

BURSA TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
BİLGİSAYAR AĞLARI DERSİ DÖNEM PROJESİ FİNAL RAPORU

**Python ile Geliştirilen Temel Güvenlikli Dosya Paylaşım Sistemi:
Şifreleme, Paket Parçalama ve Ağ Trafiği Analizi Uygulaması**

Adile AKKILIÇ

21360859052

2024-2025 BAHAR DÖNEMİ

Doç. Dr. İzzet Fatih ŞENTÜRK

İÇİNDEKİLER

1. Giriş
2. Teknik Detaylar
 - 2.1 Sistem Mimarisi
 - 2.2 Uygulama Yapısı
 - 2.3 Test Senaryoları ve Sonuçlar
 - 2.4 Güvenlik İncelemesi
3. Sınırlılıklar ve İyileştirmeler
 - 3.1 Sınırlılıklar
 - 3.2 İyileştirme Önerileri
4. Sonuç
5. Kaynakça
6. Ekler

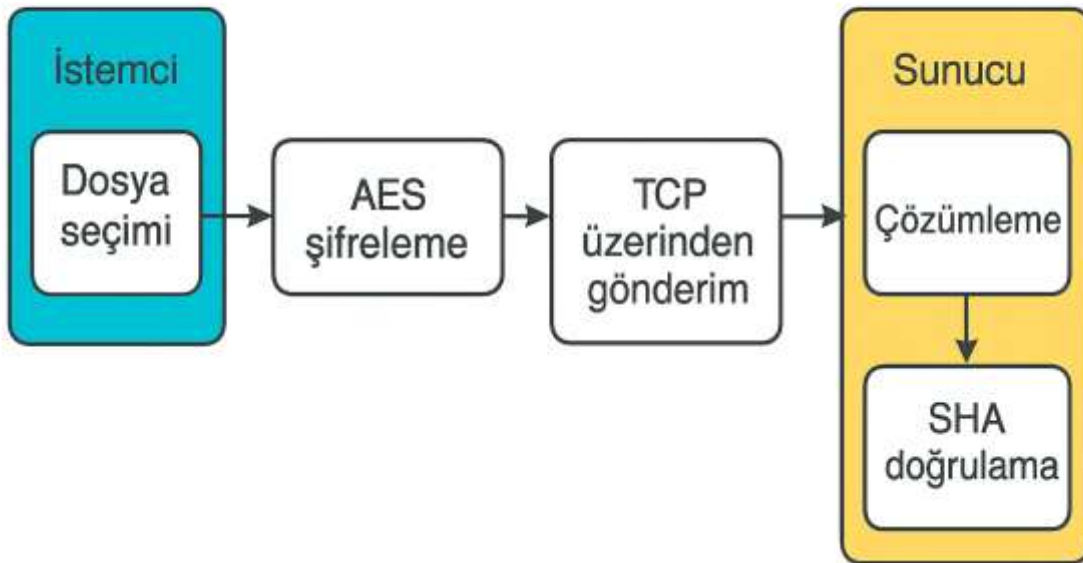
1. GİRİŞ

Günümüzde verilerin güvenli bir şekilde paylaşılması artık sadece büyük kuruluşların değil, bireylerin de ihtiyaç duyduğu bir konu hâline geldi. Özellikle dijital dosyaların ağ üzerinden gönderimi sırasında güvenliği sağlamak, veri bütünlüğünü korumak ve olası saldırılara karşı tedbir almak büyük önem taşıyor. Bu projede, bu ihtiyaca yönelik sade ama etkili bir çözüm geliştirildi. Projede temel hedef, bir dosyanın şifrelenerek parçalara ayrılması, ağ üzerinden güvenli bir biçimde gönderilmesi ve alıcı tarafından başarıyla birleştirilip çözülmesini sağlamaktır. Bunun için AES-256-CBC şifreleme algoritması kullanıldı. Ayrıca, şifreleme anahtarı güvenli bir şekilde türetilibilsin diye PBKDF2 yöntemine de yer verildi. Dosya transferi sırasında veriler parçalara ayrıldı ve her bir parça için kontrol (ACK) mekanizması geliştirildi. Bu sayede hem eksiksiz hem de güvenli bir aktarım hedeflendi. Kullanıcıların bu süreci daha rahat deneyimleyebilmesi için Flask tabanlı basit ama işlevsel bir web arayüzü de geliştirildi. Dosya seçimi, gönderim durumu ve SHA-256 hash kontrolü gibi önemli bilgiler bu arayüz üzerinden görüntülenebilir hâle getirildi.

2. Teknik Detaylar

2.1 Sistem Mimarisi

Bu projede, istemci-sunucu temelli bir mimari benimsenmiştir. Sistemin temel amacı, kullanıcıdan alınan dosyaların şifrelenmiş biçimde güvenli olarak karşı tarafa iletilmesini sağlamak ve alıcı tarafında bu dosyaların çözülerek doğruluğunun kontrol edilmesidir. Tüm veri aktarımı, TCP protokolü üzerinden gerçekleştirilmiştir. Sistem iki ana bileşenden oluşmaktadır: istemci tarafı ve sunucu tarafı. İstemci, kullanıcı arayüzü (GUI) üzerinden dosya seçilmesini sağlar ve ardından bu dosyayı AES-CBC algoritması kullanarak şifreler. Şifrelenen veri TCP soketi aracılığıyla sunucuya gönderilir. Gönderim öncesinde IV (Initialization Vector) sabit veya dinamik olarak belirlenebilir ve şifreleme anahtarı PBKDF2 ile üretilir. Verinin bütünlüğü SHA-256 algoritması ile sağlanır. Sunucu tarafında ise alınan şifreli veri, uygun anahtar ve IV kullanılarak çözülür. Elde edilen düz veri diske kaydedilir ve SHA-256 özeti hesaplanarak, istemciden gelen özet ile karşılaştırılır. Böylece dosyanın doğruluğu teyit edilir.



Şekil 1: İstemci-Sunucu Tabanlı Dosya Aktarım Mimarisi

Sistemin genel veri akışı ve işleyiş mantığı, Şekil 1’de gösterilen mimari şema ile özetlenmiştir. Bu şema, istemcinin kullanıcı etkileşimi, şifreleme işlemi, veri iletimi ve sunucunun çözümleme sürecini katmanlı bir yapı ile açıklamaktadır. Bu yapı sayesinde hem veri gizliliği

hem de bütünlüğü korunmakta, aynı zamanda kullanıcı dostu bir arayüz ile süreç şeffaf biçimde izlenebilmektedir.

2.2 Uygulama Yapısı

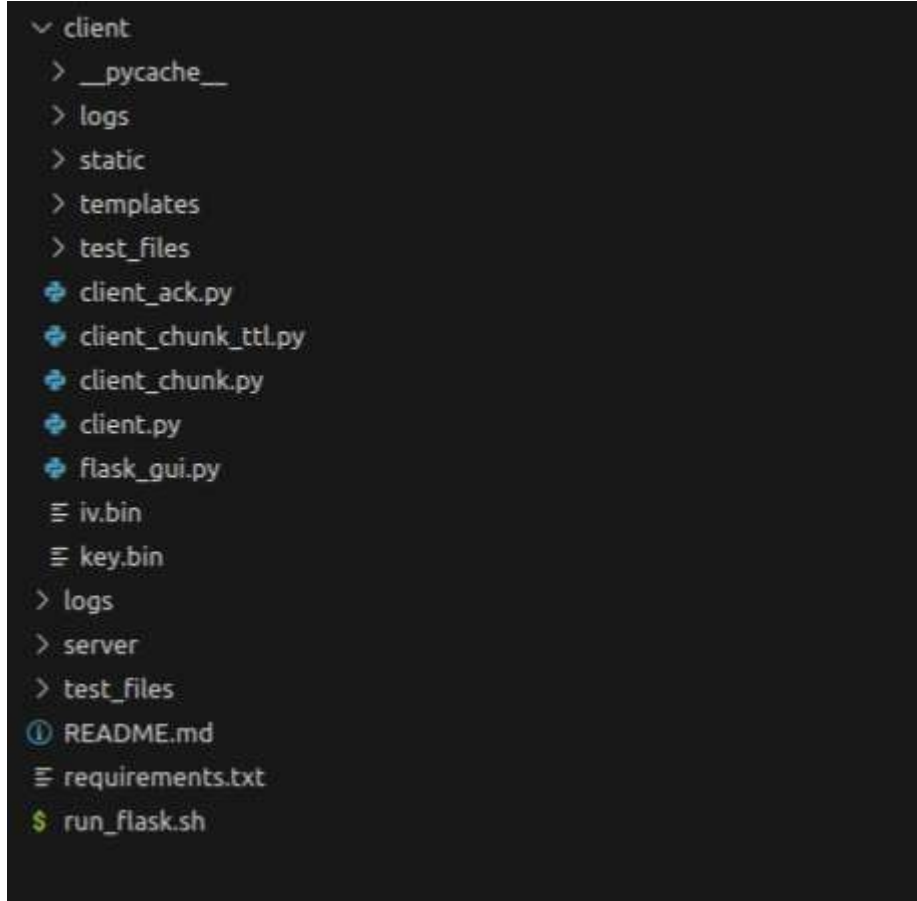
Geliştirilen sistem, işlevselliği yüksek ve sürdürülebilir bir yapı gözetilerek tasarlanmıştır. Proje dosyaları, görev ve sorumluluklarına göre ayrı klasörler altında gruplandırılmış; istemci, sunucu, loglama mekanizması ve test dosyaları net bir şekilde birbirinden ayrılmıştır. Bu yapı, hem geliştirme sürecinin daha kolay yönetilmesini hem de sistemin okunabilirliğinin ve test edilebilirliğinin artırılmasını sağlamıştır. Kodun modüler biçimde organize edilmesi, ileride yapılacak güncellemeler ve iyileştirmeler için de esneklik sunmaktadır.

Client Klasörü

Bu klasör, istemci tarafındaki tüm işlemleri ve iş mantığını içermektedir. Dosya şifreleme, parçalara ayırma, TCP soketleri aracılığıyla sunucuya veri gönderme ve yeniden iletim gibi işlemler burada gerçekleştirilmiştir. Ayrıca Flask tabanlı kullanıcı arayüzü de yine bu klasörün içerisinde yer almaktadır. client klasöründe yer alan önemli dosyalar şunlardır:

- client.py: Dosyayı şifreleyip tek seferde gönderir. Temel istemci mantığını uygular.
- client_chunk.py: Dosyayı belirli boyutlarda parçalara ayırarak gönderim sağlar.
- client_ack.py: Her bir parçanın başarılı şekilde ulaşp ulaşmadığını kontrol eder, gerekli durumlarda yeniden gönderim yapar.
- client_chunk_ttl.py: IP başlığındaki TTL değerini ayarlayarak veri iletiminin ağ davranışını gözlemlemeye yönelik özel testler yapılmasına imkân tanır.
- flask_gui.py: Web arayüzünü sağlayan dosyadır. Kullanıcının dosya seçmesini, SHA-256 hesaplamasını ve gönderim işlemlerini yönetir.

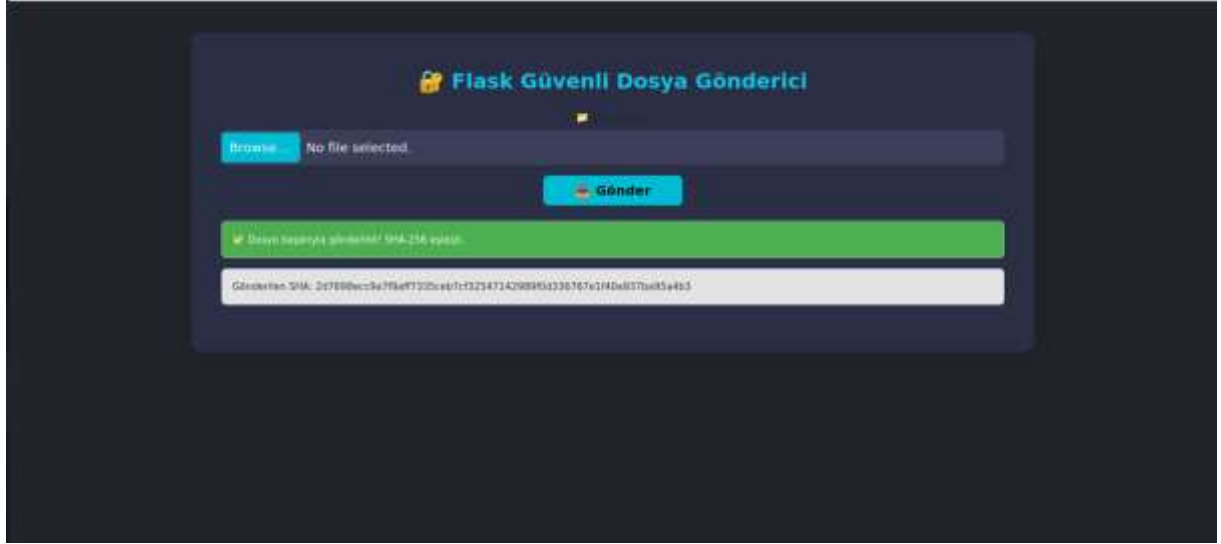
Bu yapının bütünlüğü, istemci tarafındaki işlemlerin her birinin ayrı modüller aracılığıyla ele alınmasına ve daha sonra gerektiğinde kolaylıkla genişletilebilmesine olanak tanımaktadır.



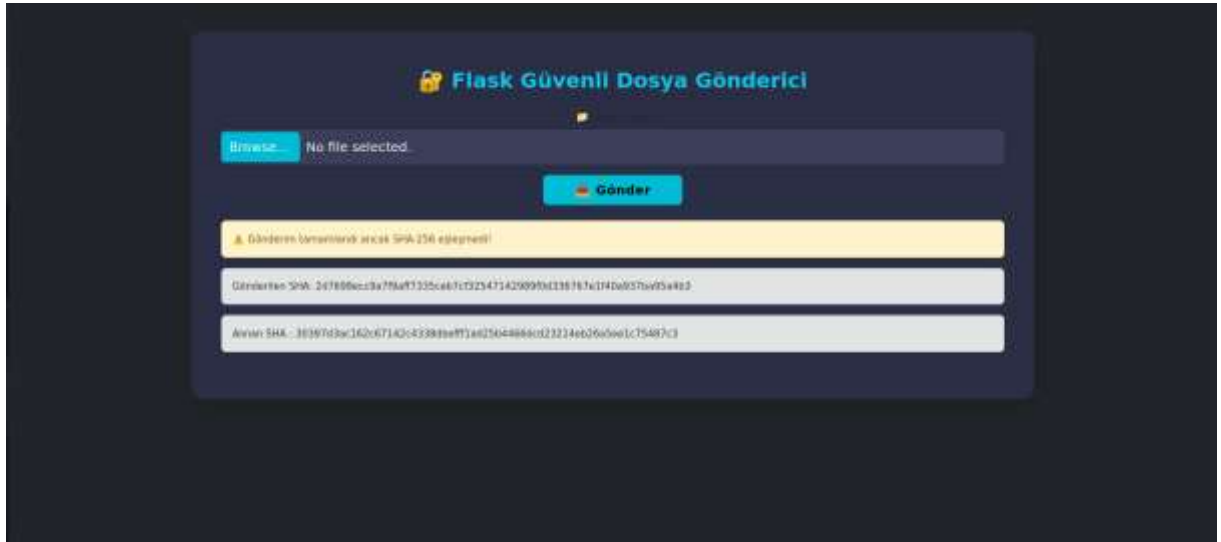
Şekil 2: Client klasör yapısı

Web Arayüzü (Flask)

Dosya şifreleme, Kullanıcı deneyimini artırmak amacıyla projeye Flask tabanlı bir web arayüzü entegre edilmiştir. Bu arayüz, teknik bilgisi olmayan kullanıcıların bile dosya gönderme işlemini rahatlıkla gerçekleştirmesini sağlar. Arayüz üzerinden kullanıcı bir dosya seçmekte, sistem bu dosyayı şifreleyip sunucuya göndermekte ve ardından SHA-256 özeti üzerinden gönderim başarı durumu kontrol edilmektedir. Sistem, gönderim işleminin başarılı olması hâlinde yeşil renkli bir uyarı mesajı, SHA eşleşmesi başarısız olursa sarı renkli bir uyarı kutusu ile kullanıcıyı bilgilendirir. Görsel sadelik ve okunabilirlik göz önünde bulundurularak Bootstrap destekli bir tasarım tercih edilmiştir.



Şekil 3: Arayüzde SHA-256'nın eşleşme durumu



Şekil 4: Arayüzde SHA-256'nın eşleşmeme durumu

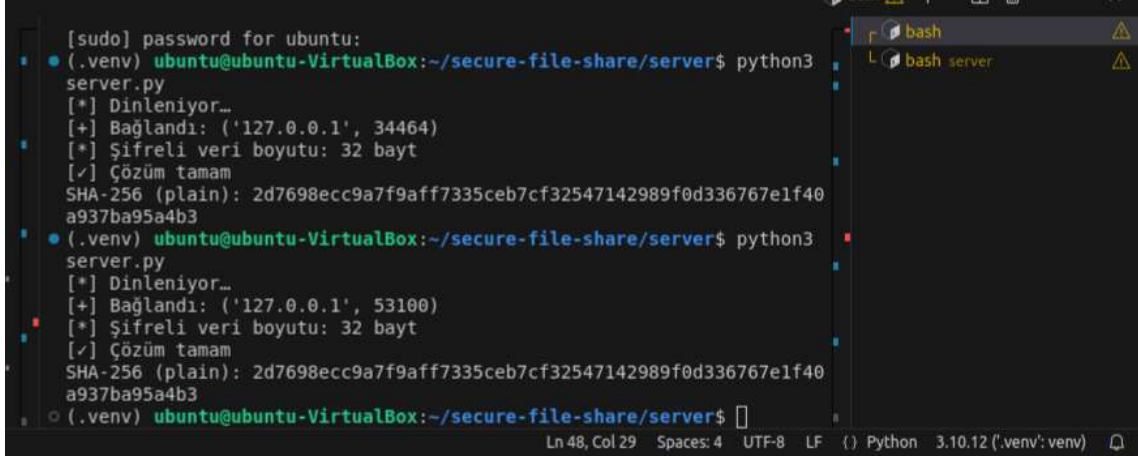
Server Klasörü

Server klasörü, sunucu tarafındaki veri alımı, çözümleme ve SHA doğrulama işlemlerini içermektedir. Gönderilen veriler TCP üzerinden alınarak, sırasına göre birleştirilmekte ve AES-CBC algoritması ile çözülmemektedir. Doğrulan dosya sistem üzerinde received.txt adıyla kaydedilir. Sunucu tarafında kullanılan temel dosyalar şunlardır:

- server.py: Şifrelenmiş verinin doğrudan alınıp çözüldüğü basit sunucu uygulamasıdır.
- server_chunk.py: Parça parça gelen verileri sırayla birleştirerek tek bir dosya haline

getirir.

- server_ack.py: Her parçaya karşılık istemciye ACK göndererek güvenli veri iletimine katkı sağlar.



```
[sudo] password for ubuntu:
(.venv) ubuntu@ubuntu-VirtualBox:~/secure-file-share/server$ python3
server.py
[*] Dinleniyor...
[+] Bağlandı: ('127.0.0.1', 34464)
[*] Şifreli veri boyutu: 32 bayt
[✓] Çözüm tamam
SHA-256 (plain): 2d7698ecc9a7f9aff7335ceb7cf32547142989f0d336767e1f40
a937ba95a4b3
(.venv) ubuntu@ubuntu-VirtualBox:~/secure-file-share/server$ python3
server.py
[*] Dinleniyor...
[+] Bağlandı: ('127.0.0.1', 53100)
[*] Şifreli veri boyutu: 32 bayt
[✓] Çözüm tamam
SHA-256 (plain): 2d7698ecc9a7f9aff7335ceb7cf32547142989f0d336767e1f40
a937ba95a4b3
(.venv) ubuntu@ubuntu-VirtualBox:~/secure-file-share/server$
```

Şekil 5: server.py terminal çıktısı

Logs ve test_files Klasörleri

logs klasörü, yapılan her gönderim işleminin detaylarını .csv formatında tutar. Her satırda gönderilen parçanın sıra numarası, veri boyutu ve iletim durumu ("OK" veya "FAIL") yer alır. Bu sayede sistemin performansı, yeniden iletim gerekliliği ve başarı oranları analiz edilebilir hâle gelmiştir.

test_files klasörü ise sistemde kullanılan örnek dosyaların yer aldığı ve sunucu tarafından üretilen çıktı dosyalarının saklandığı alandır. Kullanıcıdan alınan sample.txt veya file_1m.bin gibi dosyalar bu klasörde tutulur; sunucu tarafında oluşturulan received.txt dosyası da yine burada yer alır.

2.3 Test Senaryoları ve Sonuçlar

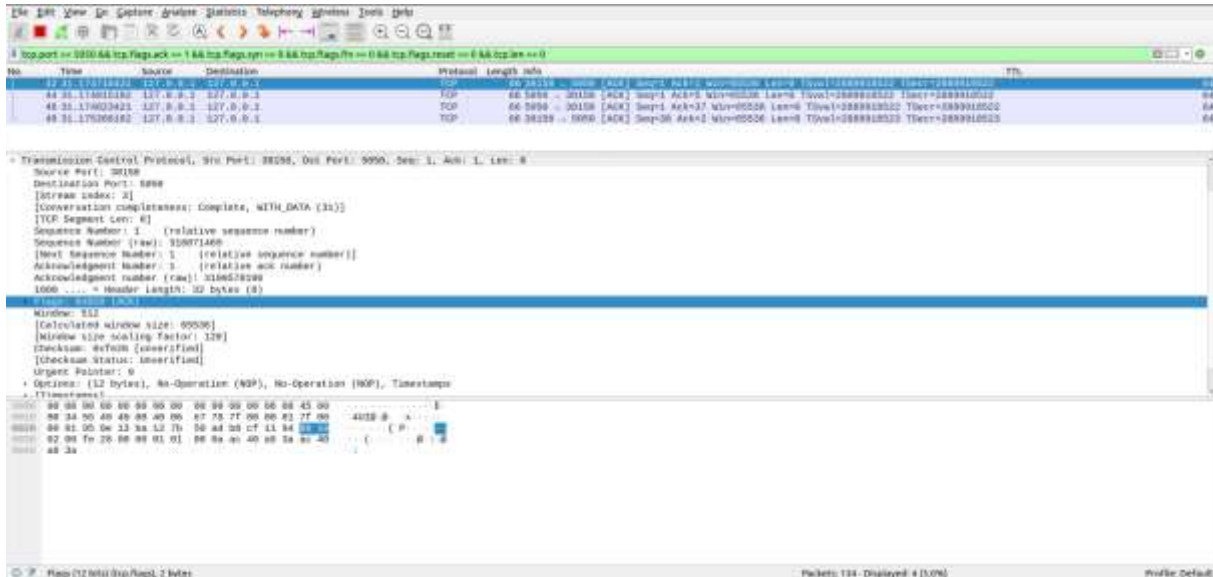
Geliştirdiğim sistemin çeşitli test senaryoları altında detaylıca değerlendirilip, hem dosya bütünlüğü hem de farklı koşullarda sergilediği davranış yakından gözlemlenmiştir. Aşağıda gerçekleştirilen testler, elde edilen sonuçlar ve ilgili ekran görüntüleri bulunmaktadır.

2.3.1 Dosya Bütünlüğü ve SHA-256 Doğrulama

Gönderilen dosyanın bütünlüğünü doğrulamak amacıyla istemci tarafında şifrelemeden önce

2.3.3 Wireshark ile Paket Seviyesi Analiz

Gönderilen Veri aktarımı sürecinde oluşturulan TCP paketleri Wireshark aracılığıyla detaylı analiz edilmiştir. Yapılan incelemede, IP başlıklarında TTL değerlerinin manuel ayarlandığı, TCP ACK paketlerinin doğru şekilde alındığı ve paketlerin düzgün bir sırada iletildiği doğrulanmıştır. Ayrıca, şifrelenmiş verilerin okunmaz durumda olduğu da teyit edilmiştir.



Şekil 7: saf ACK paketi

GŞekil 7’de tcp.flags == 0x10 filtresiyle yakalanan veri içermeyen saf ACK paketi görülmekte ; bu da istemci sunucu arasında paketlerin doğru sırayla alındığını teyit etmektedir.

2.3.4 Performans Değerlendirmeleri

Gerçekleştirilen performans değerlendirmeleri kapsamında sistemin dosya transferi süreci, farklı dosya boyutları ve çeşitli ağ koşulları altında kapsamlı bir şekilde test edilmiştir. Bu süreçte hem ideal (düşük gecikmeli ve kayıpsız) ağ ortamları hem de gecikme ve kayıp gibi olumsuz koşulları simüle eden senaryolar dikkate alınmıştır. Değerlendirme metriklerinden biri olarak sistemin oluşturduğu log kayıtları incelenmiş ve özellikle veri parçalarının durumu analiz edilmiştir. Her veri parçası için ayrı ayrı tutulan log kayıtlarında, parçaların boyut bilgisi ve alıcıya başarıyla ulaşıp ulaşmadığını belirten durum bilgisi (“OK” veya hata durumu) yer almaktadır. Aşağıda Şekil 8’de sunulan örnek log görüntüsünde de görüldüğü üzere, test edilen senaryoda tüm veri parçaları OK durumu ile tamamlanmış ve bu durum, sistemin istikrarlı bir

şekilde çalıştığını ve veri bütünlüğünü başarılı bir şekilde koruduğunu göstermiştir.

Bu analizler sonucunda, sistemin sadece ideal koşullarda değil, aynı zamanda zorlu ağ ortamlarında da yüksek güvenilirlik ve performans sergilediği tespit edilmiştir. Böylece, sistemin genel veri iletim başarımı açısından tutarlı ve güvenilir bir yapıya sahip olduğu deneysel olarak doğrulanmıştır.

```

1 0.000000, OK
2 0.000000, OK
3 0.000000, OK
4 0.000000, OK
5 0.000000, OK
6 0.000000, OK
7 0.000000, OK
8 0.000000, OK
9 0.000000, OK
10 0.000000, OK
11 0.000000, OK
12 0.000000, OK
13 0.000000, OK
14 0.000000, OK
15 0.000000, OK
16 0.000000, OK
17 0.000000, OK
18 0.000000, OK
19 0.000000, OK
20 0.000000, OK
21 0.000000, OK
22 0.000000, OK
23 0.000000, OK
24 0.000000, OK
25 0.000000, OK
26 0.000000, OK
27 0.000000, OK
28 0.000000, OK
29 0.000000, OK
30 0.000000, OK
31 0.000000, OK
32 0.000000, OK
33 0.000000, OK
34 0.000000, OK
35 0.000000, OK
36 0.000000, OK
37 0.000000, OK
38 0.000000, OK
39 0.000000, OK
40 0.000000, OK
41 0.000000, OK
42 0.000000, OK
43 0.000000, OK
44 0.000000, OK
45 0.000000, OK
46 0.000000, OK
47 0.000000, OK
48 0.000000, OK
49 0.000000, OK
50 0.000000, OK
51 0.000000, OK
52 0.000000, OK
53 0.000000, OK
54 0.000000, OK
55 0.000000, OK
56 0.000000, OK
57 0.000000, OK
58 0.000000, OK
59 0.000000, OK
60 0.000000, OK
61 0.000000, OK
62 0.000000, OK
63 0.000000, OK
64 0.000000, OK
65 0.000000, OK
66 0.000000, OK
67 0.000000, OK
68 0.000000, OK
69 0.000000, OK
70 0.000000, OK
71 0.000000, OK
72 0.000000, OK
73 0.000000, OK
74 0.000000, OK
75 0.000000, OK
76 0.000000, OK
77 0.000000, OK
78 0.000000, OK
79 0.000000, OK
80 0.000000, OK
81 0.000000, OK
82 0.000000, OK
83 0.000000, OK
84 0.000000, OK
85 0.000000, OK
86 0.000000, OK
87 0.000000, OK
88 0.000000, OK
89 0.000000, OK
90 0.000000, OK
91 0.000000, OK
92 0.000000, OK
93 0.000000, OK
94 0.000000, OK
95 0.000000, OK
96 0.000000, OK
97 0.000000, OK
98 0.000000, OK
99 0.000000, OK
100 0.000000, OK

```

Şekil 8: Tüm paketlerin başarılı şekilde gönderildiği .csv dosya içeriği

2.3.5 Kullanıcı Arayüzü Testleri

Kullanıcı arayüzünün, aktarım süreci boyunca kullanıcılara anlık ve anlaşılır geri bildirimler sağladığı test edilmiştir. Dosya seçimi, aktarım süreci ve SHA kontrol adımlarının kullanıcı dostu ve net şekilde tasarlandığı, hata durumlarında da net uyarı mesajlarının kullanıcıya başarılı şekilde sunulduğu görülmüştür. Başarılı durum Şekil 3'te de görülmektedir.

2.4 Güvenlik İncelemesi

Sistemin kullanıcı arayüzü, aktarım süreci boyunca kullanıcılara kolay anlaşılır ve anlık geri bildirimler sunacak şekilde tasarlanmıştır. Yapılan testlerde, arayüzün her adımda kullanıcıyı yönlendirdiği ve bilgilendirdiği görülmüştür. Özellikle dosya seçimi, aktarımın başlatılması, ilerleme çubuğunun gösterilmesi ve SHA kontrolü gibi adımların sade ve kullanıcı dostu şekilde sunulduğu dikkat çekmiştir. Aktarım sırasında bir sorun yaşandığında sistem, kullanıcıya neyin yanlış gittiğini açıkça belirten uyarılar vermektedir. Böylece kullanıcı ne yapması gerektiğini kolayca anlayabilmekte ve süreci güvenle yönetebilmektedir. Aynı şekilde, aktarım başarıyla tamamlandığında da kullanıcıya net ve tatmin edici bir bilgilendirme sağlanmaktadır. Bu durum Şekil 3'te de görülebileceği gibi, sistemin hem teknik olarak doğru çalıştığını hem de kullanıcı açısından anlaşılır ve güven verici bir deneyim sunduğunu ortaya koymaktadır. Genel olarak değerlendirildiğinde, arayüz hem basitliği hem de işlevselliği ile kullanıcıyı yormadan süreci takip etmesini mümkün kılmaktadır.

2.4.1 AES-256 Şifreleme (CBC Modu)

Veri aktarımı sırasında güvenliği en üst seviyede tutmak için AES şifreleme algoritmasının CBC (Cipher Block Chaining) modu tercih edilmiştir. Bu yöntemde, dosyalar istemci tarafında 256 bit uzunluğunda güçlü bir anahtarla şifrelenir. CBC modunun en dikkat çekici özelliği, her bir veri bloğunun bir önceki bloğa bağlı şekilde şifrelenmesidir. Yani aynı içerik defalarca bile şifrelenmiş olsa, her seferinde farklı bir çıktı üretir. Bu da veri tekrarlarının fark edilmesini engelleyerek gizliliği önemli ölçüde artırır. Şifreleme işlemine geçilmeden önce, dosyanın boyutu AES algoritmasının ihtiyaç duyduğu blok boyutuna uygun hale getirilir. Bunun için yaygın ve güvenli bir dolgu yöntemi olan PKCS#7 padding kullanılmıştır. Bu adım, dosya boyutu tam olarak AES'in beklediği uzunlukta olmasa bile şifrelemenin sorunsuz bir şekilde gerçekleşmesini sağlar. Tüm bu adımlar bir araya geldiğinde, sistemin sadece işlevsel değil, aynı zamanda güvenlik açısından da sağlam bir yapı sunduğu görülmektedir. Kullanıcı

verilerinin bütünlüğünü ve gizliliğini korumak için bu şifreleme yaklaşımı oldukça etkili bir çözüm sunmaktadır.

2.4.2 Anahtar ve IV (Initialization Vector) Kullanımı

Güvenliği daha ileri seviyeye taşımak için projede statik bir anahtar yerine dinamik olarak üretilen anahtarlar tercih edilmiştir. Bu anahtarlar, PBKDF2 algoritması ile kullanıcı tarafından sağlanan parola temel alınarak oluşturulur. Bu sayede, paroladan her kullanımda farklı ve güvenli bir anahtar türetimi mümkün kılınmıştır. Ayrıca, şifrelemenin güvenilirliğini korumak için her oturumda rastgele oluşturulan 16 byte uzunluğundaki IV (Initialization Vector) kullanılır. Bu IV, her yeni veri aktarımının başında sunucuya iletilerek güvenli veri transferi sağlanır.

2.4.3 SHA-256 ile Veri Bütünlüğü Doğrulaması

Dosya aktarımı tamamlandıktan sonra, verinin aktarım sırasında bozulup bozulmadığını kontrol etmek için hem istemci (gönderici) hem de sunucu (alıcı) tarafında SHA-256 özet değeri hesaplanmaktadır. Bu özet, dosyanın içeriğini temsil eden benzersiz bir dijital parmak izi gibidir. Aktarımdan önce ve sonra hesaplanan bu iki değer karşılaştırıldığında, birebir eşleşmeleri verinin bütünlüğünü koruduğunu, yani herhangi bir kayıp ya da değişiklik yaşanmadığını gösterir. Eğer bu iki özet değeri birbirine eşitse, sistem kullanıcıya başarılı bir aktarım gerçekleştiğine dair net bir geri bildirim sunar. Bu durum, Şekil 3'te görülen arayüz üzerinden kullanıcıya görsel olarak da aktarılır. Ancak eğer değerler eşleşmezse, yani dosyada bir bozulma ya da beklenmeyen bir değişiklik oluşmuşsa, kullanıcıya durumu açıkça belirten bir uyarı mesajı gösterilir. Aynı zamanda bu olay, sistem tarafından sha_mismatch.log adlı kayıt dosyasına detaylı şekilde not edilir. Bu sayede hem kullanıcı bilgilendirilmiş olur hem de sistem yöneticileri ileride bu tür durumları analiz edebilir. Bu mekanizma, sistemin sadece dosyaları göndermekle kalmadığını, aynı zamanda gönderilen verinin doğruluğunu ve güvenilirliğini de sürekli olarak kontrol ettiğini göstermektedir.

2.4.4 Parça Tabanlı Kontrol ve ACK Mekanizması

Dosya aktarımı sırasında olası veri kayıplarını önlemek için sistemde bir ACK (acknowledgement – onay) mekanizması kullanılmıştır. Bu yapı, gönderilen her veri parçasının gerçekten alıcıya ulaşmış olup olmadığını kontrol etmek amacıyla geliştirilmiştir. client_ack.py ve server_ack.py modülleri bu süreci yönetmekte, istemci tarafı her gönderimden sonra sunucudan

açık bir “onay” beklemektedir. Eğer gönderilen veri parçası için onay mesajı zamanında alınamazsa, istemci ilgili parçayı yeniden göndermeyi dener. Bu işlem en fazla üç kez tekrarlanır. Üç denemeye rağmen onay alınamazsa, sistem aktarımı durdurur ve kullanıcının olası bir sorunla karşı karşıya olduğunu bildirir. Bu mekanizma, ağdaki olası kesintiler, gecikmeler veya kayıplar gibi durumlara karşı sistemin daha dayanıklı olmasını sağlar. Aynı zamanda, her bir parçanın eksiksiz ulaştığından emin olarak genel aktarım güvenliğini artırır. Sonuç olarak, bu yöntem sayesinde hem veri kaybı riski en aza indirilmiş olur hem de sistem, daha tutarlı ve sağlam bir veri iletişim yapısı sunar.

2.4.5 Statik Kod Analizi (Bandit)

Projenin kaynak kodları, güvenlik açıklarını ve olası riskleri tespit etmek amacıyla Bandit statik analiz aracıyla taranmıştır. Yapılan bu taramada, açık anahtar kullanımı, güvenli olmayan kütüphaneler ve şifrelenmiş verilere erişim gibi kritik noktalar kontrol edilmiştir. Yapılan analiz sonucunda herhangi kritik bir güvenlik açığı bulunmamıştır.

3. Sınırlılıklar ve İyileştirmeler

3.1. Sınırlılıklar

Geliştirme sürecinde karşılaşılan temel sınırlılıklar şunlardır:

- **Qt Tabanlı GUI Uygulaması:** İlk olarak Qt kullanılarak oluşturulması planlanan grafik kullanıcı arayüzü, VirtualBox üzerinde Ubuntu ortamında VS Code entegrasyonu ile ilgili uyumsuzluk ve görüntüleme problemleri nedeniyle gerçekleştirilememiştir. Bu sebeple daha stabil ve pratik olan Flask web tabanlı GUI tercih edilmiştir.
- **Çoklu Kullanıcı Desteği:** Sistem şu an sadece tek kullanıcı bağlantısını desteklemektedir. Eş zamanlı birden fazla kullanıcı desteği henüz sağlanamamıştır.

3.2. İyileştirme Önerileri

Sistemin gelecekteki sürümlerinde daha iyi hale getirilmesi için aşağıdaki iyileştirmeler önerilmektedir:

- **Mobil Uygulama Desteği:** Sistemin mobil cihazlar üzerinden kullanılabilmesi, kullanıcı erişimini ve sistemin pratikliğini artıracaktır.

- **Gerçek Zamanlı Bildirim ve İzleme:** Kullanıcıya sistem durumu hakkında anlık bildirim gönderen ve gerçek zamanlı izleme sağlayan mekanizmalar entegre edilebilir.
- **Otomatik Veri Yedekleme:** Veri kaybını minimize etmek için otomatik veri yedekleme sisteminin entegre edilmesi önerilir.
- **Yapay Zeka Tabanlı Güvenlik:** Sisteme kullanıcı aktivitelerini izleyerek olağandışı davranışları otomatik tespit eden yapay zeka destekli analiz araçları eklenebilir.
- **Bulut Hizmetleri ile Entegrasyon:** Sistemin AWS veya Azure gibi bulut platformlarına taşınması, ölçeklenebilirlik ve güvenilirliği artıracaktır.
- **Kapsamlı Analitik Araçları:** Sistem performansını ve güvenliğini daha etkin takip etmek için detaylı loglama ve analitik araçları entegre edilebilir.

Bu önerilerin uygulanmasıyla sistemin performansı ve kullanıcı deneyimi önemli ölçüde iyileştirilecektir.

4. SONUÇ

Bu proje kapsamında geliştirilen güvenli dosya aktarım sistemi, temel şifreleme prensiplerini gerçek zamanlı TCP bağlantısı üzerinden pratik bir biçimde uygulamayı başarmıştır. AES-CBC algoritmasıyla veriler şifrelenmiş, SHA-256 algoritması ile veri bütünlüğü kontrol edilmiş ve istemci-sunucu mimarisi başarılı şekilde kurulmuştur. Flask tabanlı kullanıcı arayüzü ile desteklenen sistem, kullanıcı dostu bir deneyim sunarken, güvenlik odaklı özelliklerle donatılmıştır. Sistemde loglama, geri bildirim ve hata yönetimi gibi temel konular da etkin bir şekilde uygulanmıştır. Özellikle ağ davranışlarını analiz etmek için Wireshark ile yapılan detaylı paket incelemeleri, sistemin doğru ve güvenli çalıştığını kanıtlamıştır.

Proje sürecinde karşılaşılan platform kaynaklı uyumluluk problemleri, Flask web tabanlı arayüz kullanılarak başarıyla aşıp ve platform bağımsızlığı sağlanmıştır. Bu deneyim sayesinde bilgisayar ağları, veri güvenliği ve sistem entegrasyonu konularında kapsamlı bilgi ve uygulamalı olarak pratik yapılmıştır. Bu proje, gerçek dünya problemlerine yönelik çözüm üretme becerisini geliştirdiği gibi, daha ileri seviyede özellikler için bir altyapı sunmaktadır. İlerleyen aşamalarda çoklu kullanıcı desteği, HTTPS protokolü üzerinden güvenli aktarım ve mobil uygulama entegrasyonu gibi özelliklerle sistemin daha da geliştirilmesi uygulamayı daha

gelişmiş hale getirebilir.

5. KAYNAKÇA

Dheeraj, M. (2023). How to encrypt and decrypt files in Python using AES – a step-by-step guide. Medium. <https://medium.com/@dheeraj.mickey/how-to-encrypt-and-decrypt-files-in-python-using-aes-a-step-by-step-guide-d0eb6f525e4e>

GeeksforGeeks. (n.d.). Stop and Wait ARQ. <https://www.geeksforgeeks.org/stop-and-wait-arq/>

Privia Security. (2021). Wireshark ile Ağ Trafiğini İncelemek. <https://www.priviasecurity.com/blog/wireshark-ile-ag-trafigini-incelemek/>

ÇözümPark. (n.d.). Wireshark Temel Seviye. <https://www.cozumpark.com/wireshark-temel-seviye/>

Geleceği Yazarlar. (n.d.). Wireshark ile Port Trafik Tarama Analizi. Turkcell. <https://gelecegiyazarlar.turkcell.com.tr/blog/wireshark-ile-port-trafik-tarama-analizi>

DataCamp. (n.d.). A Complete Guide to Socket Programming in Python. <https://www.datacamp.com/tutorial/a-complete-guide-to-socket-programming-in-python>

6. EKLER

github linki : <https://github.com/Adileakklc/secure-file-share>

youtube linki : <https://youtu.be/3HALmuyQzB0>

linkedin linki : https://www.linkedin.com/posts/adileakklc_temel-g%C3%BCvenlikli-dosya-payla%C5%9F%C4%B1m-sistemi-activity-7338300268171661313-oJ-n?utm_source=share&utm_medium=member_desktop&rcm=ACoAADf7bvgBcxOLg0WJDjBHvVW_p5foXqDiAIs