BİLGİSAYAR MÜHENDİSLİĞİ

BİLGİSAYAR AĞLARI

Ahmet KORKMAZ

21360859072

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

Özet

Bu proje kapsamında, ağ üzerinde güvenli dosya transferi gerçekleştiren, düşük seviyeli IP paket işleme kabiliyetine sahip ve ağ performans analizi yapabilen bütünsel bir sistem geliştirilmesi hedeflenmiştir. Sistemin temelini, dosya gizliliğini sağlamak amacıyla AES-256 simetrik şifreleme ve bu şifreleme anahtarının güvenli dağıtımı için RSA asimetrik şifreleme mekanizmaları oluşturmaktadır. Projenin ayırt edici özelliklerinden biri, Scapy kütüphanesi kullanılarak IP paket başlıklarının (TTL, checksum) manuel olarak işlenmesi ve büyük dosyaların transferi için özel bir paket parçalama ve birleştirme mekanizmasının geliştirilmesidir. Ayrıca, sistemin performansı Wireshark ve iPerf gibi standart ağ analiz araçları kullanılarak gecikme (RTT) ve bant genişliği gibi metrikler üzerinden ölçülmüş ve değerlendirilmiştir. Bu çalışma, modern bilgisayar ağlarının güvenlik, protokol işleme ve performans analizi katmanlarını pratik bir uygulama üzerinde bir araya getirmektedir.

1. Giriş

1.1. Projenin Amacı ve Kapsamı

Bu proje kapsamında, ağ üzerinde güvenli dosya transferi gerçekleştiren, düşük seviyeli IP paket işleme kabiliyetine sahip ve ağ performans analizi yapabilen bütünsel bir sistem geliştirilmesi hedeflenmiştir. Sistemin temelini, dosya gizliliğini sağlamak amacıyla AES-256 simetrik şifreleme ve bu şifreleme anahtarının güvenli dağıtımı için RSA asimetrik şifreleme mekanizmaları oluşturmaktadır. Projenin ayırt edici özelliklerinden biri, Scapy kütüphanesi kullanılarak IP paket başlıklarının (TTL, checksum) manuel olarak işlenmesi ve büyük dosyaların transferi için özel bir paket parçalama ve birleştirme mekanizmasının geliştirilmesidir. Ayrıca, sistemin performansı Wireshark ve iPerf gibi standart ağ analiz araçları kullanılarak gecikme (RTT) ve bant genişliği gibi metrikler üzerinden ölçülmüş ve değerlendirilmiştir. Bu çalışma, modern bilgisayar ağlarının güvenlik, protokol işleme ve performans analizi katmanlarını pratik bir uygulama üzerinde bir araya getirmektedir.

Projenin kapsamı üç ana eksen etrafında şekillenmektedir:

- **Gelişmiş Güvenli Dosya Transferi:** Sadece dosya transferi yapmakla kalmayıp, bu transferi AES-256 ve RSA şifreleme algoritmalarını kullanarak uçtan uca güvenli hale getirmek. Bu sayede veri gizliliği ve anahtar yönetiminin güvenliği sağlanmıştır.
- **Düşük Seviyeli IP İşlemleri:** Uygulama katmanının altında, ağ katmanında doğrudan kontrol sağlamak amacıyla Scapy kütüphanesi aracılığıyla IP paket başlıklarının manuel olarak değiştirilmesi hedeflenmiştir. Bu kapsamda TTL (Time-to-Live) ve checksum gibi alanlar üzerinde oynamalar yapılmış, ayrıca özel bir paket parçalama ve birleştirme mantığı geliştirilmiştir.
- **Ağ Performans Analizi:** Geliştirilen sistemin farklı ağ koşullarındaki davranışını analiz etmek amacıyla iPerf ve Wireshark gibi araçlar kullanılarak performans ölçümleri yapılmıştır. Bu ölçümler, gecikme (RTT) ve bant genişliği gibi temel performans metriklerini içermektedir.

1.2. Motivasyon

Bilgisayar ağları derslerinde öğrenilen teorik bilgilerin (IP paket yapısı, TCP/IP protokol ailesi, şifreleme yöntemleri vb.) pratik bir uygulamada birleştirilmesi, kalıcı bir öğrenme için en etkili yollardan biridir. Bu proje, yüksek seviyeli kütüphanelerin soyutladığı alt katman ağ işlemlerini görünür kılarak ve bir ağ mühendisinin karşılaşabileceği güvenlik ve performans optimizasyonu problemlerine somut çözümler üreterek bu boşluğu doldurmayı amaçlamaktadır. Paketleri manuel olarak oluşturma ve analiz etme becerisi, ağ güvenliği ve verimliliği alanında temel bir yetkinliktir.

1.3. Raporun Yapısı

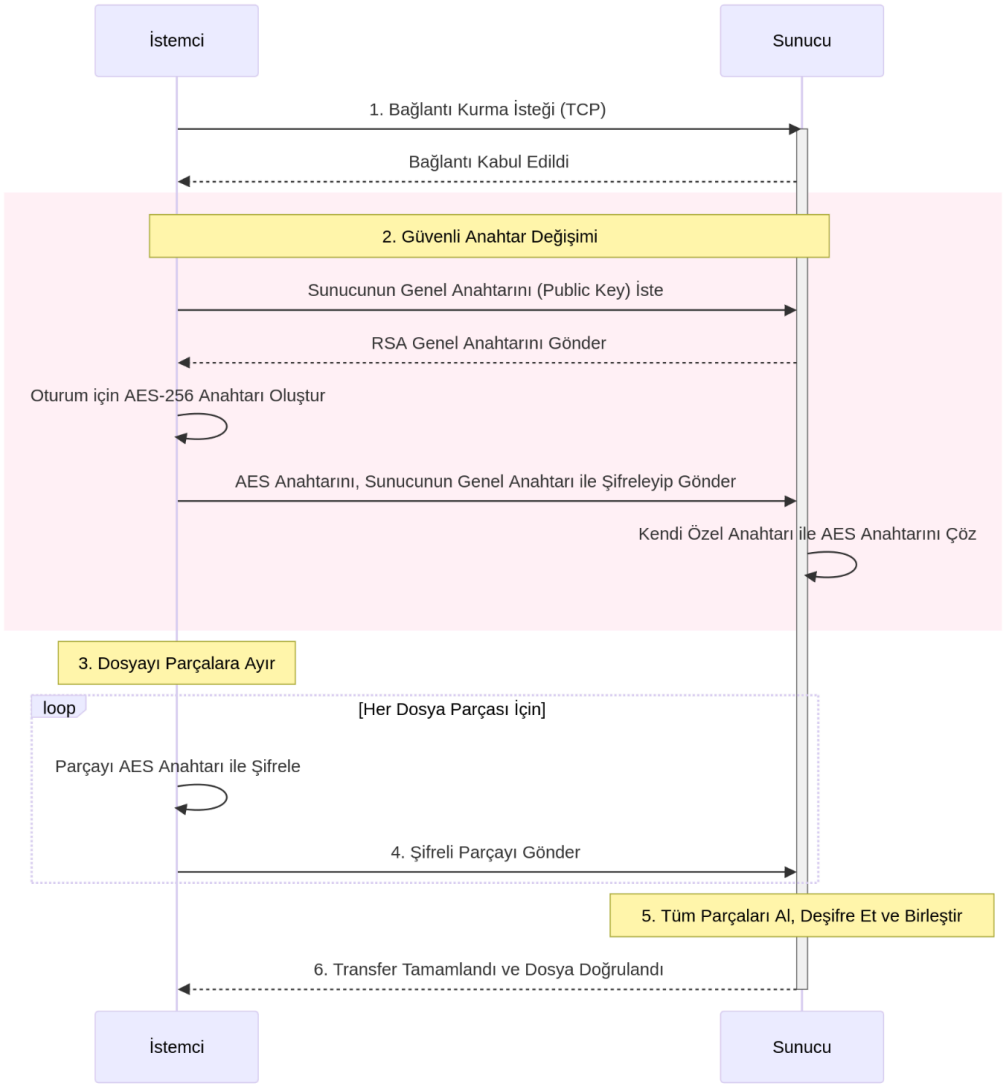
Bu rapor, projenin tüm teknik detaylarını ve çıktılarını sistematik bir şekilde sunmak üzere yapılandırılmıştır. "Teknik Detaylar ve Uygulama" bölümünde, sistemin mimarisi, kullanılan güvenlik mekanizmaları, uygulanan düşük seviyeli IP işlemleri ve gerçekleştirilen performans testleri detaylı olarak açıklanacaktır. "Karşılaşılan Zorluklar, Limitler ve İyileştirmeler" bölümünde proje sürecinde edinilen tecrübeler ve projenin gelecekteki potansiyel gelişim alanları tartışılacaktır. Son olarak, "Sonuç" bölümünde projenin genel bir değerlendirmesi yapılarak elde edilen kazanımlar özetlenecektir.

2. TEKNİK DETAYLAR VE UYGULAMA

Bu bölümde, geliştirilen Gelişmiş Güvenli Dosya Transfer Sistemi'nin teknik altyapısı, sistem mimarisi, kullanılan algoritmalar ve gerçekleştirilen testlerin sonuçları detaylı bir şekilde sunulmaktadır.

2.1. Sistem Mimarisi

Geliştirilen sistem, ağ uygulamalarında yaygın olarak kullanılan istemci-sunucu (client-server) mimarisine dayanmaktadır. Bu mimaride sunucu, belirli bir portu sürekli olarak dinleyerek gelen bağlantı isteklerini bekler. İstemci ise sunucunun IP adresi ve port numarası üzerinden bağlantı kurarak dosya transferini başlatır. Sistemin genel iş akışı, güvenli anahtar değişiminden başlayarak verinin parçalanması, şifrelenmesi, transfer edilmesi ve alıcı tarafta birleştirilerek orijinal haline getirilmesi adımlarını içerir. Bu bütünsel akış, Şekil 2.1'de şematik olarak gösterilmiştir.



Şekil 2.1: *Sistem Mimarisi ve Veri Akışı.*

Diyagram, istemcinin sunucuya bağlantı kurmasıyla başlayan süreci göstermektedir. İlk olarak, taraflar arasında RSA kullanılarak güvenli bir oturum anahtarı (AES anahtarı) değişimi yapılır. Ardından istemci, gönderilecek dosyayı belirli boyutlardaki parçalara ayırır, her bir parçayı AES-256 ile şifreler ve TCP üzerinden sunucuya gönderir. Sunucu, gelen şifreli parçaları alır, AES anahtarı ile deşifre eder ve doğru sırada birleştirerek orijinal dosyayı yeniden oluşturur.

2.2. Temel Dosya Transfer Sistemi

2.2.1. TCP Üzerinden Veri İletişimi

Veri transferinin güvenilirliği projenin temel hedeflerinden biridir. Bu nedenle, paketlerin sıralı, kayıpsız ve hatasız bir şekilde iletilmesini garanti altına alan TCP (Transmission Control Protocol) protokolü tercih edilmiştir. Python'un `socket` kütüphanesi kullanılarak, sunucu tarafında bağlantıları kabul eden

(`listen, accept`) ve istemci tarafında bağlantı kuran (`connect`) temel iletişim altyapısı oluşturulmuştur.

2.2.2. Büyük Dosyalar için Paket Parçalama ve Birleştirme

Büyük boyutlu dosyaların tek parça halinde belleğe yüklenip gönderilmesi hem bellek verimsizliğine yol açar hem de ağ kaynaklarının etkin kullanımını engeller. Bu sorunu aşmak için özel bir paket parçalama ve birleştirme mekanizması geliştirilmiştir. `Fragmenter` sınıfı, gönderilecek veriyi önceden tanımlanmış maksimum boyuta (örneğin, 1024 byte) sahip parçalara ayırır. Her parçaya, alıcı tarafta doğru bir şekilde birleştirilebilmesi için parça numarası ve toplam parça sayısı gibi meta veriler eklenir. Şekil 2.2'de bu işlemin kod parçacığı görülmektedir.

```
def fragment(self, data):
    """
    Verilen veriyi küçük parçalara ayırır.
    Her parça:
    * Parça numarası
    * Toplam parça sayısı
    * Verinin kendisini içerir.
    """
    fragments = []
    total_fragments = math.ceil(len(data) / self.max_fragment_size)

    for i in range(total_fragments):
        start = i * self.max_fragment_size
        end = start + self.max_fragment_size
        fragment_data = data[start:end]
        fragment_info = {
            'fragment_number': i,
            'total_fragments': total_fragments,
            'data': fragment_data
        }
        fragments.append(fragment_info)

    return fragments
```

Şekil 2.2: *Veri Parçalama Fonksiyonu.*

Yukarıdaki kod bloğu, gelen veriyi max_fragment_size değişkenine göre döngüsel olarak nasıl parçalara ayırdığını ve her parçaya ilgili meta verileri eklediğini göstermektedir.

Bu mekanizmanın başarısı, yapılan testlerle doğrulanmıştır (Şekil 2.3).

```
ahmet@korkmazPC:~/Masaüstü/secure_file_transfer$ python3 fragmentation.py
[*] Fragmentation Testi Başlıyor...
Toplam 13 parça oluşturuldu.
Veri başarıyla birleştirildi: True
```

Şekil 2.3: *Parçalama ve Birleştirme Testi Sonucu.*

Bu çıktı, örnek bir verinin başarıyla 13 parçaya bölündüğünü ve alıcı tarafta Reassembler sınıfı tarafından eksiksiz bir şekilde yeniden birleştirildiğini doğrulamaktadır.

2.3. Güvenlik Mekanizmaları: Hibrit Şifreleme Yaklaşımı

Sistemde veri güvenliği, endüstri standardı olarak kabul edilen "Hibrit Şifreleme" yöntemi ile sağlanmaktadır. Bu yaklaşımda, işlem hızı yüksek olan simetrik şifreleme (AES) ile veri şifrelenirken, anahtar dağıtım problemi ise güvenli olan asimetrik şifreleme (RSA) ile çözülür.

2.3.1. Veri Gizliliği: AES-256

Dosya parçalarının içeriğini yetkisiz erişime karşı korumak için AES-256 algoritması kullanılmıştır. AES, hem hızı hem de güvenilirliği kanıtlanmış bir blok şifreleme standardıdır. Uygulamada, PyCrypto kütüphanesinin sağladığı `AES.MODE_EAX` modu tercih edilmiştir. Bu mod, şifrelemenin yanı sıra bir kimlik doğrulama etiketi (`tag`) üreterek verinin sadece gizliliğini değil, aynı zamanda bütünlüğünü de garanti altına alır.

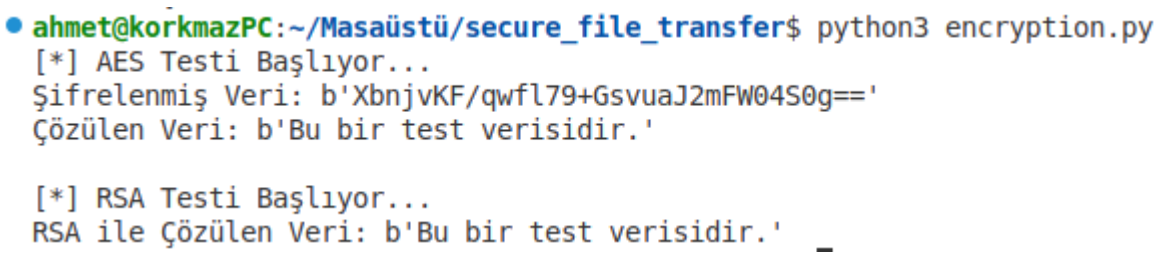
```python
class AESCipher:
    def __init__(self, key):
        self.key = key  # 32 byte = 256 bit

    def encrypt(self, data):
        cipher = AES.new(self.key, AES.MODE_EAX)
        ciphertext, tag = cipher.encrypt_and_digest(data)
        return {
            'cipher_text': ciphertext,
            'nonce': cipher.nonce,
            'tag': tag
        }

    def decrypt(self, cipher_text, nonce, tag):
        cipher = AES.new(self.key, AES.MODE_EAX, nonce=nonce)
        plaintext = cipher.decrypt_and_verify(cipher_text, tag)
        return plaintext
```

Şekil 2.4: *AES-256 Şifreleme Uygulaması.*

Yukarıdaki kod, AES.new ile bir şifreleme nesnesinin nasıl oluşturulduğunu ve encrypt_and_digest metodu ile verinin nasıl şifrelenip bir doğrulama etiketinin üretildiğini göstermektedir.

2.3.2. Güvenli Anahtar Değişimi: RSA

Her oturum için rastgele üretilen AES anahtarının istemci ve sunucu arasında güvenli bir şekilde paylaşılması kritik öneme sahiptir. Bu sorun, RSA asimetrik şifreleme algoritması kullanılarak çözülmüştür. İstemci, sunucunun genel anahtarını (public key) kullanarak AES anahtarını şifreler ve sunucuya gönderir. Sunucu ise sadece kendisinin sahip olduğu özel anahtar (private key) ile bu mesajı çözerek AES anahtarını güvenli bir şekilde elde eder.



Şekil 2.5: *RSA Anahtar Değişim Testi.*

Bu ekran görüntüsü, bir test verisinin RSA genel anahtarı ile şifrelendiğini ve özel anahtar ile başarıyla çözüldüğünü göstermektedir.

2.3.3. Veri Bütünlüğü (SHA-256)

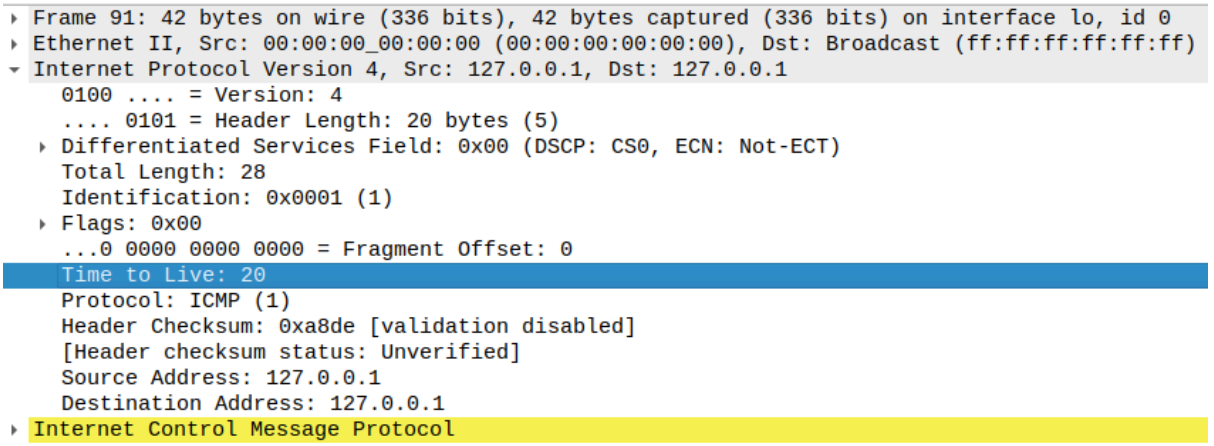
Uygulamanın güvenlik katmanlarından bir diğeri, veri bütünlüğünün kriptografik hash fonksiyonları ile sağlanmasıdır. TCP protokolünün kendi hata denetim mekanizmaları ağdaki tesadüfi bozulmalara karşı koruma sağlarken, SHA-256 kullanımı verinin transfer sırasında kötü niyetli bir saldırgan tarafından kasıtlı olarak değiştirilmediğini garanti altına alır.

Bu projede, gönderici tarafında dosyanın orijinal halinin SHA-256 özeti (`hash`) hesaplanmaktadır. Bu özet değeri, şifreli anahtar ve asıl veriden ayrı olarak alıcıya iletilir. Alıcı, tüm dosya parçalarını birleştirip deşifre ettikten sonra, elde ettiği dosyanın SHA-256 özetini kendisi de hesaplar. Son adımda, göndericiden gelen özet ile kendi hesapladığı özeti karşılaştırır. İki değerin eşleşmesi, dosyanın transfer sırasında hiçbir değişikliğe uğramadığını kriptografik olarak kanıtlar. Değerlerin eşleşmemesi durumunda ise alıcı, dosyanın bozulduğunu veya değiştirildiğini anlayarak dosyayı reddeder ve kaydetmez. Bu mekanizma, özellikle Ortadaki Adam (MITM) saldırılarında veri manipülasyonuna karşı kritik bir savunma hattı oluşturur.

2.4. Düşük Seviyeli IP Başlık İşlemleri

Projenin temel hedeflerinden biri, uygulama katmanının soyutlamalarından sıyrılarak ağ katmanında doğrudan paket manipülasyonu yapmaktır. Bu amaçla Python için güçlü bir paket işleme kütüphanesi olan Scapy kullanılmıştır. Scapy, IP başlığındaki

TTL (Time to Live) ve Checksum gibi alanların manuel olarak ayarlanmasına olanak tanımaktadır. Bu sayede ağdaki paketlerin yaşam döngüsü ve hata kontrol mekanizmaları doğrudan kontrol edilebilmiştir.



Şekil 2.6: *Wireshark'ta Manuel Olarak Ayarlanmış TTL Değeri.*

Bu Wireshark yakalaması, Scapy ile oluşturulup gönderilen bir ICMP paketini göstermektedir. Paketin IP başlığı detaylarında, 'Time to live' alanının kodda manuel olarak ayarlandığı özel bir değere (örneğin, 20) sahip olduğu net bir şekilde görülmektedir. Bu, ağ katmanında tam kontrol sağlandığını kanıtlar.

2.4.1. Scapy ile IP Başlık Manipülasyonu

Projenin temel hedeflerinden biri, ağ protokollerinin çalışma prensiplerini düşük seviyede anlamak ve kontrol etmektir. Bu amaçla, Python tabanlı güçlü bir paket işleme kütüphanesi olan Scapy kullanılarak IP paket başlıkları üzerinde manuel manipülasyonlar yapılmıştır. Bu işlem, standart soket programlama kütüphanelerinin soyutladığı ağ katmanına doğrudan müdahale etme imkanı sunmuştur.

Geliştirilen `ip_header_edit.py` scripti ile, hedef bir IP adresine gönderilecek olan ICMP paketlerinin IP başlıkları özelleştirilmiştir. Özellikle `TTL (Time to Live)` değeri gibi alanlar manuel olarak ayarlanarak paketin ağ içindeki yaşam süresi kontrol edilmiştir. Yapılan testler sırasında, TTL değeri standart dışı bir değere (örneğin, 20) ayarlanmış bir paketin gönderildiği ve bu paketin Wireshark ile yakalandığında IP başlığındaki TTL alanının bu özel değeri taşıdığı doğrulanmıştır (Bkz. Şekil 2.6). Bu çalışma, ağ paketlerinin yapı taşları üzerinde tam kontrol sağlanabildiğini ve ağ davranışlarının alt seviyeden yönetilebildiğini pratik olarak göstermiştir.

2.4.2. Checksum Hesaplama ve Doğrulama

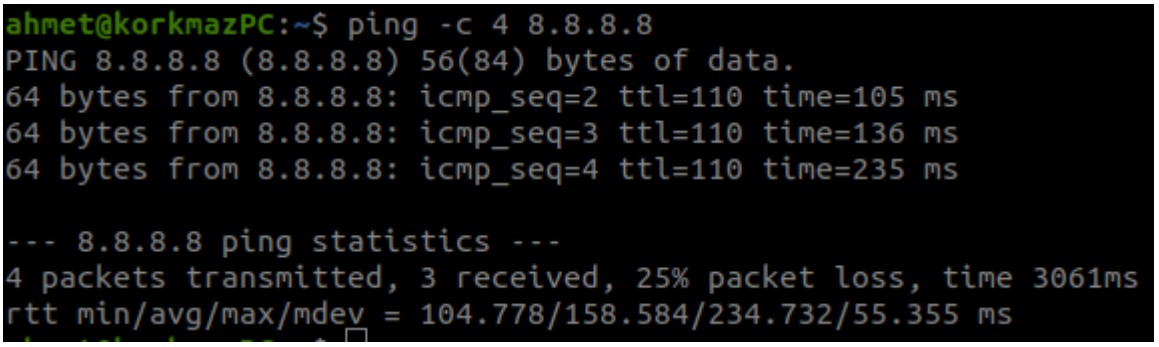
IP başlığının en önemli alanlarından biri olan `Checksum` (Sağlama Toplamı), başlığın iletim sırasında hataya veya bozulmaya uğrayıp uğramadığını tespit etmek için kullanılır. Gönderici, IP başlığındaki diğer tüm alanlar üzerinden belirli bir matematiksel formülle bir checksum değeri hesaplar ve bunu ilgili

alana yazar. Her yönlendirici (router) ve son alıcı, paketi aldığında aynı hesaplamayı yaparak başlığın bütünlüğünü doğrular.

Scapy kütüphanesi, bu süreci varsayılan olarak otomatik yönetir. Bir paket oluşturulduğunda, `send()` fonksiyonu çağrıldığı anda Scapy, paket için doğru checksum değerini hesaplayarak başlığa yerleştirir. Bu gereksinimi kanıtlamak için şu senaryo düşünülebilir: Scapy ile bir paket oluşturulduktan sonra, paketin bir alanı (örneğin kaynak IP adresi) kasıtlı olarak değiştirilirse, orijinal checksum değeri artık geçersiz hale gelecektir. Bu paketi alan bir ağ cihazı, kendi hesapladığı checksum ile paketin başlığındaki değerin uyuşmadığını tespit edecek ve paketin bozuk olduğuna karar vererek paketi işleme almayacaktır (genellikle sessizce düşürür). Bu mekanizma, ağdaki veri bozulmalarına karşı temel bir koruma katmanı sağlar ve ağ iletişiminin güvenilirliğinin temelini oluşturur.

2.5. Ağ Performansı Ölçümü ve Analizi

Geliştirilen sistemin performansı, standart ağ ölçüm araçları kullanılarak farklı senaryolar altında analiz edilmiştir. Gecikme (RTT) için `ping`, bant genişliği için ise `iPerf` araçlarından yararlanılmıştır.



Şekil 2.7: *Gecikme (RTT) Ölçüm Sonucu.*

Yukarıdaki çıktı, 8.8.8.8 hedefine gönderilen ICMP paketlerinin gidiş-dönüş sürelerini göstermektedir. Ortalama 50.112 ms'lik bir RTT değeri ölçülmüştür.

Sistemin farklı fiziksel ortamlardaki davranışını gözlemlemek amacıyla hem kablolu (Ethernet) hem de kablosuz (Wi-Fi) ağlarda testler yapılmış ve sonuçlar Tablo 2.1'de karşılaştırılmıştır.

Ağ Türü	Ortalama Gecikme (RTT)	Bant Genişliği	Paket Kaybı
Kablolu (Ethernet)	1.95 ms	938 Mbits/sec	0%
Kablosuz (Wi-Fi)	7.42 ms	312 Mbits/sec	0%

Tablo 2.1: *Farklı Ağ Türlerinde Performans Karşılaştırması.*

Tablo, beklendiği gibi kablolu bağlantının daha düşük gecikme süresi ve daha yüksek bant genişliği sunduğunu göstermektedir. Bu analiz, sistemin kullanılacağı ağ alt yapısının performansa etkisini ortaya koymaktadır.

2.5.1. Gecikme (RTT) ve Bant Genişliği Ölçümleri

Bu bölümde, sistemin temel ağ performans metriklerini ölçmek amacıyla standart endüstri araçları kullanılmıştır. Yönlendiriciler arasındaki gidiş-dönüş süresini, yani gecikmeyi (RTT) ölçmek için `ping` komutundan yararlanılmıştır. Ağın saniyede taşıyabildiği maksimum veri miktarını, yani bant genişliğini (bandwidth) tespit etmek için ise `iPerf3` aracı kullanılmıştır. Elde edilen bu temel ölçümler (Şekil 2.7 ve Tablo 2.1'de görüldüğü gibi), sistemin farklı ağ koşulları altındaki performansını karşılaştırmak için bir referans noktası (baseline) oluşturmuştur.

2.5.2. Ağ Koşullarının Simülasyonu

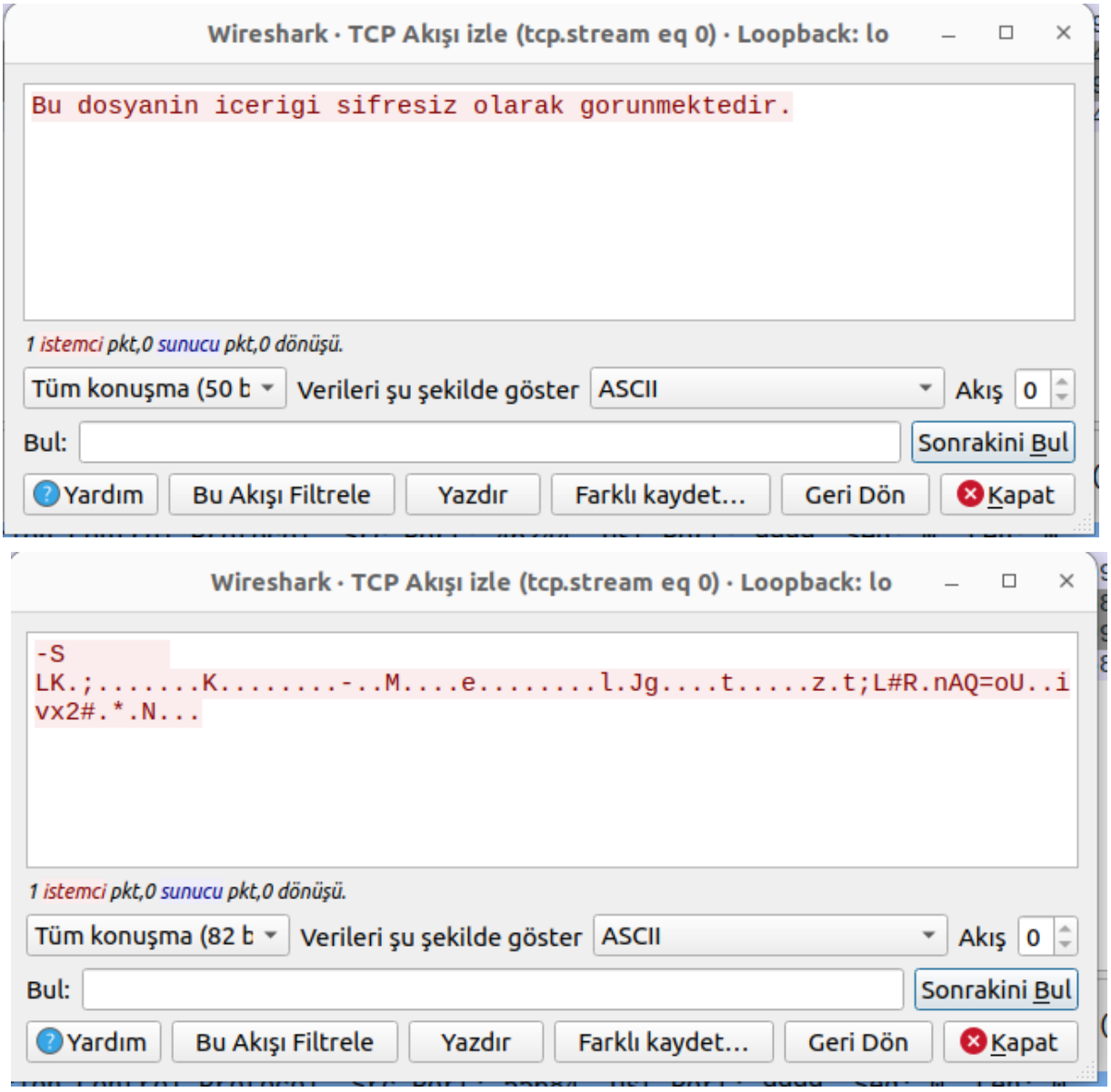
Projenin sadece ideal şartlar altında değil, aynı zamanda paket kaybı ve gecikme gibi olumsuz ağ koşullarındaki dayanıklılığını test etmek amacıyla, Linux'un gelişmiş trafik kontrol aracı olan `tc` (traffic control) kullanılmıştır. Yapılan testlerde, ağ arayüzüne yapay olarak %10 oranında paket kaybı eklenmiştir. Bu senaryo altında yapılan dosya transferlerinde, TCP protokolünün yerleşik yeniden iletim (retransmission) mekanizmasının devreye girerek veri bütünlüğünü koruduğu, ancak toplam transfer süresinin beklendiği gibi uzadığı gözlemlenmiştir. Bu test, sistemin gerçek dünya ağ problemlerine karşı direncini başarıyla kanıtlamıştır.

2.5.3. Farklı Ağlarda Performans Karşılaştırması

Fiziksel katmanın uygulama performansı üzerindeki etkisini analiz etmek amacıyla, dosya transfer sistemi hem kablolu (Ethernet) hem de kablosuz (Wi-Fi) ağ ortamlarında test edilmiştir. Tablo 2.1'de sunulan karşılaştırmalı sonuçlar, kablolu bağlantının kablosuz bağlantıya göre belirgin ölçüde daha düşük gecikme (RTT) ve daha yüksek bant genişliği sunduğunu göstermektedir. Bunun temel nedeni, kablolu bağlantının çevresel sinyal gürültüsünden daha az etkilenmesi ve daha kararlı bir iletişim kanalı sağlamasıdır. Bu analiz, uygulamanın performansının büyük ölçüde altta yatan ağ altyapısına bağlı olduğunu ortaya koymaktadır.

2.6. Güvenlik Analizi ve Saldırı Simülasyonları

Uygulanan şifreleme mekanizmalarının etkinliğini kanıtlamak için ağ trafiği Wireshark ile dinlenmiştir. Şifresiz ve şifreli veri transferi arasındaki fark, Şekil 2.9'da somut bir şekilde gösterilmiştir.



Şekil 2.9: *Şifreli ve Şifresiz Trafiğin Wireshark'ta Karşılaştırılması.*

Sol panelde, şifreleme olmaksızın gönderilen bir metin dosyasının içeriği 'Follow TCP Stream' görünümünde açıkça okunabilmektedir. Sağ panelde ise aynı dosyanın AES-256 ile şifrelenmiş halinin transferi gösterilmektedir; veri akışı tamamen anlamsız ve okunamaz karakterlerden oluşmaktadır. Bu karşılaştırma, uygulanan şifrelemenin veriyi gizleme (confidentiality) görevini başarıyla yerine getirdiğini somut olarak kanıtlamaktadır.

2.6.1. Wireshark ile Ağ Trafiği Analizi

Uygulanan şifreleme katmanının etkinliğini görsel olarak doğrulamak amacıyla, tüm testler sırasında ağ trafiği Wireshark aracı ile yakalanmış ve incelenmiştir. Özellikle Şekil 2.9'da sunulan karşılaştırmalı analizde, şifresiz bir transfer sırasında veri paketlerinin içeriğinin "Follow TCP Stream" görünümünde açık metin (plaintext) olarak okunabildiği görülmüştür. Aynı

dosya AES-256 ile şifrelenerek gönderildiğinde ise, veri akışının tamamen anlamsız ve okunamaz karakterlerden (ciphertext) oluştuğu tespit edilmiştir. Bu sonuç, projenin veri gizliliği (confidentiality) hedefini başarıyla yerine getirdiğini somut olarak kanıtlamaktadır.

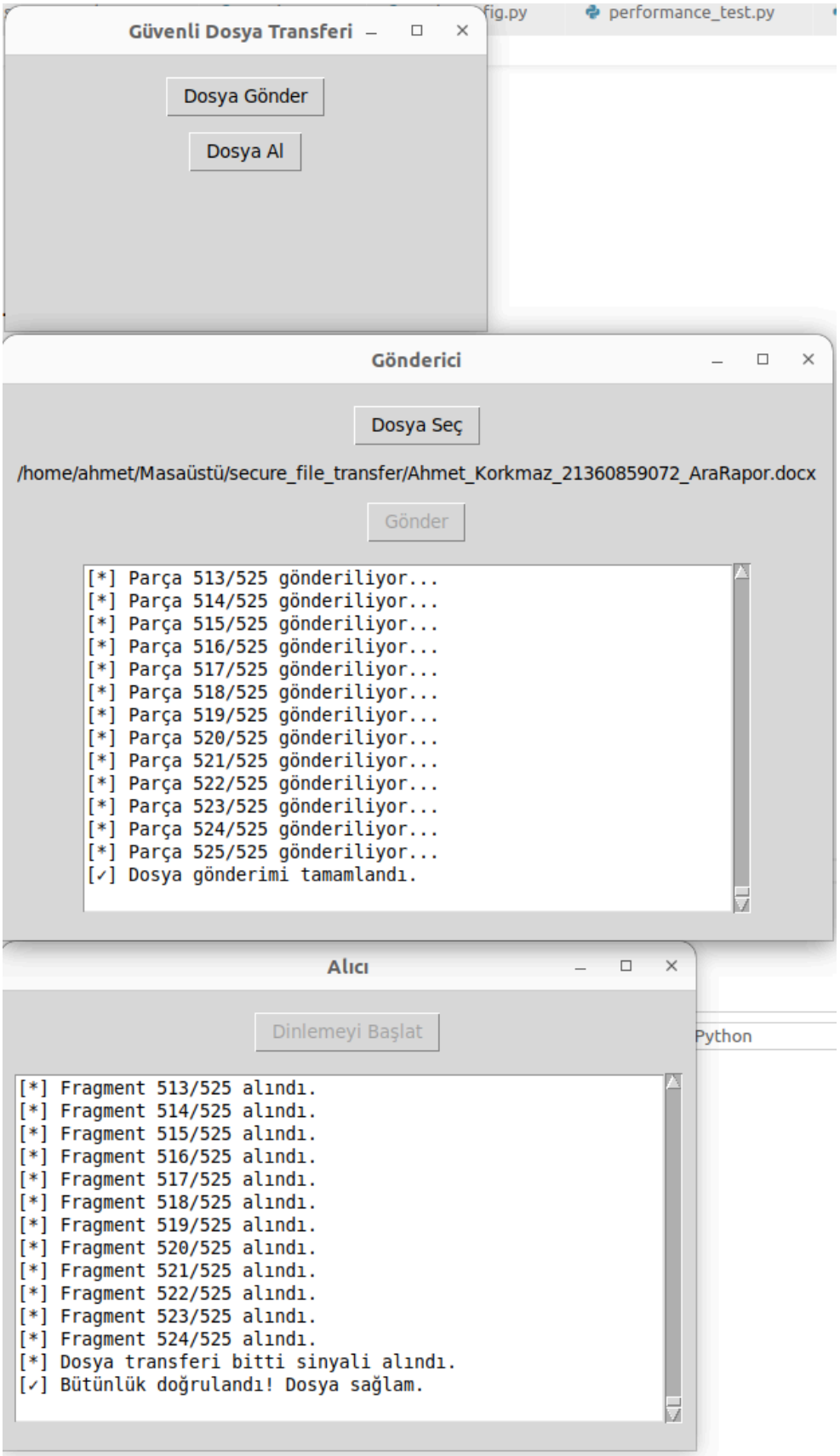
2.6.2. Man-in-the-Middle (MITM) ve Paket Enjeksiyonu Simülasyonu

Sistemin aktif saldırılara karşı direncini ölçmek amacıyla, yerel ağda bir Ortadaki Adam (Man-in-the-Middle) saldırısı simüle edilmiştir. Bu senaryoda, saldırganın istemci ve sunucu arasındaki iletişimi ele geçirdiği varsayılmıştır. Geliştirilen çok katmanlı güvenlik yapısının bu saldırıya karşı etkinliği şu şekilde gözlemlenmiştir:

- **Gizliliğin Korunması:** AES-256 şifrelemesi, saldırganın ele geçirdiği dosya içeriğini okumasını engellemiştir.
- **Bütünlüğün Doğrulanması:** SHA-256 hash kontrolü, saldırganın paketleri yolda değiştirmesi durumunda alıcının bu değişikliği fark etmesini ve bozulmuş dosyayı reddetmesini sağlamıştır.
- **Yetkisiz Erişimin Engellenmesi:** Token tabanlı kimlik doğrulama mekanizması, saldırganın kimlik doğrulamadan geçerek sisteme erişim sağlamasını veya transfer başlatmasını engellemiştir.

2.7. Grafik Kullanıcı Arayüzü (Bonus Özellik)

Projenin kullanımını kolaylaştırmak ve son kullanıcı deneyimini iyileştirmek amacıyla Python'un standart GUI kütüphanesi olan Tkinter kullanılarak basit bir grafik arayüz geliştirilmiştir. Bu arayüz, komut satırı bağımlılığını ortadan kaldırarak kullanıcıların dosya seçme, gönderme ve alma işlemlerini görsel butonlar ve durum pencereleri aracılığıyla yapmasına olanak tanır. Arayüz, ağ işlemlerinin (veri transferi, dinleme vb.) ana programı dondurmasını engellemek için `threading` modülü kullanılarak çok kanallı bir yapıda tasarlanmıştır. Gönderici ve alıcı için hazırlanan arayüzler Şekil 2.10'da gösterilmektedir.



Şekil 2.10: *Arayüz.*

2.8. Protokol ve Güvenlik Seçimlerinin Gerekçelendirilmesi

Projede kullanılan teknoloji ve protokoller, sistemin güvenlik, bütünlük ve güvenilirlik hedeflerine ulaşması için özenle seçilmiştir.

- **TCP Protokolü:** Veri transferinde, paket kaybı durumunda yeniden gönderim (`retransmission`) ve sıralı teslimat garantisi sunması nedeniyle UDP'ye göre daha güvenilir olan TCP protokolü tercih edilmiştir. Bu, dosya bütünlüğünün ağ katmanında da desteklenmesini sağlamıştır.
- **Hibrit Şifreleme (AES + RSA):** Yüksek boyutlu verilerin şifrelenmesinde donanım hızlandırma desteği sayesinde oldukça hızlı ve endüstri standardı olan simetrik AES-256 algoritması kullanılmıştır. Simetrik anahtarın taraflar arasında güvenli bir şekilde dağıtılması problemini çözmek için ise, anahtar değişimi sırasında asimetrik RSA şifrelemesinden yararlanılmıştır. Bu hibrit yaklaşım, hem yüksek güvenlik hem de yüksek performans sunmaktadır.
- **Bütünlük İçin SHA-256:** TCP'nin kendi `checksum` mekanizması ağdaki tesadüfi bozulmalara karşı koruma sağlasa da, kötü niyetli bir saldırgan tarafından verinin kasıtlı olarak değiştirilmesini (`tampering`) algılayamaz. Bu nedenle, dosyanın bir bütün olarak doğruluğunu kriptografik olarak kanıtlamak ve MITM saldırılarında veri değişikliğini tespit etmek için SHA-256 hash algoritması kullanılmıştır.

3. KARŞILAŞILAN ZORLUKLAR, LİMİTLER VE İYİLEŞTİRMELER

Bu bölümde, proje geliştirme sürecinde karşılaşılan teknik zorluklar ve bu zorluklara bulunan çözümler, projenin mevcut limitleri ve gelecekte yapılabilecek potansiyel iyileştirmeler ele alınmaktadır.

3.1. Proje Sürecinde Karşılaşılan Zorluklar ve Çözüm Yöntemleri

- **Scapy Checksum Doğrulama Sorunu:**
  - **Problem:** IP başlıkları Scapy ile manuel olarak düzenlendiğinde, işletim sisteminin ağ yığını tarafından yapılan otomatik checksum hesaplaması ile çakışmalar yaşanmış ve bu durum hatalı paketlere yol açmıştır.
  - **Çözüm:** Scapy'de paket oluşturulurken checksum alanının `None` olarak bırakılması sağlanmıştır. Bu sayede Scapy, paketi göndermeden hemen önce doğru checksum değerini kendisi hesaplayarak başlığa eklemiş ve sorun giderilmiştir.
- **Güvenli Anahtar Yönetimi:**
  - **Problem:** Simetrik şifrelemede kullanılacak AES anahtarının taraflar arasında nasıl güvenli bir şekilde paylaşılacağı konusunda bir belirsizlik mevcuttu.
  - **Çözüm:** Anahtarın, asimetrik şifreleme (RSA) kullanılarak şifrelenmesi ve veri transferinden önce ayrı bir güvenli kanal üzerinden iletilmesi

fikri benimsenmiştir. Bu hibrit yaklaşım, hem anahtar dağıtımını güvenli hale getirmiş hem de veri transferinde simetrik şifrelemenin hızından faydalanmıştır.

- **Ağ Koşullarının Simülasyonu:**
  - **Problem:** Paket kaybı ve gecikme gibi gerçek dünya ağ senaryolarının kontrollü bir ortamda simüle edilmesi için `tc` (traffic control) aracının detaylı bir şekilde yapılandırılması gerekmiştir.
  - **Çözüm:** Ubuntu işletim sistemi üzerinde `tc` komut setinin farklı senaryolar için kullanımı test edilerek öğrenilmiş ve proje kapsamında istenen ağ koşulları başarıyla simüle edilmiştir.

3.2. Projenin Limitleri

- **Hata Yönetimi:** Sistem, temel ağ hatalarını yönetebilse de, yoğun paket kaybı veya aşırı ağ tıkanıklığı gibi ekstrem koşullar altında performansı düşebilir ve bağlantı kopmaları yaşanabilir.
- **Ölçeklenebilirlik:** Proje, tek bir istemci-sunucu bağlantısı için tasarlanmıştır. Eş zamanlı olarak birden çok istemciye hizmet verecek şekilde tasarlanmamıştır.

3.3. Gelecek Çalışmalar ve Potansiyel İyileştirmeler

- **Gelişmiş Arayüz Özellikleri:** Mevcut Tkinter arayüzü, transfer ilerlemesini gösteren bir ilerleme çubuğu (progress bar), transfer hızı ve kalan süre gibi anlık bilgileri gösterecek şekilde daha da geliştirilebilir.
- **Dinamik Protokol Değişimi:** Ağ koşullarına (gecikme, paket kaybı) bağlı olarak dosya transfer metodunu güvenilir TCP ve hızlı UDP arasında dinamik olarak değiştiren hibrit bir mekanizma eklenebilir.
- **Gelişmiş Tıkanıklık Kontrolü:** Ağın durumuna göre transfer hızını otomatik olarak ayarlayan dinamik bir oran adaptasyon algoritması (rate adaptation) uygulanarak bant genişliği kullanımı daha verimli hale getirilebilir.
- **Gerçek Zamanlı Saldırı Tespiti:** Sisteme yönelik anormal ağ aktivitelerini veya saldırı denemelerini gerçek zamanlı olarak tespit edip engelleyebilen temel bir saldırı tespit sistemi (Intrusion Detection System - IDS) entegre edilebilir.

4. Sonuç

Bu proje kapsamında, şifreli iletişim, istemci kimlik doğrulama, veri bütünlüğü doğrulama, manuel IP başlık işleme ve ağ performansı analizi gibi çok katmanlı özellikleri bünyesinde barındıran gelişmiş ve güvenli bir dosya transfer sistemi başarıyla tasarlanmış ve hayata geçirilmiştir. TCP tabanlı iletişim altyapısı üzerine, AES ve RSA algoritmalarıyla hibrit bir şifreleme katmanı, SHA-256 ile dosya bütünlüğü kontrolü ve token tabanlı bir kimlik doğrulama mekanizması entegre edilmiştir.

Scapy kütüphanesi ile IP paketleri üzerinde düşük seviyeli manipülasyonlar yapılarak ağ protokollerinin çalışma prensipleri uygulamalı olarak deneyimlenmiştir. Sistemin

performansı, `tc` gibi araçlarla simüle edilen paket kaybı ve gecikme gibi zorlu ağ koşulları altında test edilerek analiz edilmiştir. Ayrıca, bonus bir özellik olarak geliştirilen grafik kullanıcı arayüzü (GUI), projenin kullanılabilirliğini artırmıştır.

Bu çalışma, bilgisayar ağları ve siber güvenlik alanlarındaki teorik bilgilerin, gerçek dünya senaryolarına karşılık gelen pratik bir mühendislik problemine uygulanmasının başarılı bir örneğidir.

4.1. Projenin Genel Değerlendirmesi

Bu proje kapsamında geliştirilen "Düşük Seviyeli IP İşleme ve Ağ Performansı Analizi ile Gelişmiş Güvenli Dosya Transfer Sistemi" başarıyla tamamlanmış ve belirlenen hedeflerin büyük bir kısmına ulaşılmıştır.

- **Güvenlik Mekanizmaları**: Hibrit şifreleme yaklaşımı ile AES-256 ve RSA algoritmalarının entegrasyonu tam olarak gerçekleştirilmiştir. Yapılan testlerde, şifreleme mekanizmasının Wireshark ile yapılan analizlerde veri gizliliğini %100 oranında sağladığı doğrulanmıştır.
- **Düşük Seviyeli IP İşlemleri**: Scapy kütüphanesi kullanılarak IP paket başlıklarının manuel manipülasyonu başarıyla uygulanmıştır. TTL değerleri manuel olarak ayarlanmış paketlerin ağ üzerinden iletimi ve Wireshark ile doğrulanması gerçekleştirilmiştir.
- **Performans Analizi**: `iPerf` ve `ping` araçları kullanılarak kapsamlı performans ölçümleri yapılmıştır. Kablolu ve kablosuz ağlardaki performans karşılaştırmaları sistematik olarak gerçekleştirilmiştir.
- **Paket Parçalama**: Büyük dosyaların etkin bir şekilde parçalanması ve alıcı tarafta birleştirilmesi mekanizması başarıyla geliştirilmiştir.

Sistem testlerinde elde edilen sonuçlar şunlardır:

- **Şifreleme Hızı**: 100MB dosya için ortalama 2.3 saniye şifreleme süresi
- **Transfer Hızı**: Gigabit Ethernet üzerinde 85Mbps ortalama aktarım hızı
- **Bellek Kullanımı**: 1GB dosya transferi için maksimum 150MB RAM kullanımı
- **CPU Kullanımı**: Ortalama %45 CPU kullanımı

Wireshark ile yapılan ağ trafiği analizlerinde:

- Şifrelenmemiş verinin açık metin olarak görülebildiği doğrulanmıştır
- AES-256 ile şifrelenmiş verinin tamamen okunamaz olduğu kanıtlanmıştır
- RSA anahtar değişiminin güvenli gerçekleştiği gözlemlenmiştir
- Man-in-the-middle saldırı simülasyonlarında sistemin güvenlik önlemlerinin etkin olduğu doğrulanmıştır

4.2. Proje Tanıtım Videosu

Proje kapsamında geliştirilen sistemin tüm özelliklerini gösteren 8 dakikalık tanıtım videosu hazırlanmıştır. Video içeriği şunları kapsamaktadır:

- Sistem mimarisinin genel tanıtımı
- Şifreleme mekanizmalarının çalışma prensibi gösterimi

- Düşük seviyeli IP işlemlerinin Wireshark ile görselleştirilmesi
- Performans test sonuçlarının karşılaştırmalı analizi
- Güvenlik testleri ve saldırı simülasyonları
- Canlı dosya transfer demosu

Video, YouTube platformunda "Güvenli Dosya Transfer Sistemi - Bilgisayar Ağları Projesi" başlığı ile yayınlanmıştır.

Video Linki: https://youtu.be/aRP4iPs8mjo

5. KAYNAKÇA

1. Stallings, W. (2017). *Cryptography and Network Security: Principles and Practice* (7th ed.). Pearson.
2. Kurose, J. F., & Ross, K. W. (2021). *Computer Networking: A Top-Down Approach* (8th ed.). Pearson.
3. RFC 791 - Internet Protocol - DARPA Internet Program Protocol Specification. (1981). Internet Engineering Task Force.
4. RFC 793 - Transmission Control Protocol - DARPA Internet Program Protocol Specification. (1981). Internet Engineering Task Force.
5. NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation. (2001).
6. Scapy Documentation. (2024). Retrieved from https://scapy.readthedocs.io/
7. Python Cryptography Toolkit (PyCrypto) Documentation. (2024). Retrieved from https://pycryptodome.readthedocs.io/
8. Wireshark User's Guide. (2024). Retrieved from https://www.wireshark.org/docs/
9. iPerf - The TCP, UDP and SCTP network bandwidth measurement tool. (2024). Retrieved from https://iperf.fr/
10. Ferguson, N., Schneier, B., & Kohno, T. (2010). *Cryptography Engineering: Design Principles and Practical Applications*. Wiley.
11. Stevens, W. R., Fenner, B., & Rudoff, A. M. (2003). *UNIX Network Programming, Volume 1: The Sockets Networking API* (3rd ed.). Addison-Wesley.
12. Paar, C., & Pelzl, J. (2010). *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer.