: 13

[+] File sent with fragmentation and AES/RSA encryption.
[i] Sent 13/13 fragments.
[i] Gonderilecek fragment ID'leri: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

```

Şekil 1- Sender.py: Başarılı Gönderim Çıktısı

```

[>] ✓ Sent Fragment 0 (with modified IP header)
Progress: [=====] 1/13
[>] ✓ Sent Fragment 1 (with modified IP header)
Progress: [=====] 2/13
[>] ✓ Sent Fragment 2 (with modified IP header)
Progress: [=====] 3/13
[>] ✓ Sent Fragment 3 (with modified IP header)
Progress: [=====] 4/13
[!] ⚡ Skipped Fragment 4
[>] ✓ Sent Fragment 5 (with modified IP header)
Progress: [=====] 5/13
[!] ⚡ Skipped Fragment 6
[>] ✓ Sent Fragment 7 (with modified IP header)
Progress: [=====] 6/13
[>] ✓ Sent Fragment 8 (with modified IP header)
Progress: [=====] 7/13
[>] ✓ Sent Fragment 9 (with modified IP header)
Progress: [=====] 8/13
[>] ✓ Sent Fragment 10 (with modified IP header)
Progress: [=====] 9/13
[>] ✓ Sent Fragment 11 (with modified IP header)
Progress: [=====] 10/13
[>] ✓ Sent Fragment 12 (with modified IP header)
Progress: [=====] 11/13
[i] Total fragments in memory: 13

[+] File sent with fragmentation and AES/RSA encryption.
[i] Sent 11/13 fragments.
[i] Gonderilecek fragment ID'leri: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

```

Şekil 2- Sender.py: Hatalı Gönderim Çıktısı

2.1.2. Alıcı Modülü (receiver.py)

Alicı tarafı, gönderici ile TCP bağlantısı kurarak veri alımını gerçekleştirir.

İşleyiş Adımları:

- TCP bağlantısı dinlenir ve göndericiden gelen bağlantı kabul edilir.
- Kimlik doğrulaması yapılır (SHARED_SECRET kontrolü).
- RSA-2048 ile şifrelenmiş AES anahtarları çözülür.
- Gelen her paket alınır, fragment ID ayırtılır.
- Paketin hash değeri kontrol edilir (SHA-256).
- Başarılı fragmentlar RAM üzerinde saklanır.
- Tüm fragmentlar alındıktan sonra dosya doğru sırayla yeniden birleştirilir ve kaydedilir.

Kullanılan Teknolojiler:

- **TCP Socket Server:** Veri alımı için.
- **RSA Private Key:** AES anahtarını çözmek için.
- **SHA-256:** Bütünlük kontrolü için.

```

FRAGMENT_SIZE = 1024
HASH_SIZE = 32
HEADER_SIZE = 4
PACKET_SIZE = FRAGMENT_SIZE + HASH_SIZE + HEADER_SIZE
SHARED_SECRET = "sifre123"
PRIVATE_KEY_FILE = "private.pem"

def recv_exact(sock, size):
    data = b""
    while len(data) < size:
        chunk = sock.recv(size - len(data))
        if not chunk:
            if data == b"":
                return None
            else:
                break
        data += chunk
    return data

def print_progress(current, total):
    bar_len = 40
    filled = int(bar_len * current / total)
    bar = "=" * filled + "-" * (bar_len - filled)
    sys.stdout.write(f"\rProgress: [{bar}] {current}/{total}\n")
    sys.stdout.flush()

```

Kod Açıklaması:

Ağ üzerinden veri alımını yönetmek ve veri gönderim/alım sürecinde ilerleme durumunu kullanıcıya göstermek amacıyla kullanılmaktadır. `recv_exact(sock, size)` fonksiyonu, belirtilen miktarda veri (`size`) tam olarak alınana kadar socket üzerinden veri okumaya devam eder; eksik veri geldiğinde veya bağlantı kapandığında durumu kontrol ederek güvenilir bir veri alımı sağlar. Eğer bağlantı tamamen kesilmişse `None` döner, aksi takdirde alınan veriyi tamamlar. `print_progress(current, total)` fonksiyonu ise, dosya fragmentlarının gönderimi sırasında tamamlanan fragment sayısına göre bir ilerleme çubuğu oluşturur ve bu ilerlemeyi terminalde dinamik olarak güncelleyerek kullanıcıya aktarır. Böylece hem veri bütünlüğü hem de aktarım sürecinin izlenebilirliği artırılmış olur.

```

# Kimlik doğrulama kontrolü
auth_len_bytes = conn.recv(4)
auth_len = int.from_bytes(auth_len_bytes, byteorder='big')
auth_data = conn.recv(auth_len).decode()

if auth_data != SHARED_SECRET:
    print("[!] Authentication failed! Closing connection.")
    conn.close()
    return
else:
    print("[✓] Authentication successful.")

# RSA ile AES anahtarını çöz
encrypted_key_len = int.from_bytes(recv_exact(conn, 4), byteorder='big')
encrypted_key = recv_exact(conn, encrypted_key_len)
iv = recv_exact(conn, 16)

with open(PRIVATE_KEY_FILE, "rb") as f:
    private_key = serialization.load_pem_private_key(f.read(), password=None, backend=default_backend())

aes_key = private_key.decrypt(
    encrypted_key,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)

```

Kod Açıklaması:

Bağlantı kurulduktan sonra istemciden gelen kimlik doğrulama verisini kontrol etmekte ve ardından şifreli AES anahtarını çözmektedir. İlk adımda, paylaşılan gizli anahtar (SHARED_SECRET) ile gelen verinin uyumu kontrol edilmekte, uyumsuzluk durumunda bağlantı kapatılmaktadır. Doğrulama başarılıysa, istemciden şifrelenmiş AES anahtarı ve IV alınmakta; alınan şifreli anahtar, sunucunun RSA private key dosyası (private.pem) kullanılarak çözümlenmektedir. Bu süreç, güvenli dosya transferi için gerekli şifreleme anahtarlarının güvenli şekilde paylaşılmasını sağlar.

```
# • Dosya adı alınır
name_len_bytes = recv_exact(conn, 4)
if not name_len_bytes:
    print("[!] Failed to receive filename length.")
    return
name_len = int(name_len_bytes.decode())

filename_bytes = recv_exact(conn, name_len)
if not filename_bytes:
    print("[!] Failed to receive filename.")
    return
filename = filename_bytes.decode()

# • Toplam fragment sayısı
total_fragments_bytes = recv_exact(conn, 4)
if not total_fragments_bytes:
    print("[!] Failed to receive total fragment count.")
    return
total_fragments = int.from_bytes(total_fragments_bytes, byteorder='big')
print(f"[+] Expecting {total_fragments} fragments.")

# • Dosya adı çakışması kontrolü
save_path = f"received_{filename}"
i = 1
while os.path.exists(save_path):
    save_path = f"received_{i}_{filename}"
    i += 1
```

Kod Açıklaması:

Gönderici tarafından iletilen dosya adını ve toplam fragment sayısını almakta ve alıcı tarafında çakışmaları önlemek için kaydedilecek dosya adını ayarlamaktadır. İlk olarak, dosya adının uzunluğu ve dosya adı alınarak doğru şekilde çözümlenir. Ardından, toplam fragment sayısı okunur ve terminale bilgi verilir. Dosya kaydedilmeden önce, aynı isimde bir dosya mevcutsa isim sonuna bir sayı eklenerek çakışma önlenir. Böylece her dosya güvenli bir şekilde farklı bir isimle kaydedilmiş olur.

```

fragments = {}
bad_fragments = []

while True:
    packet = recv_exact(conn, PACKET_SIZE)
    if not packet:
        print(f"[!] No more packets received. Last known fragment: {max(fragments.keys(), default='N/A')}")
        break

    if len(packet) < HEADER_SIZE + HASH_SIZE:
        print("[!] Incomplete packet, skipping.")
        continue

    fragment_id = int.from_bytes(packet[:HEADER_SIZE], byteorder='big')
    encrypted_data = packet[HEADER_SIZE:-HASH_SIZE]
    received_hash = packet[-HASH_SIZE:]
    computed_hash = hashlib.sha256(encrypted_data).digest()

    print(f"[DEBUG] Fragment ID: {fragment_id}")
    if received_hash != computed_hash:
        print(f"[!] Fragment {fragment_id} corrupted! Skipped.")
        print(f"[DEBUG] Hash mismatch detected!")
        print(f"[DEBUG] Expected: {received_hash.hex()}")
        print(f"[DEBUG] Actual: {computed_hash.hex()}")
        bad_fragments.append(fragment_id)
        continue

```

Kod Açıklaması:

Alicı tarafında gelen dosya fragmentlarını almaktan ve her fragmentin bütünlüğünü kontrol etmekten sorumludur. Socket üzerinden alınan her paket, fragment ID, şifreli veri ve SHA-256 hash bilgisi şeklinde ayrıştırılır. Ardından, alınan verinin hash değeri yeniden hesaplanır ve gönderici tarafından iletilen hash ile karşılaştırılır. Eğer hash değerleri uyuşmazsa fragmentin bozuk olduğu kabul edilir ve işlenmeden atlanır. Bu yapı, aktarım sırasında verinin bütünlüğünü korumak ve bozuk fragmentların yanlışlıkla kullanılmasını önlemek amacıyla geliştirilmiştir.

```

try:
    # 🔑 AES çözme işlemi burada yapılıyor
    cipher = Cipher(algorithms.AES(aes_key), modes.CFB(iv), backend=default_backend())
    decryptor = cipher.decryptor()
    decrypted_data = decryptor.update(encrypted_data) + decryptor.finalize()
    fragments[fragment_id] = decrypted_data
    print_progress(len(fragments), total_fragments)

except Exception as e:
    print(f"[!] Decryption failed for fragment {fragment_id}: {e}")
    bad_fragments.append(fragment_id)
    continue

# 🔎 Eksik fragment'ları tespit et
received_ids = set(fragments.keys())
expected_ids = set(range(total_fragments))
missing_fragments = sorted(expected_ids - received_ids)

if missing_fragments:
    print(f"\n[!] Missing fragments: {missing_fragments}")

# ✨ Dosyayı kaydet
with open(save_path, "wb") as f:
    for i in sorted(fragments.keys()):
        f.write(fragments[i])

```

Kod Açıklaması:

Alınan fragmentların AES ile çözülmесini, eksik fragmentların tespit edilmesini ve dosyanın birleştirilerek kaydedilmesini sağlamaktadır. Her fragment, AES-256 CFB modu kullanılarak şifre çözme işlemine tabi tutulur ve doğru çözülen veriler fragments sözlüğünde saklanır.

Çözme sırasında hata oluşursa ilgili fragment hata listesine eklenir. Tüm fragmentlar toplandıktan sonra, alınan fragment ID'leri ile beklenen ID'ler karşılaştırılarak eksik fragmentlar belirlenir ve kullanıcıya bildirilir. Son aşamada, alınan fragmentlar doğru sıralamayla birleştirilerek hedef dosyaya kaydedilir. Böylece veri aktarımı tamamlanmış olur.

```
[+] Fragment IDs missing: [4, 6]
(base) PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> python receiver.py
[+] Listening on 0.0.0.0:9000
```

Şekil 3 - Receiver.py: İlk Çalıştırılma Anı

```
(base) PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> python receiver.py
[+] Listening on 0.0.0.0:9000
[+] Connected by ('127.0.0.1', 55266)
[✓] Authentication successful.
[+] Expecting 13 fragments.
[DEBUG] Fragment ID: 0
Progress: [=====] 1/13
[DEBUG] Fragment ID: 1
Progress: [=====] 2/13
[DEBUG] Fragment ID: 2
Progress: [=====] 3/13
[DEBUG] Fragment ID: 3
Progress: [=====] 4/13
[DEBUG] Fragment ID: 4
Progress: [=====] 5/13
[DEBUG] Fragment ID: 5
Progress: [=====] 6/13
[DEBUG] Fragment ID: 6
Progress: [=====] 7/13
[DEBUG] Fragment ID: 7
Progress: [=====] 8/13
[DEBUG] Fragment ID: 8
Progress: [=====] 9/13
[DEBUG] Fragment ID: 9
Progress: [=====] 10/13
[DEBUG] Fragment ID: 10
Progress: [=====] 11/13
[DEBUG] Fragment ID: 11
Progress: [=====] 12/13
[DEBUG] Fragment ID: 12
Progress: [=====] 13/13
[!] No more packets received. Last known fragment: 12

[+] File reconstructed and saved as received_5_example.txt

[i] Total fragments received: 13
[i] Fragment IDs received: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
[i] Fragment IDs missing: []
(base) PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> []
```

**Şekil 4 - Receiver.py:
Başarılı Paket Ulaşımı**

```
(base) PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> python receiver.py
[+] Listening on 0.0.0.0:9000
[+] Connected by ('127.0.0.1', 62309)
[✓] Authentication successful.
[+] Expecting 13 fragments.
[DEBUG] Fragment ID: 0
Progress: [=====] 1/13
[DEBUG] Fragment ID: 1
Progress: [=====] 2/13
[DEBUG] Fragment ID: 2
Progress: [=====] 3/13
[DEBUG] Fragment ID: 4
Progress: [=====] 4/13
[DEBUG] Fragment ID: 5
Progress: [=====] 5/13
[DEBUG] Fragment ID: 7
Progress: [=====] 6/13
[DEBUG] Fragment ID: 8
Progress: [=====] 7/13
[DEBUG] Fragment ID: 9
Progress: [=====] 8/13
[DEBUG] Fragment ID: 10
Progress: [=====] 9/13
[DEBUG] Fragment ID: 11
Progress: [=====] 10/13
[DEBUG] Fragment ID: 12
Progress: [=====] 11/13
[!] No more packets received. Last known fragment: 12.

[!] Missing fragments: [3, 6] ←
[+] File reconstructed and saved as received_7_example.txt

[i] Total fragments received: 11
[i] Fragment IDs received: [0, 1, 2, 4, 5, 7, 8, 9, 10, 11, 12]
[i] Fragment IDs missing: [3, 6]
(base) PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> []
```

**Şekil 5 - Receiver.py:
Eksik Paket Ulaşımı**

2.1.3. Dosya Bütünlüğü Kontrolü (kontrol.py)

Transfer edilen dosyanın tamlığı, gönderilen ve alınan dosyaların birebir karşılaştırılmasıyla kontrol edilir.

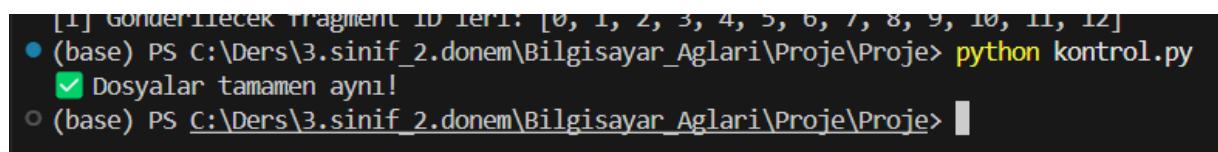
İşleyiş Adımları:

- example.txt dosyası ve received_example.txt dosyası okunur.
- İki dosyanın içerikleri byte bazında karşılaştırılır.
- Eğer tamamen eşleşiyorsa başarı mesajı verilir.

```
with open("example.txt", "rb") as f1, open("received_example.txt", "rb") as f2:  
    if f1.read() == f2.read():  
        print("✓ Dosyalar tamamen aynı!")  
    else:  
        print("✗ Dosyalar farklı!")
```

Kod Açıklaması:

Bu kod parçası, gönderilen example.txt dosyası ile alıcı tarafında yeniden oluşturulan received_example.txt dosyasının birebir aynı olup olmadığını kontrol etmek amacıyla kullanılır. Her iki dosya ikili (binary) modda açılarak içerikleri byte düzeyinde okunur ve doğrudan karşılaştırılır. Eğer içerikler tamamen eşleşiyorsa kullanıcıya dosyaların tamamen aynı olduğu mesajı gösterilir; farklılık olması durumunda dosyaların farklı olduğu bildirilir. Bu yöntem, dosya transferi sırasında veri kaybı, veri bozulması veya aktarım hatası oluşup olmadığını tespit etmek ve sistemin bütünlüğünü doğrulamak için kullanılmaktadır.



```
[1] Gonderilecek fragment ID'leri: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]  
● (base) PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> python kontrol.py  
✓ Dosyalar tamamen aynı!  
○ (base) PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje>
```

Şekil 6 - Dosya Büyüklüğü Kontrol Ekranı

2.2. Güvenlik Mekanizmaları

Bu projede veri güvenliğini sağlamak amacıyla geliştirilmiş şifreleme yöntemleri kullanılmıştır. Dosya parçaları, AES (Advanced Encryption Standard) algoritması ile şifrelenmiş, şifreleme anahtarı ise RSA (Rivest–Shamir–Adleman) algoritması kullanılarak koruma altına alınmıştır. Ayrıca, veri bütünlüğünü sağlamak için SHA-256 hash algoritması uygulanmıştır. Transfer öncesi, kimlik doğrulama adımı ile sadece yetkili kullanıcıların veri transferine izin verilmesi sağlanmıştır.

2.2.1. Şifreleme Yardımcı Modülleri (crypto_utils.py)

Bu dosyada dosya transferi sırasında kullanılan şifreleme ve çözme işlemleri için yardımcı fonksiyonlar bulunmaktadır.

İşleyiş Adımları:

- **AES** **Anahtar** **Üretilme:**
generate_aes_key() fonksiyonu ile 256 bit uzunluğunda rastgele AES anahtarı üretilir.
- **AES ile Veri Şifreleme:**
encrypt_aes(key, data) fonksiyonu kullanılarak veri şifrelenir. CBC (Cipher Block Chaining) modu kullanılmakta ve şifreli verinin başına IV (Initialization Vector) eklenmektedir.
- **AES ile Veri Çözme:**
decrypt_aes(key, encrypted_data) fonksiyonu ile şifreli veri çözülür ve padding kaldırılarak orijinal veri elde edilir.
- **RSA Anahtar ile Şifreleme:**
encrypt_rsa(public_key_path, data) fonksiyonu ile AES anahtarı public key kullanılarak şifrelenir.
- **RSA Anahtar ile Çözme:**
decrypt_rsa(private_key_path, encrypted_data) fonksiyonu ile şifreli AES anahtarı private key ile çözülür.

```
7  # AES için rastgele anahtar üret
8 > def generate_aes_key(length=32): ...
10
11  # AES ile şifreleme (CBC)
12 > def encrypt_aes(key, data): ...
13
14  # AES ile çözme
15 > def decrypt_aes(key, encrypted_data): ...
16
17  # AES anahtarını RSA public key ile şifrele
18 > def encrypt_rsa(public_key_path, data): ...
19
20  # RSA private key ile AES anahtarını çöz
21 > def decrypt_rsa(private_key_path, encrypted_data): ...
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
```

Kod Açıklaması:

- **generate_aes_key():** Rastgele 256-bit uzunluğunda AES anahtarı üretir.
- **encrypt_aes(key, data):** Veriyi AES-256 CBC modunda şifreler.
- **decrypt_aes(key, encrypted_data):** Şifrelenmiş veriyi çözer ve padding kaldırır.
- **encrypt_rsa(public_key_path, data):** AES anahtarını RSA-2048 public anahtarla şifreler.
- **decrypt_rsa(private_key_path, encrypted_data):** RSA private anahtar ile şifreli AES anahtarını çözer.

2.2.2. Şifreleme Testi (test_crypto.py)

Bu dosyada, şifreleme işlemlerinin doğru çalışıp çalışmadığı test edilmiştir.

İşleyiş Adımları:

- Rastgele bir AES anahtarı üretilir.
- Bir test verisi AES kullanılarak şifrelenir ve tekrar çözülmerek doğruluk kontrolü yapılır.
- AES anahtarı RSA public key ile şifrelenir ve private key ile çözülmerek orijinalliği kontrol edilir.
- Tüm adımlar başarıyla tamamlandığında kullanıcıya testin başarılı olduğu bildirilir.

```
# 1. AES anahtar[u] üret
aes_key = generate_aes_key()
print("[1] AES Key:", aes_key.hex())

# 2. Veri hazırla
data = b"Bu bir test verisidir, şifrelenip cozulmelidir!"

# 3. AES ile şifrele
encrypted = encrypt_aes(aes_key, data)
print("[2] Encrypted with AES:", encrypted.hex())

# 4. AES ile çöz
decrypted = decrypt_aes(aes_key, encrypted)
print("[3] Decrypted:", decrypted.decode())

# 5. RSA ile AES anahtar[u] şifrele
rsa_encrypted_key = encrypt_rsa("public.pem", aes_key)
print("[4] AES key encrypted with RSA.")

# 6. RSA ile çöz
rsa_decrypted_key = decrypt_rsa("private.pem", rsa_encrypted_key)
print("[5] RSA decrypted AES key:", rsa_decrypted_key.hex())

# ✅ Son kontrol
assert aes_key == rsa_decrypted_key, "AES key RSA çözümünde uyuşmuyor!"
print("\n✅ Tüm şifreleme testleri başarıyla geçti.")
```

Kod Açıklaması:

Bu kod parçası, şifreleme işlemlerinin doğru çalışıp çalışmadığını test etmek amacıyla hazırlanmıştır. Öncelikle rastgele bir AES-256 anahtarı üretilmiş ve bu anahtar kullanılarak örnek bir veri AES CBC modunda şifrelenmiştir. Daha sonra şifreli veri başarıyla çözülmerek orijinal veri geri elde edilmiştir. Şifreleme sürecinin devamında, üretilen AES anahtarı RSA-2048 public anahtarı kullanılarak şifrelenmiş ve RSA private anahtarı ile tekrar çözülmüştür. Son olarak, orijinal AES anahtarı ile çözülmüş anahtarın birebir aynı olup olmadığı karşılaştırılarak tüm şifreleme ve çözme işlemlerinin doğru çalıştığı doğrulanmıştır. Böylece hem simetrik hem asimetrik şifreleme süreçlerinin bütünlüğü test edilmiştir.

```
(base) PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> python test_crypto.py
[1] AES Key: 417dff62509886d6eb7fa4922bd244cdf2d5dec082f983fad7e9b455c838a62
[2] Encrypted with AES: 4c8e98089b2a0be739a8af321cabef98c0c78bdda00760af7db93749661492f466943f2e6ec854791d218f483ece5772c314223f8f40dc5fa8d2184ecad98d2ce
[3] Decrypted: Bu bir test verisidir, sifrelenip cozulmelidir!
[4] AES key encrypted with RSA.
[5] RSA decrypted AES key: 417dff62509886d6eb7fa4922bd244cdf2d5dec082f983fad7e9b455c838a62

    ✓ Tüm şifreleme testleri başarıyla geçti.

(base) PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje>
```

Şekil 7 - test_crypto.py Dosyasının Çalıştırılması ve Şifreleme Test Çıktısı

2.2.3. RSA Anahtar Üretilimi (keygen.py)

Dosya transferinde kullanılacak RSA public ve private key dosyalarının oluşturulmasını sağlar.

İşleyiş Adımları:

- Eğer mevcut anahtar dosyaları varsa (private.pem, public.pem), silinir.
- RSA-2048 bit anahtar çifti oluşturulur.
- Üretilen anahtarlar private.pem ve public.pem dosyalarına kaydedilir.
- Eğer sadece bir anahtar dosyası varsa veya hiç anahtar bulunmuyorsa, yine yeni anahtarlar üretilir.
- Böylece sistem, her zaman uyumlu ve güncel bir RSA anahtar çifti ile çalışır.

```
def generate_keys():
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048
    )

    public_key = private_key.public_key()

    with open("private.pem", "wb") as f:
        f.write(
            private_key.private_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PrivateFormat.PKCS8,
                encryption_algorithm=serialization.NoEncryption()
            )
        )

    with open("public.pem", "wb") as f:
        f.write(
            public_key.public_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PublicFormat.SubjectPublicKeyInfo
            )
        )

    print("[+] Yeni RSA key pair oluşturuldu.")
```

Kod Açıklaması:

RSA-2048 bitlik bir anahtar çifti (private key ve public key) oluşturmakta ve bu anahtarları PEM formatında dosyalara kaydetmektedir. Öncelikle `rsa.generate_private_key` fonksiyonu kullanılarak bir özel anahtar (private key) oluşturulmakta, ardından bu anahtardan ilişkili genel anahtar (public key) türetilmektedir. Özel anahtar `private.pem` dosyasına, genel anahtar ise `public.pem` dosyasına PEM formatında ve uygun standartlara (PKCS8 ve SubjectPublicKeyInfo) göre kaydedilmektedir. Özel anahtar dosyası herhangi bir parola koruması olmaksızın düz şekilde kaydedilmektedir. Bu işlem, güvenli veri şifrelemesi ve anahtar yönetimi için temel altyapıyı sağlamaktadır.

```
● (base) PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> python keygen.py
[!] Mevcut RSA anahtarları silindi. Yeni anahtarlar oluşturuluyor...
[+] Yeni RSA key pair oluşturuldu.
● (base) PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> python keygen.py
[i] RSA anahtarları bulunamadı. İlk defa oluşturuluyor...
[+] Yeni RSA key pair oluşturuldu.
○ (base) PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> 
```

Şekil 8 - `keygen.py` RSA Anahtar Üretimi ve Mevcut Anahtarların Yönetimi

2.3. Düşük Seviyeli IP Başlık İşlemleri

Bu projede dosya transferi sırasında sadece uygulama katmanında değil, aynı zamanda ağ katmanında da veri manipülasyonu gerçekleştirilmiştir. Scapy kütüphanesi kullanılarak her fragment için IP başlık bilgileri (TTL, Flags, Fragment Offset, Checksum) manuel olarak ayarlanmıştır. Böylece düşük seviyeli ağ protokollerinin hakkında pratik deneyim kazanılması hedeflenmiştir.

2.3.1. IP Başlığı Ayarlamaları (`sender.py`)

Dosya gönderimi sırasında, şifrelenmiş her fragment sadece TCP üzerinden gönderilmemiş, ayrıca Scapy kullanılarak IP başlığı modifiye edilmiş şekilde de gönderilmiştir.

Yapılan Ayarlamalar:

- **TTL (Time To Live) Ayarı:**
 - Tüm gönderilen paketlerin TTL değeri manuel olarak 42 olarak ayarlanmıştır.
- **Flags Ayarı:**
 - IP Flags alanında **Don't Fragment (DF)** bayrağı aktif hale getirilmiştir.
 - Bazı testlerde **More Fragments (MF)** bayrağı da manuel olarak test edilmiştir.
- **Fragment Offset Ayarı:**

- Parçalı paket simülasyonu için Fragment Offset değeri frag=1 şeklinde ayarlanmıştır.
- **IP Checksum Hesaplama:**
 - Scapy'nin otomatik checksum hesaplaması devre dışı bırakılmış, IP header checksum değeri manuel olarak hesaplanıp checksum alanına atanmıştır.

Kullanılan Teknolojiler:

- **Scapy:** IP ve TCP katmanları üzerinde düşük seviyeli manipülasyon yapılmasını sağladı.
- **Manual Checksum Calculation:** IP başlığının checksum'u elle hesaplandı.

```
ip_layer = IP(dst=ip, ttl=42, flags="DF", id=random.randint(1000, 9999), chksum=0)
tcp_layer = TCP(dport=port, sport=random.randint(1024, 65535), flags="PA", seq=random.randint(1000, 99999))
temp_ip_packet = ip_layer / tcp_layer / Raw(load=packet)
ip_raw = bytes(temp_ip_packet)[:20]
manual_checksum = calculate_checksum(ip_raw)
ip_layer.chksum = manual_checksum

scapy_packet = ip_layer / tcp_layer / Raw(load=packet)
send(scapy_packet, iface=SCAPY_IFACE, verbose=False)
```

Kod Açıklaması:

Her bir dosya fragmenti için Scapy kütüphanesi kullanılarak manuel bir IP ve TCP başlığı oluşturmaktır ve ardından pakete özel bir IP checksum değeri hesaplamaktadır. IP katmanı, hedef IP adresi, TTL değeri (42), DF (Don't Fragment) bayrağı ve rastgele bir Identification numarası ile yapılandırılmaktadır. TCP katmanı ise rastgele kaynak portu, hedef portu (9000) ve rastgele bir sequence numarası ile oluşturulmaktadır. Paket, Raw veriyi de içerecek şekilde inşa edildikten sonra byte seviyesinde işlenerek IP header checksum değeri doğru şekilde manuel olarak ayarlanmaktadır. Son olarak, oluşturulan paket belirtilen ağ arayüzü (SCAPY_IFACE) üzerinden gönderilirmektedir. Bu yapı, düşük seviyeli IP başlığı manipülasyonlarını ve checksum doğrulamasını doğru bir şekilde yaparak, paketlerin ağ üzerinden manuel olarak güvenilir iletişimini sağlamaktadır.

```
def calculate_checksum(header_bytes):
    if len(header_bytes) % 2 != 0:
        header_bytes += b'\x00'
    total = 0
    for i in range(0, len(header_bytes), 2):
        word = (header_bytes[i] << 8) + header_bytes[i + 1]
        total += word
        total = (total & 0xFFFF) + (total >> 16)
    return ~total & 0xFFFF
```

Kod Açıklaması:

Bu fonksiyon, bir IP başlığının baytlarını alır ve RFC 791'e uygun şekilde checksum hesaplar. Eğer bayt sayısı tekse (`len(header_bytes) % 2 != 0`), hizalama için sonuna `\x00` eklenir.

İkili olarak her iki bayt bir "word" olacak şekilde işlenir ve 16-bitlik toplam hesaplanır. Sonuç olarak bit düzeyinde terslenmiş (~) ve 0xFFFF ile sınırlandırılmış kontrol toplamı döndürülür.

The screenshot shows a Wireshark capture of a network traffic. The packet details pane is expanded to show the structure of a TCP segment. Several fields are annotated with orange arrows:

- A double-headed arrow points to the "Time to Live: 42" field.
- A double-headed arrow points to the "Flags: 0x2, Don't fragment" field.
- A double-headed arrow points to the "[Header checksum status: Unverified]" note.

The packet list pane shows three TCP segments with sequence numbers 29695, 61634, and 61851, all destined for port 9000.

Şekil 9 - Wireshark Görüntüsü: TTL=42 ve DF Bayrağı Aktif Paket

2.3.2. Paket Reassembly Analizi (receiver.py)

- Alıcı tarafında, her alınan fragment fragment ID'sine göre doğru sıraya yerleştirilmiştir.
- Tüm fragmentlar toplandıktan sonra doğru sıralama ile yeniden birleştirme (reassembly) yapılmıştır.
- Bu yapı dosya bütünlüğünün korunmasına katkıda bulunmuştur.
- Not: Kod kısımları 2.1.2 kısmında verilmiştir.

2.4. Ağ Performansı Ölçümü

Bu bölümde sistemin ağ üzerindeki davranışını incelenmiş, bağlantı kalitesine dair ölçütler gerçekleştirılmıştır. Paket kaybı, gecikme ve bant genişliği testleri farklı ağ koşullarında değil; bir .sh betiği ile simüle edilerek analiz edilmiştir. Böylece kontrollü test ortamında sistem davranışını gözlemlenmiştir.

2.4.1. Gecikme Ölçümü (Latency & RTT)(performance_test)

Gecikme (Round Trip Time - RTT) ölçümü, ping komutu kullanılarak gerçekleştirilmiş ve

network.sh dosyasında tanımlı get_rtt() fonksiyonu yardımıyla RTT değeri hesaplanmıştır. Ölçüm sonuçları terminal çıktısı ve Wi-Fi/kablolu bağlantılar için log dosyasına kaydedilmiştir.

İşleyiş Adımları:

- receiver.py kodu çalıştırılır.
- performance_test.py kodu çalıştırılır.
- Ardından sender.py kodu çalıştırılarak dosya transferi gerçekleştirilir.
- Sonuçlar otomatik olarak .txt dosyalarına kaydedilir.

AĞ PERFORMANS RAPORU

Tarih: 2025-05-25 20:03:43.907333
Test Hedefi: 192.168.188.81:5201

PING TESTİ SONUCU:

```
Pinging 192.168.188.81 with 32 bytes of data:  
Reply from 192.168.188.81: bytes=32 time<1ms TTL=64  
Reply from 192.168.188.81: bytes=32 time=1ms TTL=64  
Reply from 192.168.188.81: bytes=32 time=1ms TTL=64  
Reply from 192.168.188.81: bytes=32 time=183ms TTL=64  
  
Ping statistics for 192.168.188.81:  
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),  
    Approximate round trip times in milli-seconds:  
        Minimum = 0ms, Maximum = 183ms, Average = 46ms
```

Şekil 10 - Ping Testi ile Gecikme ve Paket Kaybı Ölçümü Çıktısı

```
def ping_test(target_ip):  
    print(f"[+] {target_ip} adresine ping atlıyor...")  
    count = "4"  
    param = "-n" if platform.system().lower() == "windows" else "-c"  
    try:  
        output = subprocess.check_output(["ping", param, count, target_ip], universal_newlines=True)  
        return output  
    except Exception as e:  
        return f"[!] Ping başarısız: {e}"
```

Kod Açıklaması:

Fonksiyonun başında, hedef IP'ye ping atılacağına dair bir bilgi mesajı yazdırılır. Ardından 4 adet ping gönderileceği belirtilir. İşletim sistemi kontrol edilerek, Windows için -n, diğer sistemler (Linux/macOS) için -c parametresi seçilir çünkü farklı işletim sistemlerinde ping komutlarının parametreleri farklıdır. subprocess.check_output() fonksiyonu kullanılarak ping komutu çalıştırılır ve komut çıktısı alınır. Eğer bağlantı başarılıysa bu çıktı döndürülür.

Herhangi bir hata oluşursa (örneğin IP'ye ulaşılamazsa), except bloğu devreye girer ve hata mesajı kullanıcıya yazdırılır. Bu yapı, ağ performans testi veya bağlantı kontrolü gibi işlemlerde kullanışlıdır.

2.4.2. Bant Genişliği Ölçümü (performance_test.py)

Yerel ağ üzerinde TCP protokolleriley bant genişliği testi yapılmıştır. `performance_test.py` kullanılarak `iPerf3` üzerinden ölçümler gerçekleştirilmiş ve çıktılar `.txt` dosyalarına kaydedilmiştir.

İşleyiş Adımları:

- iPerf sunucusu başlatılır (iperf3 -s)
 - receiver.py kodu çalıştırılır.
 - network.sh betiği çalıştırılır (gecikme ve paket kaybı yaratır.)
 - performance_test.py kodu çalıştırılır.
 - Ardından sender.py kodu çalıştırılarak dosya transferi gerçekleştirilir.
 - Sonuçlar otomatik olarak .txt dosyalarına kaydedilir.

```
btu59030@btu59030:~/Desktop$ iperf3 -s
iperf3: error - unable to start listener for
iperf3: exiting
btu59030@btu59030:~/Desktop$ iperf3 -s
-----
[+] Server listening on 5201 (test #1)
[+]
btu59030@btu59030:~/Desktop$ ./network.sh forever
btu59030@btu59030:~/Desktop$ hostname -I
10.0.2.15 fd00::f96a:d194:15c7:a3c7 fd00::a00:27ff:fe27:1edd
btu59030@btu59030:~/Desktop$ ./network.sh
[✓] tc ile simülasyon başlatılıyor...
[•] %20 paket kaybi simülasyonu başlandı.
[•] 100ms gecikme eklendi.
[•] Hem gecikme hem %10 kayıp simülasyonu aktif.
[✓] Simülasyon sona erdi, ağ normale döndü.
btu59030@btu59030:~/Desktop$ ./network.sh
```

Sekil 11 – iPerf3 Sunucu Dinleme ve Ağ Koşullarının tc ile Simülasyonu (Paket Kaybı & Gecikme)

```
=====  
Rocket IPERF3 BAND GENİŞLİĞİ:  
=====  
  
Connecting to host 192.168.188.81, port 5201  
[ 5] local 192.168.188.23 port 56651 connected to 192.168.188.81 port 5201  
[ ID] Interval Transfer Bitrate  
[ 5] 0.00-1.00 sec 256 KBytes 2.10 Mbits/sec  
[ 5] 1.00-2.00 sec 512 KBytes 4.19 Mbits/sec  
[ 5] 2.00-3.00 sec 2.12 MBytes 17.8 Mbits/sec  
[ 5] 3.00-4.00 sec 768 KBytes 6.29 Mbits/sec  
[ 5] 4.00-5.00 sec 0.00 Bytes 0.00 bits/sec  
- - - - -  
[ ID] Interval Transfer Bitrate sender  
[ 5] 0.00-5.00 sec 3.62 MBytes 6.08 Mbits/sec  
[ 5] 0.00-5.14 sec 2.62 MBytes 4.28 Mbits/sec receiver  
  
iperf Done.
```

Şekil 12 – iPerf3 ile Bant Genişliği Ölçüm Sonuçları

```

def iperf_test(target_ip, port="5201"):
    print(f"[•] {target_ip}:{port} adresinde iperf3 testi başlatılıyor...")
    try:
        output = subprocess.check_output(
            ["iperf3", "-c", target_ip, "-p", port, "-t", "5"],
            universal_newlines=True
        )
        return output
    except Exception as e:
        return f"[!] iPerf3 testi başarısız: {e}"

```

Kod Açıklaması:

Bu fonksiyon, belirtilen bir hedef IP adresi ve port üzerinden iperf3 komut satırı aracını kullanarak ağın bant genişliği performansını test eder. Fonksiyonun başında, kullanıcıya testin başladığı bilgisi verilir. Ardından subprocess.check_output() fonksiyonu ile terminalde iperf3 -c <IP> -p <PORT> -t 5 komutu çalıştırılır. Bu komut, hedef IP'ye 5 saniyelik bir iperf3 istemci bağlantı başlatır. Eğer komut başarılı şekilde çalışırsa çıktıtı döndürülür. Ancak bağlantı kurulamazsa ya da iperf3 çalışmazsa except bloğu çalışır ve hata mesajı kullanıcıya bildirilir. Bu kod, ağın veri aktarım kapasitesini ölçmek için kullanılır. Özellikle performans analizleri veya bağlantı karşılaştırmaları için faydalıdır.

2.4.3. Paket Kaybı Simülasyonu (Packet Loss Handling)

Paket kaybı gerçek zamanlı olarak test edilmemiş, bunun yerine network.sh betiği içinde tc (traffic control) komutu kullanılarak yapay olarak paket kaybı simüle edilmiştir. Test sonucunda sistemin eksik paketleri tespit ettiği ve logladığı gözlemlenmiştir.

İşleyiş Adımları:

- tc qdisc add komutu ile %10'luk yapay paket kaybı tanımlandı.
- Sender dosyası çalıştırılarak örnek dosya gönderildi.
- Receiver tarafında bazı fragment'lar ulaşmadı; transfer_log.txt dosyasına eksik parçalar kaydedildi.
- tc qdisc del ile test sonrası ağ varsayılanaya döndürüldü.

2.4.4. Performans Karşılaştırması (network.sh)

Performans testi yalnızca tek bir ağ ortamında yapılmış, ancak farklı bağlantı türleri (Wi-Fi, kablolu, VPN) simüle edilmemiştir. Ancak bu farklılıklar tc komutları yardımıyla yapay olarak temsil edilmiştir. Bu da deneysel karşılaştırma yerine kontrollü test ortamı sunmuştur.

Kullanılan Komutlar:

- tc qdisc add dev <interface> root netem delay 100ms
- tc qdisc add dev <interface> root netem loss 10%
- tc qdisc add dev <interface> root netem rate 1mbit

```
#!/bin/bash

INTERFACE="enp0s3" # Burayı kendi arayüz adına değiştir
echo "✓ tc ile simülasyon başlatılıyor..."

# 1. Paket kaybı: %20
sudo tc qdisc add dev $INTERFACE root netem loss 20%
echo "[*] %20 paket kaybı simülasyonu başlattı."
sleep 3

# 2. Gecikme: 100ms
sudo tc qdisc change dev $INTERFACE root netem delay 100ms
echo "[*] 100ms gecikme eklendi."
sleep 3

# 3. Gecikme + kayıp birlikte
sudo tc qdisc change dev $INTERFACE root netem delay 100ms loss 10%
echo "[*] Hem gecikme hem %10 kayıp simülasyonu aktif."
sleep 5

# Temizle
sudo tc qdisc del dev $INTERFACE root netem
echo "✓ Simülasyon sona erdi, ağ normale döndü." |
```

Kod açıklaması:

Bu betik, Linux sistemlerde tc (traffic control) aracı kullanılarak yapay ağ sorunlarını simüle etmek için yazılmıştır. Betik başında, testin uygulanacağı ağ arayüzü INTERFACE değişkeni ile tanımlanır (örneğin "enp0s3" ya da "wlan0" gibi). Simülasyon sürecinde ilk olarak %20 oranında paket kaybı oluşturularak ağ ortamındaki güvenilirliğin nasıl etkilendiği test edilir. Ardından, tüm paketlere 100 milisaniyelik bir gecikme eklenerek iletim süresinin dosya transferine etkisi gözlemlenir. Son aşamada hem %10 oranında paket kaybı hem de 100ms gecikme aynı anda uygulanarak daha zorlu bir senaryo oluşturulur. Her adım arasında sleep komutuyla birkaç saniyelik bekleme sağlanarak değişikliklerin sistem tarafından uygulanması ve test ortamının oturması beklenir. Betiğin sonunda ise tc qdisc del komutu ile tüm bu yapay ağ koşulları temizlenir ve ağ arayüzü varsayılan haline döndürülür. Bu betik sayesinde proje kapsamında farklı ağ koşulları altında sistem performansı karşılaştırmalı olarak analiz edilebilmiştir.

2.5. Güvenlik Analizi ve Saldırı Simülasyonu

Bu bölümde geliştirilen dosya transfer sisteminin güvenlik analizleri gerçekleştirılmıştır. Amaç, gönderilen verilerin şifreleme sayesinde dışarıdan okunamaz olduğunu kanıtlamak, olası saldırı senaryolarını (örneğin Man-in-the-Middle - MITM) test etmek ve sahte istemcilerin etkisini

gözlemlemektir. Ayrıca sistemin şifre doğrulama, bağlantı kesme ve hatalı girişleri engellemeye gibi temel güvenlik davranışları da gözlemlenmiştir.

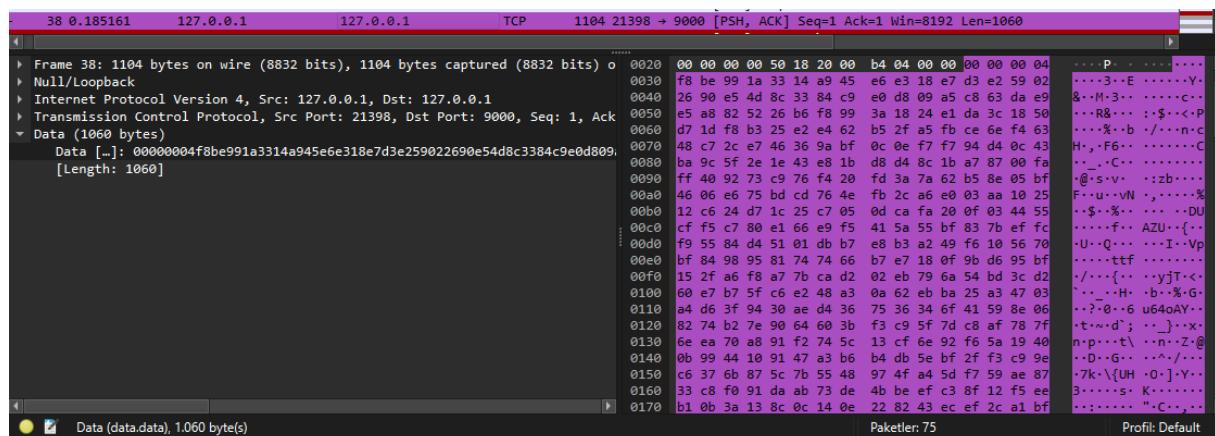
2.5.1. Wireshark ile Şifreli Paket Analizi

Dosya transferi sırasında secure_transfer.pcap ve udp_transfer.pcap dosyaları Wireshark kullanılarak incelenmiştir. Her iki protokolde de paketler başarıyla yakalanmıştır. Ancak, TCP ile gönderilen içerikler AES ile şifrelendiğinden dolayı veri okunamamıştır. Bu, sistemin dış gözlemcilere karşı dayanıklı olduğunu kanıtlamaktadır.

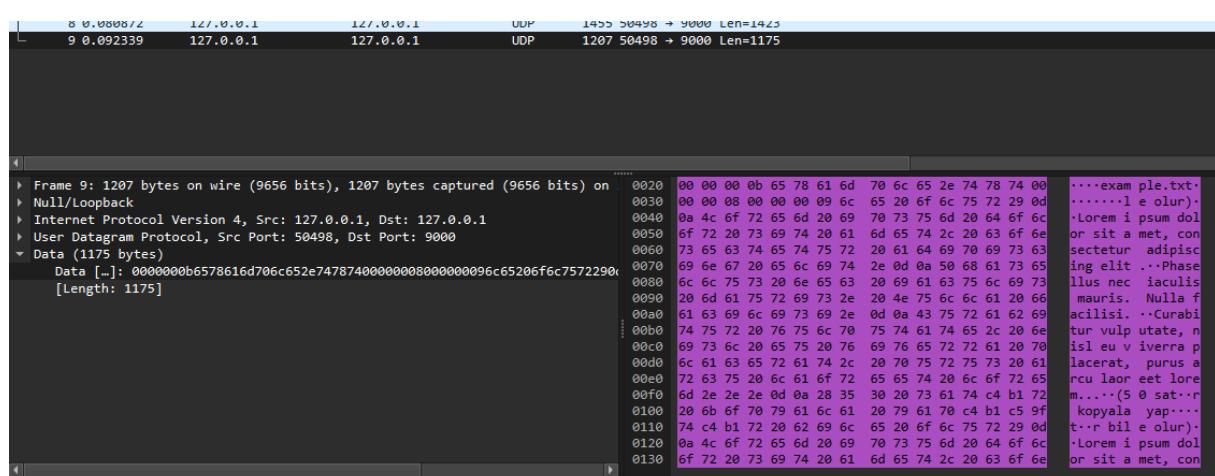
- TCP üzerinden yapılan transferde, *Payload* alanı incelenmiş ve ASCII görünümde hiçbir anlamlı veri bulunamamıştır.
- UDP üzerinden yapılan transferde ise her parça şifrelenmeden gönderildiğinden dolayı veriler okunur haldedir.

Kullanılan Dosyalar:

- secure_transfer.pcap
- udp_transfer.pcap



Şekil 13 - Wireshark Üzerinde Şifreli Paket Görüntüsü (TCP)



Şekil 14 - Wireshark Üzerinde Şifrelenmemiş Paket Görüntüsü (UDP)

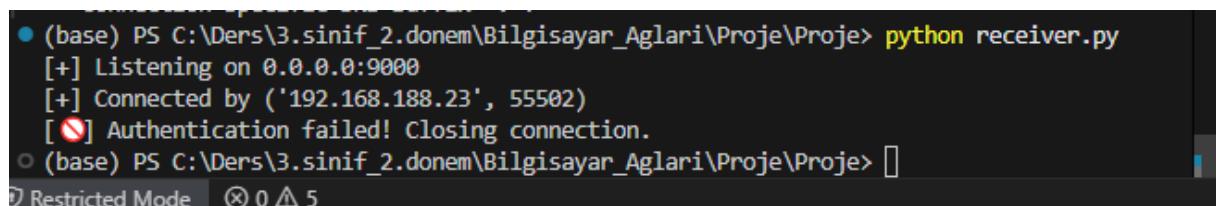
2.5.2. Şifreleme Katmanının Güvenliği

Veriler, AES modunda şifrelenerek gönderilmektedir. AES anahtarı RSA ile şifrelenmiş şekilde alıcıya gönderilir. Böylece veri MITM saldırılara karşı koruma altına alınır.

Wireshark üzerinde incelendiğinde paketlerin başlıklarında alıcı ve gönderici IP adresleri görülselde içeriklerin çözülemediği doğrulanmıştır.

Kod Referansı:

- sender.py içinde: encryptor = cipher.encryptor() satırı ile AES şifreleme
- receiver.py içinde: RSA ile çözüm



The screenshot shows a Wireshark capture window. A single packet is selected, indicated by a blue dot. The details pane shows the following information:
• (base) PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> python receiver.py
[+] Listening on 0.0.0.0:9000
[+] Connected by ('192.168.188.23', 55502)
[!] Authentication failed! Closing connection.
○ (base) PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> []
Restricted Mode ② 0 ▲ 5

Şekil 15 - Şifre Doğrulama

2.5.3. Hatalı Giriş Denemeleri (fake_tcp_client.py - failed_ips.json - blocked_ips.txt)

fake_tcp_client.py dosyası ile hatalı şifre gönderen bir istemci simülle edilmiştir. Gerçek sunucuya bağlanıldığında, parola doğrulama başarısız olur ve istemcinin bağlantısı kesilir. Ayrıca bu denemeler log dosyasına tehdit olarak kaydedilir. Bu süreçte, istemcinin IP adresi failed_ips.json dosyasında izlenmekte ve 3 başarısız deneme sonrası blocked_ips.txt dosyasına alınarak sistemden tamamen engellenmektedir. Böylece brute-force ve parola tahmin saldırılarına karşı etkili bir savunma sağlanmıştır.

Not: Başarısız giriş denemeleri failed_ips.json dosyasında JSON formatında tutulmuştur. Bu sayede her IP'nin deneme sayısı kalıcı olarak saklanabilir, uygulama kapansa bile sayaç bilgisi korunur. JSON yapısı sayesinde veri okunabilir ve genişletilebilir şekilde yapılandırılmıştır.

```
import socket

HOST = "127.0.0.1" #kendi receiver IP'n ile değiştir
PORT = 9000          # receiver.py zaten bu portta dinliyor

try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((HOST, PORT)) # TCP bağlantısı kur
    s.sendall(b"FAKE_PACKET_FROM_ATTACKER") # Sahte veri gönder
    s.close()
    print("[√] Sahte TCP bağlantısı kuruldu ve veri gönderildi.")
except Exception as e:
    print(f"[!] Bağlantı başarısız: {e}")
```

Kod Açıklaması:

Yukarıdaki kodda, sahte bir TCP istemcisi oluşturulmuştur. Bu istemci 127.0.0.1 adresinde 9000 numaralı porta bağlanarak sunucuya "FAKE_PACKET_FROM_ATTACKER" adlı sahte bir veri gönderir. Bu bağlantı, sunucunun şifre doğrulamasını atlayarak yetkisiz erişim denemelerini test etmek amacıyla gerçekleştirilmiştir. Eğer sunucu tarafındaki doğrulama mekanizması düzgün çalışıyorsa bu bağlantıyı reddeder ve IP adresini kaydeder. Bu tür testler, brute-force saldırularını veya bilinçli yetkisiz erişim girişimlerini analiz etmek için önemlidir.

```
PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> python receiver.py
[RTT] 0ms ölçüldü.
[ADAPT] RTT düşük, TCP modunda dinleniyor.
[+] Listening on 127.0.0.1:9000
[+] Connected by ('127.0.0.1', 61799)
[✗] Şifre hatalı! (1) → 127.0.0.1
PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> python receiver.py
[RTT] 0ms ölçüldü.
[ADAPT] RTT düşük, TCP modunda dinleniyor.
[+] Listening on 127.0.0.1:9000
[+] Connected by ('127.0.0.1', 61855)
[✗] Şifre hatalı! (1) → 127.0.0.1
PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> python receiver.py
[RTT] 0ms ölçüldü.
[ADAPT] RTT düşük, TCP modunda dinleniyor.
[+] Listening on 127.0.0.1:9000
[+] Connected by ('127.0.0.1', 61859)
[✗] Şifre hatalı! (2) → 127.0.0.1
PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> python receiver.py
[RTT] 0ms ölçüldü.
[ADAPT] RTT düşük, TCP modunda dinleniyor.
[+] Listening on 127.0.0.1:9000
[+] Connected by ('127.0.0.1', 61862)
[✗] Şifre hatalı! (3) → 127.0.0.1
[✗] 127.0.0.1 kalıcı olarak engellendi
PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> python receiver.py
[RTT] 0ms ölçüldü.
[ADAPT] RTT düşük, TCP modunda dinleniyor.
[+] Listening on 127.0.0.1:9000
[+] Connected by ('127.0.0.1', 61866)
[BLOCKED] Bağlantı reddedildi - engellenmiş IP: 127.0.0.1
PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje>
```

Şekil 16 – Şifre Doğrulama Hataları ve IP Engelleme Sistemi (TCP)

2.5.4. MITM ve Paket Yakalama Denemesi

Bu bölümde, sistemin aktarım sırasında üçüncü bir taraf (örneğin bir saldırgan) tarafından dinlenip dinlenmeyeceği test edilmiştir. Bunun için **Wireshark** aracı kullanılarak ağ trafigi yakalanmış ve analiz edilmiştir. Amaç, Man-in-the-Middle (MITM) saldırularına karşı sistemin dirençli olup olmadığını gözlemlemektir.

Test Senaryosu:

- secure_transfer.pcap ve udp_transfer.pcap dosyaları, dosya transferi sırasında kaydedilmiştir.
- Özellikle **TCP bağlantısı üzerinde gönderilen verilerin şifrelendiği** gözlemlenmiştir. Payload alanı okunabilir değildir.
- Bu durum, **AES/RSA şifreleme sisteminin etkili bir şekilde çalıştığını ve MITM saldırularına karşı koruma sağladığını** göstermektedir.
- Buna karşılık, **UDP ile gönderilen veriler şifrelenmeden** iletiliği için Wireshark ile kolayca okunabilmiştir.

```
PS C:\Users\erenra> & "C:\Program Files\Wireshark\tshark.exe" -i "\Device\NPF_Loopback" -f "tcp port 9000" -w "secure_transfer.pcap"
Capturing on 'Adapter for loopback traffic capture'
75
PS C:\Users\erenra>
```

Şekil 17 - Wireshark ile Şifreli Paket Dinleme Süreci

2.6. Arayüz Tespiti Modülü (iface_finder.py)

Bu modülde, sistemin doğru ağ arayüzü üzerinden çalışmasını sağlamak amacıyla, aktif IP adresi ve ona karşılık gelen fiziksel ağ arayüzü tespit edilmektedir.

İşleyiş Adımları:

- Mevcut ağ arayüzleri listelenir.
- Her arayüzün IP adresi sorgulanır.
- Yerel IP adresi ile eşleşen ağ arayüzü bulunarak kullanıcıya gösterilir.

```
# Aktif IP'yi bul
hostname = socket.gethostname()
local_ip = socket.gethostbyname(hostname)
print(f"[i] Yerel IP adresin: {local_ip}")

# IP ile eşleşen arayüzü bul
for iface in get_if_list():
    try:
        ip = get_if_addr(iface)
        if ip == local_ip:
            print(f"[✓] Bu senin aktif arayüzün olabilir: {iface}")
    except Exception:
        pass
```

Kod Açıklaması:

Sistemde aktif olan ağ arayüzü (interface) tespit etmek için kullanılır. Öncelikle, cihazın yerel IP adresi elde edilir. Ardından sistemdeki tüm ağ arayızları taranarak, IP adresi yerel IP ile eşleşen arayüz belirlenir. Bu arayüz, veri transfer işlemleri için kullanılacak doğru ağ yolunu temsil etmektedir. Kod, hata durumlarını yakalayarak işlemi kesintiye uğratmadan devam ettirecek şekilde yapılandırılmıştır.

```
● (base) PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> python iface_finder.py
[i] Yerel IP adresin: 192.168.56.1
[✓] Bu senin aktif arayüzün olabilir: \Device\NPF_{F6C392A5-D2DE-4A80-82A1-6EAEFFBAEDBD}
○ (base) PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje>
```

Şekil 18 - Aktif Ağ Arayüzü ve Yerel IP Adresinin Tespiti (iface_finder.py)

2.7. Log Kayıt Sistemi (log.txt)

Proje kapsamında dosya transfer sürecine ait tüm önemli olaylar merkezi bir log dosyasında (log.txt) tutulmaktadır. Bu sistem hem gönderici (sender) hem de alıcı (receiver) tarafında entegre edilmiştir.

- Tüm loglar tek bir dosyada tutulur: log.txt içerisinde zaman damgası ile birlikte hem UDP hem TCP transferlerine ait bilgiler, uyarılar ve hata mesajları yer alır.
- Zaman Damgası (Timestamp) eklidir: Her log satırı, olayın gerçekleştiği anı net bir şekilde belirtir.
- Dosya gönderim ve alım özeti loglanır: Gönderilen dosya adı, toplam parça sayısı, alınan dosya adı gibi bilgiler özet halinde kaydedilir.
- Parça ilerlemesi ve uyarılar da dahildir: Her başarılı gönderim, alınan parça sayısı, eksik veya bozuk veriler ayrı ayrı işlenip belirtilir.
- Sahte paket ve flood saldırıları gibi güvenlik olayları detaylı olarak kaydedilir.

```
[2025-06-08 12:51:07] [UDP] Chunk 9/9 gönderildi.  
[2025-06-08 12:51:07] [UDP] Chunk 9/9 alındı.  
[2025-06-08 12:51:07] [UDP] Dosya kaydedildi: received_udp_7_example.txt  
[2025-06-08 12:51:07] [SUMMARY] Alınan dosya: example.txt, Toplam chunk: 9  
[2025-06-08 12:51:07] [UDP] Tüm 9 parça gönderildi.  
[2025-06-08 12:51:07] [SUMMARY] UDP ile gönderilen dosya: example.txt, Parça sayısı: 9  
[2025-06-08 12:51:49] [CONNECT] Connected to 127.0.0.1:9000  
[2025-06-08 12:51:49] [AUTH] Authentication successful from ('127.0.0.1', 56841)  
[2025-06-08 12:51:49] [KEY] AES anahtarı RSA ile şifrelenip gönderildi.  
[2025-06-08 12:51:49] [INFO] Sending file: example.txt as example.txt  
[2025-06-08 12:51:49] [INFO] Total fragments: 13  
[2025-06-08 12:51:49] [FRAGMENT] Sent Fragment 0 successfully.  
[2025-06-08 12:51:49] [ADAPT] RTT: 0ms → Sleep: 0.010s  
[2025-06-08 12:51:49] [KEY] AES anahtarı RSA ile başarıyla çözüldü.
```

Şekil 19 - log.txt Üzerinden Dosya Transferi, Kimlik Doğrulama ve Şifreleme Süreci Kaydı

```
def log_message(msg, log_file="log.txt"):  
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")  
    with open(log_file, "a", encoding="utf-8") as f:  
        f.write(f"[{timestamp}] {msg}\n")  
  
    print("[UDP] Tüm parçalar gönderildi.")  
    log_message(f"[UDP] Tüm {total_chunks} parça gönderildi.")  
    log_message(f"[SUMMARY] UDP ile gönderilen dosya: {file_path}, Parça sayısı: {total_chunks}")  
  
    print(f"[UDP] Dosya kaydedildi: {save_path}")  
    log_message(f"[UDP] Dosya kaydedildi: {save_path}")  
    log_message(f"[SUMMARY] Alınan dosya: {filename}, Toplam chunk: {total_chunks}")
```

Kod Açıklaması:

Bu üç kod parçası, sistemin dosya transfer sürecini izlemek ve detaylı şekilde kayıt altına almak için oluşturulmuş gelişmiş bir loglama mekanizmasını göstermektedir. İlk kod parçasında tanımlanan log_message fonksiyonu, gelen mesajı zaman damgası (timestamp) ile birlikte log.txt adlı dosyaya ekler. Böylece olayların hangi tarihte ve saatte gerçekleştiği kayıt altına alınır. İkinci kod parçasında, UDP ile dosya gönderiminin tamamlandığına ve kaç parça (chunk) gönderildiğine dair bilgiler bu log fonksiyonu aracılığıyla hem sistem çıktısına hem de log dosyasına yazılır. Üçüncü kod parçasında ise alınan dosyanın başarıyla kaydedildiği ve toplam

kaç parça içerdiği bilgisi kaydedilir. Bu yapı, hem gönderici hem alıcı tarafında dosya hareketlerinin ve güvenlik olaylarının izlenmesini sağlar, hata tespiti ve performans değerlendirmesi gibi işlemleri kolaylaştırır.

3. SINIRLAMALAR VE GELİŞTİRME FİKİRLERİ

3.1. Ek Özellikler (Bonus Puanlar)

3.1.1. Hibrit TCP/UDP Geçişi (sender.py – receiver.py)

Bu bölümde ağ koşullarına göre transfer yönteminin dinamik olarak TCP veya UDP protokolüne yönlendirildiği hibrit bir yaklaşım sunulmuştur. Amaç, ağ yoğunluğu ve gecikme durumuna göre en uygun iletişim protokolünü seçerek aktarım verimliliğini artırmaktır.

Karar Mekanizması

- Sistemde get_rtt() fonksiyonu aracılığıyla hedef IP'ye yönelik RTT ölçülür.
- Ölçülen RTT değeri **100ms'den büyükse**, ağ yoğun kabul edilir ve aktarım **UDP** üzerinden yapılır.
- RTT değeri **100ms'den düşükse**, veri güvenliği ön planda tutularak aktarım **TCP (AES/RSA şifrelemeli)** yapılır.

```
def get_rtt(ip="127.0.0.1"):
    #return 150
    try:
        output = subprocess.check_output(["ping", "-n", "1", ip], universal_newlines=True)
        for line in output.splitlines():
            if "Average" in line:
                avg = int(line.split("Average = ")[-1].replace("ms", "")).strip()
                return avg
    except Exception as e:
        log_message(f"[WARN] RTT ölçülemedi, varsayılan gecikme kullanılacak: {e}")
    return 100
```

Kod Açıklaması:

Bu fonksiyon, verilen IP adresine ping komutu göndererek RTT (Round Trip Time) değerini milisaniye cinsinden ölçer. Ölçüm sonucu "Average" kelimesi içeren satırdan ayıklanır. Eğer ölçüm başarısız olursa, sistem varsayılan olarak 100 ms RTT değeri kullanır ve bu durum log dosyasına uyarı olarak yazılır. Böylece ağ durumu bilinmese bile sistem stabil çalışmaya devam eder. Aynı yapı hem receiver.py içinde hem de sender.py içinde bulunur.

```

if __name__ == "__main__":
    ip = '127.0.0.1'
    rtt = get_rtt(ip)
    print(f"[RTT] {rtt}ms ölçüldü.")
    if rtt > 100:
        print("[ADAPT] RTT yüksek, UDP moduna geçiliyor.")
        receive_file_udp('0.0.0.0')
    else:
        print("[ADAPT] RTT düşük, TCP modunda dinleniyor.")
        start_server(ip)

```

Kod Açıklaması:

Bu fonksiyon, RTT (Round Trip Time) ölçümüne göre hangi protokolün kullanılacağına karar verir. Eğer RTT 100ms'ten büyükse, sistem UDP ile daha hızlı gönderim yapar. RTT düşükse, güvenli ve şifreli gönderim için TCP tercih edilir. Böylece ağ koşullarına adaptif bir protokol seçimi gerçekleştirilmiş olur. Bu yapı, hibrit veri aktarım mimarisinin temelini oluşturur. Aynı karar mekanizması hem **gonderici (sender.py)** hem de **alıcı (receiver.py)** tarafında uygulanır. Böylece her iki uç birbirile uyumlu olarak aynı protokolde buluşur. Aksi takdirde biri UDP dinlerken diğer TCP ile gönderim yapabilir ve iletişim başarısız olur. Bu çift taraflı dinamik yapı sayesinde sistem senkronize çalışır ve bağlantı tutarlılığı korunur.

```

def send_file_udp(file_path, ip='127.0.0.1', port=9000):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    filename = os.path.basename(file_path).encode()
    with open(file_path, 'rb') as f:
        data = f.read()

    log_message("[UDP] Dosya gönderimine başlandı..")
    with open("transfer_log.txt", "a", encoding="utf-8") as log:
        log.write(f"[UDP] Gonderilen dosya: {file_path}, boyut: {len(data)} byte\n")

    max_chunk_size = 1400
    total_chunks = (len(data) + max_chunk_size - 1) // max_chunk_size

    for i in range(total_chunks):
        chunk = data[i*max_chunk_size : (i+1)*max_chunk_size]
        header = len(filename).to_bytes(4, 'big') + filename + i.to_bytes(4, 'big') + total_chunks.to_bytes(4, 'big')
        packet = header + chunk
        sock.sendto(packet, (ip, port))
        print(f"[UDP] Chunk {i+1}/{total_chunks} gönderildi..")
        log_message(f"[UDP] Chunk {i+1}/{total_chunks} gönderildi..")
        time.sleep(0.01)

    sock.close()
    print("[UDP] Tüm parçalar gönderildi..")
    log_message(f"[UDP] Tüm {total_chunks} parça gönderildi..")
    log_message(f"[SUMMARY] UDP ile gönderilen dosya: {file_path}, Parça sayısı: {total_chunks}")
    with open("transfer_log.txt", "a", encoding="utf-8") as log:
        log.write(f" UDP ile gönderilen toplam parça: {total_chunks}\n")

```

Kod Açıklaması (sender.py):

Bu fonksiyon, dosyayı UDP protokolü üzerinden parçalara ayırarak alıcıya gönderir.

- Dosya ilk olarak open() komutu ile okunur ve baytlara çevrilir.
- Ardından maksimum 1400 byte'lık parçalara bölünür.
- Her parçaya, dosya adı, parça ID'si ve toplam parça sayısı gibi bilgiler içeren özel bir başlık (header) eklenir.
- Oluşturulan her paket socket.sendto() fonksiyonu ile gönderilir.

- Gönderim işlemleri hem ekrana yazdırılır hem de transfer_log.txt dosyasına detaylı olarak loglanır.
- Gönderim sonunda, toplam gönderilen parça sayısı özet olarak kaydedilir.

Bu yapı sayesinde UDP ile parça tabanlı dosya aktarımı yapılabilir, ayrıca saldırısı tespiti ve analiz için kayıtlar tutulur.

```

def receive_file_udp(ip='0.0.0.0', port=9000):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.bind((ip, port))
    print(f"[UDP] Dinleniyor: {ip}:{port}")

    data_chunks = {}
    filename = None
    total_chunks = None

    while True:
        try:
            data, addr = sock.recvfrom(2048)
            ip, port = addr
            now = time.time()

            if ip not in ip_first_time:
                ip_first_time[ip] = now

            ip_chunk_counter[ip] += 1

            duration = now - ip_first_time[ip]
            if ip_chunk_counter[ip] >= FLOOD_LIMIT and duration < 10:
                log_message(f"[THREAT] UDP flood şüphesi: {ip} → {ip_chunk_counter[ip]} chunk / {duration:.2f}s")
                print(f"⚠️ UDP flood şüphesi: {ip}")

            # SAHTE PAKET (too short) tespiti
            if len(data) < 12:
                log_message(f"[THREAT] Geçersiz UDP paketi: {ip} → Yetersiz veri ({len(data)} bayt)")
                print(f"⚠️ Geçersiz UDP paketi tespit edildi → {ip}")
                continue

            if not data:
                continue

            name_len = int.from_bytes(data[:4], 'big')
            filename = data[4:4+name_len].decode()
            chunk_id = int.from_bytes(data[4+name_len:8+name_len], 'big')
            total_chunks = int.from_bytes(data[8+name_len:12+name_len], 'big')
            chunk = data[12+name_len:]

            data_chunks[chunk_id] = chunk

            print(f"[UDP] Chunk {chunk_id+1}/{total_chunks} alındı..")
            log_message(f"[UDP] Chunk {chunk_id+1}/{total_chunks} alındı..")

            if len(data_chunks) == total_chunks:
                break

        except Exception as e:
            print("[] UDP alma hatası:", e)
            log_message(f"[ERROR] UDP alma hatası: {e}")
            break

    sock.close()

    if filename:
        # UDP dosya ismi otomatikleştir
        base_name = f"received_udp_{filename}"
        counter = 1
        while os.path.exists(f"received_udp_{counter}_{filename}"):
            counter += 1
        save_path = f"received_udp_{counter}_{filename}"

        with open(save_path, 'wb') as f:
            for i in sorted(data_chunks.keys()):
                f.write(data_chunks[i])

        print(f"[UDP] Dosya kaydedildi: {save_path}")
        log_message(f"[UDP] Dosya kaydedildi: {save_path}")
        log_message(f"[SUMMARY] Alınan dosya: {filename}, Toplam chunk: {total_chunks}")
        with open("transfer_log.txt", "a", encoding="utf-8") as log:
            log.write(f"[UDP] Alınan dosya: {filename}, Toplam chunk: {total_chunks}\n")

```

Kod Açıklaması (receiver.py):

receive_file_udp() fonksiyonu, UDP üzerinden gönderilen parçalı verilerin güvenli, sıralı ve bütünlüklü şekilde alıcı (receiver) tarafından alınmasını sağlar. Aynı zamanda olası saldıruları (flood, sahte paket) algılayarak loglamaya yönelik temel bir IDS (Intrusion Detection System) davranışını sergiler.

1. Bağlantı Kurulumu ve Soket Hazırlığı:

Fonksiyon, 9000 numaralı port üzerinde UDP soketini başlatır ve tüm IP'lerden veri alacak şekilde dinlemeye başlar. Her gelen paket, recvfrom() fonksiyonu ile alınır ve kaynak IP ile zamanı kaydedilir.

2. Saldırı Tespiti: UDP Flood ve Sahte Paket Kontrolü

Bu bölümde sistem, kötü niyetli davranışlara karşı proaktif kontrol sağlar:

- | | | |
|--|-------------|------------|
| • UDP | Flood | Saldırısı: |
| Aynı IP'den 10 saniyeden kısa sürede aşırı sayıda veri gelmesi durumunda sistem bu IP'yi log dosyasına "[THREAT] UDP flood şüphesi" mesajıyla kaydeder. Bu durum gerçek zamanlı olarak terminale de uyarı mesajı olarak basılır. | | |
| • Geçersiz Paket | (Sahte UDP) | Tespiti: |
| 12 byte'tan küçük gelen her veri paketi geçersiz olarak değerlendirilir. Bu tür paketler, genellikle eksik başlığa veya zararlı içeriğe sahip olabilir. Sistem bu tür paketleri işlemeye devam etmez, log'a yazıp geçer. | | |

3. Veri Parçalama ve Parça Toplama Mekanizması:

Gönderilen veri, önceden belirlenmiş chunk boyutlarına bölünmüştür. Her gelen parça:

- İlk 4 byte'ında dosya adı uzunluğu,
- Sonraki kısmında dosya adı,
- Daha sonra fragment (chunk) numarası ve toplam fragment sayısı,
- Geri kalanında ise parça verisi taşırl.

Bu bilgiler ayırtıldıktan sonra data_chunks isimli sözlükte fragment ID'sine göre tutulur. Aynı zamanda alınan her parça için terminale:

[UDP] Chunk X/Y alındı.

şeklinde görsel çıktı ve log_message() ile log dosyasına kayıt yapılır.

4. Parçaların Birleştirilmesi ve Dosyanın Kaydedilmesi:

Tüm chunk sayısı tamamlandığında, döngü sonlandırılır. Kaydetme işlemi başlamadan önce:

- Aynı isimde dosya varsa üzerine yazmamak için received_udp_1_example.txt, received_udp_2_example.txt gibi bir ad üretimi yapılır.
- Veriler chunk_id sırasına göre birleştirilerek diske yazılır.

Bu işlem sonunda terminale ve log dosyasına şu bilgiler yazılır:

- Kaydedilen dosya adı,
- Alınan toplam chunk sayısı,

- Zaman bilgisi (log'da tarih-saat damgasıyla).

Sonuç:

Bu fonksiyon sadece veri alımı yapmaz; aynı zamanda:

- UDP flood saldırısını,
- Geçersiz paketleri,
- Chunk eksikliğini,
- İkinci kez gelen paketleri tespit ederek terminal ve log dosyasına yansıtır.

Bu yapısıyla hem veri transferini hem de temel seviyede güvenlik kontrolünü tek bir modül üzerinden gerçekleştirerek UDP'nin zayıf yönlerine karşı güçlü bir çözüm sunar.

3.1.2. Dinamik Tıkanıklık Kontrolü (sender.py)

Sistem, gönderim sırasında anlık RTT ölçümüne göre gecikme süresini dinamik olarak ayarlayarak tıkanıklık kontrolü sağlar. RTT yüksekse her fragment arasında bekleme süresi artar, düşükse azaltılır. Bu mekanizma sayesinde hem ağın aşırı yüklenmesi engellenir hem de veri kaybı riski azaltılır.

```
[2025-06-07 21:14:36] [ADAPT] RTT: 0ms → Sleep: 0.010s
[2025-06-07 21:14:36] [FRAGMENT] Fragment 8 received and decrypted successfully.
[2025-06-07 21:14:36] [FRAGMENT] Sent Fragment 8 successfully.
[2025-06-07 21:14:36] [ADAPT] RTT: 0ms → Sleep: 0.010s
[2025-06-07 21:14:36] [FRAGMENT] Fragment 9 received and decrypted successfully.
[2025-06-07 21:14:36] [FRAGMENT] Sent Fragment 9 successfully.
```

Şekil 20 – RTT Değerine Göre Dinamik Gecikme Uygulaması

```
rtt = get_rtt(ip)
delay = min(max(rtt / 1000, 0.01), 0.3)
log_message(f"[ADAPT] RTT: {rtt}ms → Sleep: {delay:.3f}s")
time.sleep(delay)
```

Kod Açıklaması:

Bu kod parçası, gönderim öncesinde RTT (Round Trip Time) ölçümüne göre bekleme süresi (sleep) belirler. Hesaplama formülü:

$$\text{delay} = \min(\max(\text{rtt} / 1000, 0.01), 0.3)$$

Bu formül sayesinde:

- Çok düşük RTT'lerde sistemin aşırı hızlı davranışması için minimum 0.01 saniye gecikme uygulanır.
- Yüksek RTT'lerde sistemin aşırı yavaşlamaması için maksimum 0.3 saniyelik üst sınır tanımlanır.

`time.sleep(delay)` ile sistem bu süre kadar bekletilir. Bu yöntem, ağın yoğun olduğu durumlarda gönderimi yavaşlatır; ağ boşken ise hızlı çalışmasına olanak sağlar. Böylece dinamik tıkanıklık kontrolü gerçekleştirilmiş olur.

3.1.3. Gelişmiş Saldırı Simülasyonları ve Gerçek Zamanlı Paket Filtreleme (receiver.py)

Bu bölümde sistemin kötü niyetli paketlere karşı gösterdiği savunma yetenekleri test edilmiştir. Projede **real-time packet filtering** (gerçek zamanlı paket filtreleme) ve **intrusion detection** (saldırı tespiti) mekanizmaları başarıyla entegre edilmiştir.

Uygulanan Senaryolar:

1. UDP Flood Saldırısı:

- `fake_udp_flood.py` dosyası ile aynı IP'den çok hızlı şekilde 50 adet sahte UDP chunk gönderilmiştir.
- Receiver tarafında bu paketler takip edilip IP başına düşen fragment sayısı ve süre izlenerek, belirlenen `FLOOD_LIMIT` aşılırsa saldırı olarak işaretlenmiş ve loglanmıştır (örn: “⚠️ UDP flood şüphesi: 127.0.0.1”).

```
[...] [UDP] Chunk 18/50 alındı.  
[UDP] Chunk 19/50 alındı.  
[⚠️] UDP flood şüphesi: 127.0.0.1  
[✖️] 127.0.0.1 flood şüphesi nedeniyle paket işlenmedi ve receiver kapatılıyor.  
PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje>
```

Şekil 21 – Flood tespiti sonucu receiverin kapanması

```
ip_chunk_counter = defaultdict(int)  
ip_first_time = {}  
FLOOD_LIMIT = 20 # aynı IP'den gelen max chunk sayısı  
  
duration = now - ip_first_time[ip]  
if ip_chunk_counter[ip] >= FLOOD_LIMIT and duration < 10:  
    log_message(f"[THREAT] UDP flood şüphesi: {ip} → {ip_chunk_counter[ip]} chunk / {duration:.2f}s")  
    print(f"[⚠️] UDP flood şüphesi: {ip}")  
    print(f"[✖️] {ip} flood şüphesi nedeniyle paket işlenmedi ve receiver kapatılıyor.")  
    exit()
```

Kod Açıklaması:

Bu iki kod parçası birlikte çalışarak UDP flood saldırısını tespit etmeyi ve sistemi korumayı amaçlar. İlk olarak `ip_chunk_counter` adlı bir sözlük kullanılarak her IP adresinden gelen veri parçası (chunk) sayısı takip edilir. `ip_first_time` sözlüğü ise her IP'nin ilk bağlantı zamanını

kaydeder. FLOOD_LIMIT sabiti, bir IP'nin kısa sürede gönderebileceği maksimum chunk sayısını belirtir (örneğin 20).

İkinci görseldeki kod blogunda ise bir IP adresinden gelen chunk sayısı, belirlenen FLOOD_LIMIT değerini aşıyorsa ve bu işlemler 10 saniyeden kısa sürede gerçekleşiyse, bu durum potansiyel bir UDP flood saldırısı olarak değerlendirilir. Bu durumda, sistem bir uyarı mesajı verir, tehdidi log dosyasına kaydeder ve exit() komutuyla alıcı (receiver) programını güvenlik amacıyla kapatır. Bu yapı, gerçek zamanlı saldırısı tespiti ve müdahale için temel bir güvenlik önlemi sağlar.

2. Geçersiz Paket Tespiti:

- fake_udp_invalid_packet.py dosyası ile 12 byte'tan kısa (geçersiz) veri gönderilmiştir.
- Receiver, len(data) < 12 kontrolüyle bu paketleri anında fark etmiş ve "Geçersiz UDP paketi tespit edildi" uyarısı ile işlem yapmadan filtrelemiştir.
- [4 byte] → filename_len
- [0+ byte] → filename (boş olabilir ama alan ayrıılır)
- [4 byte] → chunk_id
- [4 byte] → total_chunks

```
PS C:\Ders\3.sinif_2.donem\Bilgisayar_Aglari\Proje\Proje> python receiver.py
[RTT] 150ms ölçüldü.
[ADAPT] RTT yüksek, UDP moduna geçiliyor.
[UDP] Dinleniyor: 0.0.0.0:9000
[!] Geçersiz UDP paketi tespit edildi → 127.0.0.1
[!] Geçersiz UDP paketi tespit edildi → 127.0.0.1
```

Şekil 22 - Geçersiz UDP paketlerinin reddedilmesi

```
# SAHTE PAKET (too short) tespiti
if len(data) < 12:
    log_message(f"[THREAT] Geçersiz UDP paketi: {ip} → Yetersiz veri ({len(data)} bayt)")
    print(f"[!] Geçersiz UDP paketi tespit edildi → {ip}")
    exit()
```

Kod Açıklaması:

Bu kod parçası, sahte veya eksik formatlı UDP paketlerini tespit etmek için kullanılır. UDP üzerinden gelen verinin (data) boyutu 12 bayttan küçükse, bu paket geçersiz sayılır. Bunun nedeni, geçerli bir veri paketinin en az 12 baytlık bir yapıya sahip olması gerektidir (örneğin: filename_len, chunk_id, total_chunks gibi alanlar). Eğer bu koşul sağlanmazsa, sistem bir tehdit mesajı loglar, ekrana uyarı verir ve ardından exit() komutu ile alıcı (receiver) uygulamasını güvenlik amacıyla sonlandırır. Bu, sistemin düşük boyutlu ve potansiyel olarak zararlı UDP paketlerine karşı korunmasını sağlar.

Sistem Tepkileri:

- Tehdit olarak işaretlenen IP'ler loglara yazılmıştır.
- Paketler kabul edilmeden doğrudan filtrelenmiş, sistemin bozulmasının önüne geçirilmiştir.

- Normal veri akışında herhangi bir kesinti yaşanmamıştır.

Sonuç:

Bu iki test ile birlikte sistemin ağ seviyesinde saldırısı türlerini tanıma ve etkisiz hale getirme becerisi doğrulanmıştır. Hem **doğrulama kuralları**, hem de **paket başlığı uzunluk ve hız analizleri**, saldırısı tespitinde başarıyla görev almıştır.

4. SONUÇ

Bu proje kapsamında, veri güvenliği, bütünlük doğrulama, ağ trafiği analizi ve düşük seviyeli IP başlık manipülasyonlarını kapsayan gelişmiş bir dosya transfer sistemi başarıyla geliştirilmiştir. Sistem, veri iletimi sırasında **AES-256** algoritmasıyla her dosya parçasını ayrı ayrı şifrelemekte; şifreleme anahtarları ise **RSA-2048** algoritması ile güvenli şekilde alıcıya iletilmektedir. Ayrıca her parça, **SHA-256** algoritmasıyla hash'lenerek veri bütünlüğü korunmuştur.

Dosya gönderimi sırasında **dinamik RTT ölçümü** yapılarak bağlantı kalitesi değerlendirilmekte ve **TCP/UDP geçiği** otomatik olarak sağlanmaktadır. Böylece sistem, ağ koşullarına adapte olan **hibrit bir iletim modeli** sunmaktadır. Ayrıca **paket kaybı ve tıkanıklık kontrolü** için temel seviyede **trafik izleme ve sınırlama** mekanizmaları da entegre edilmiştir.

Proje kapsamında yalnızca uygulama katmanı güvenliği değil, aynı zamanda **ağ katmanı seviyesi** de ele alınmıştır. Scapy kütüphanesiyle **IP başlık alanları (TTL, DF, Checksum, Fragment Offset)** manuel olarak ayarlanmış, bu paketler Wireshark ile analiz edilerek doğruluğu görselleştirilmiştir. Böylece öğrenciye **düşük seviyeli protokol manipülasyonu** konusunda uygulamalı deneyim kazandırılmıştır.

Geliştirilen sistemde ayrıca **saldırı tespiti ve engellemeye mekanizmaları** yer almaktadır. Özellikle **UDP flood** ve **geçersiz paket** gibi saldırılarda gerçek zamanlı algılanıp loglanmakta, gerektiğinde bağlantı sonlandırılarak sistem korunmaktadır. **Tshark** ve **Wireshark** kullanılarak yapılan kayıtlar, saldırının analizine olanak sağlamıştır.

Sonuç olarak sistem, **güvenli dosya iletimi, düşük seviyeli IP işlemleri, saldırısı tespiti ve performans adaptasyonu** gibi birçok kritik bileşenin entegre bir yapı sunmaktadır. Proje, ileri düzeyde ağ güvenliği, protokol mühendisliği ve sistem tasarımları gibi konularda hem teorik hem de pratik açıdan değerli bir çalışma ortaya koymuştur. Elde edilen altyapı, ileride GUI entegrasyonu, kullanıcı yönetimi ve gerçek zamanlı analiz gibi yeni modüllerle kolayca genişletilebilecek esnekliktedir.

4.1. Video ve Github

Projenin tüm aşamalarını ve işleyişini açıklayan tanıtım videosuna aşağıdaki bağlantıdan ulaşabilirsiniz:

[Proje Tanıtım Videosu \(YouTube\)](#)

Projenin kaynak kodlarına, dökümantasyonuna ve görsellere aşağıdaki GitHub bağlantısı üzerinden erişebilirsiniz:

[GitHub Proje Sayfası](#)

5. KAYNAKLAR

- Scapy Documentation. (n.d.). Retrieved April 28, 2025, from <https://scapy.readthedocs.io/>
- Daemen, J., & Rijmen, V. (2002). *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer-Verlag.
- Rivest, R., Shamir, A., & Adleman, L. (1978). *A method for obtaining digital signatures and public-key cryptosystems*. Communications of the ACM, 21(2), 120–126.
- Python Software Foundation. (n.d.). *socket — Low-level networking interface*. Retrieved April 28, 2025, from <https://docs.python.org/3/library/socket.html>
- Wireshark Foundation. (n.d.). *Wireshark User Guide*. Retrieved April 28, 2025, from https://www.wireshark.org/docs/wsug_html_chunked/
- National Institute of Standards and Technology (NIST). (2001). *Specification for the Advanced Encryption Standard (AES)*. FIPS Publication 197. <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>
- Stallings, W. (2017). *Cryptography and Network Security: Principles and Practice* (7th ed.). Pearson.
- Kaufman, C., Perlman, R., & Speciner, M. (2002). *Network Security: Private Communication in a Public World* (2nd ed.). Prentice Hall.
- Wireshark University. (n.d.). *Analyzing TCP/IP and UDP Protocols Using Wireshark*. Retrieved April 28, 2025, from <https://www.wireshark.org/>
- Wikipedia Contributors. (n.d.). *SHA-2*. Wikipedia. Retrieved April 28, 2025, from <https://en.wikipedia.org/wiki/SHA-2>