Hazırlayan:
Fatmanur Zehra Bozkurt

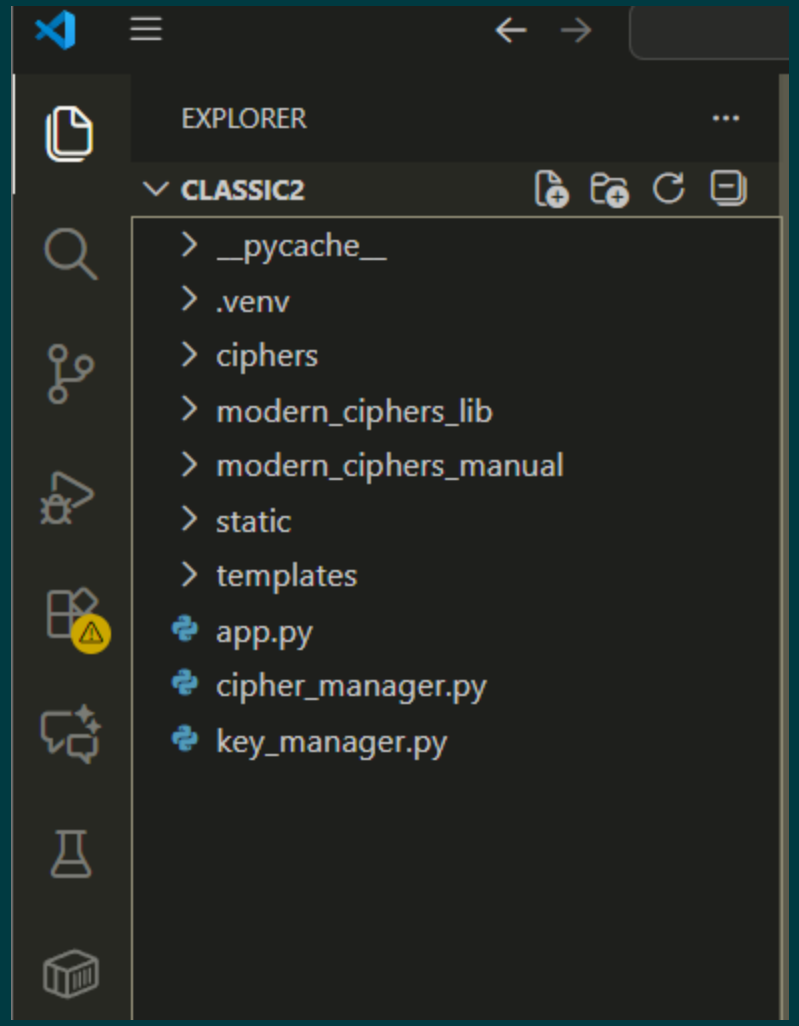
Ders / Proje Türü:
Bilgi Güvenliği / Kriptoloji Dersi Projesi

Github Bağlantısı:

fatmanurzb/classic2

KRİPTOGRAFİ TABANLI GÜVENLİ CHAT VE ÖĞRENMEYE BAŞLANGIÇ UYGULAMASI

-KLASIK VE MODERN ŞIFRELEME ALGORITMALARININ UYGULAMALI ANALIZI



CLASSIC2

- __pycache__
- .venv
- ciphers
  - __pycache__
  - __init__.py
  - affine.py
  - base.py
  - caesar.py
  - columnar.py
  - hill.py
  - pigpen.py
  - playfair.py
  - polybius.py
  - railfence.py
  - rot.py
  - substitution.py
  - vigenere.py
- modern_ciphers_lib
  - __pycache__
  - aes_lib.py
  - des_lib.py
  - rsa_lib.py
- modern_ciphers_manual
  - __pycache__
  - aes_manual.py
  - des_manual.py
- screenshots

CLASSIC2

- static
  - chat_server.js U
  - chat.js
  - main.js
  - style.css
- templates
  - aes.html
  - affine.html
  - caesar.html
  - chat_client.html
  - chat_server.html M
  - columnar.html
  - des.html
  - hill.html
  - index_client.html
  - index_server.html M
  - pigpen.html
  - playfair.html
  - polybius.html
  - railfence.html
  - rot.html
  - rsa.html
  - substitution.html
  - vigenere.html
- app.py M
- cipher_manager.py
- key_manager.py

1.GİRİŞ

Günümüzde bilgi ve iletişim teknolojilerinin hızlı gelişimi, dijital ortamda iletilen verilerin güvenliğini kritik bir konu haline getirmiştir. Özellikle mesajlaşma uygulamaları, kullanıcılar arasında sürekli veri alışverişi gerçekleştirdiğinden, bu verilerin yetkisiz kişiler tarafından ele geçirilmesi ciddi güvenlik riskleri oluşturmaktadır.

Kriptografi, verilerin gizliliğini, bütünlüğünü ve güvenliğini sağlamak amacıyla kullanılan temel bir güvenlik alanıdır. Tarihsel süreçte geliştirilen klasik şifreleme algoritmaları, günümüz güvenlik ihtiyaçları açısından yetersiz kalırken; modern kriptografik algoritmalar, daha güçlü matematiksel temellere dayanarak güvenli iletişim imkânı sunmaktadır.

2. PROJENİN AMACI VE KAPSAMI

Bu projenin temel amacı, kriptoloji alanında kullanılan klasik ve modern şifreleme algoritmalarını uygulamalı olarak göstermek ve bu algoritmaların güvenlik düzeylerini karşılaştırmalı biçimde incelemektir. Teorik olarak öğrenilen şifreleme yöntemlerinin, gerçek bir uygulama üzerinde nasıl çalıştığını gözlemlemek, kriptografinin pratikteki önemini ortaya koymaktadır.
Proje kapsamında geliştirilen web tabanlı chat sistemi, kullanıcıların mesajlarını farklı şifreleme algoritmaları ile göndermesine olanak sağlamaktadır. Sistem, klasik şifreleme algoritmaları (Caesar, Vigenere, Affine, Hill, Playfair, Columnar Transposition, Rail Fence, Polybius, Pigpen, ROT ve Substitution) ile modern şifreleme algoritmalarını (AES, DES ve RSA) destekleyecek şekilde tasarlanmıştır.
Klasik şifreleme algoritmaları, eğitim amaçlı olarak mesajların şifrelenmesi ve çözülmesi sürecinde kullanılmıştır. Modern şifreleme algoritmaları ise güvenli iletişim senaryosu kapsamında ele alınmış; simetrik şifreleme için AES ve DES, anahtar güvenliği için ise RSA algoritması kullanılmıştır.
Projenin bir diğer önemli amacı, şifrelenmiş verilerin ağ üzerindeki davranışını incelemektir. Bu doğrultuda, chat uygulaması üzerinden gönderilen mesajların ağ trafiği Wireshark aracı kullanılarak analiz edilmiştir. Böylece, şifreli ve şifresiz verilerin ağ üzerinde nasıl göründüğü gözlemlenmiş ve güvenli iletişimin önemi somut olarak ortaya konmuştur.
Sonuç olarak bu proje, kriptografi algoritmalarının çalışma prensiplerini öğretmeyi, güvenli iletişim kavramını pekiştirmeyi ve ağ trafiği analizi ile güvenlik farkındalığını artırmayı hedeflemektedir.

3. KULLANILAN TEKNOLOJİLER VE ARAÇLAR

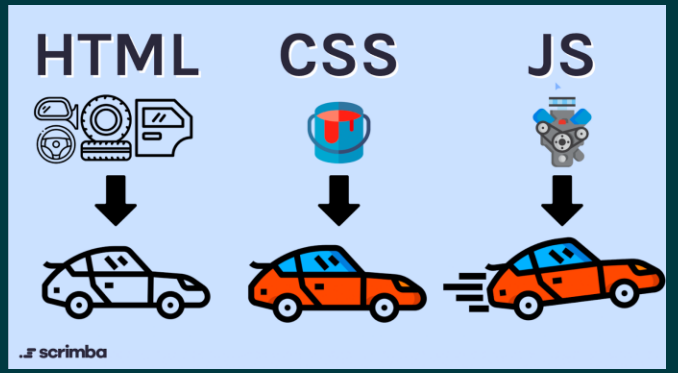
Bu projede, web tabanlı bir istemci–sunucu mimarisi kullanılarak güvenli ve klasik şifreleme algoritmalarını içeren bir haberleşme sistemi geliştirilmiştir.

Python programlama dili, projenin sunucu tarafında temel geliştirme dili olarak kullanılmıştır. Python'un sade sözdizimi ve güçlü kütüphane desteği, kriptografi uygulamalarının gerçekleştirilmesini kolaylaştırmıştır.

Flask web framework'ü, istemci ve sunucu arasındaki HTTP tabanlı sayfaların yönetilmesi için tercih edilmiştir. Flask'ın hafif ve esnek yapısı sayesinde proje modüler bir şekilde geliştirilmiştir.

Flask-SocketIO, gerçek zamanlı istemci–sunucu iletişimini sağlamak amacıyla kullanılmıştır. Bu yapı sayesinde chat sistemi, mesajları anlık olarak iletebilmekte ve kullanıcılar arasında gerçek zamanlı veri aktarımı gerçekleştirilmektedir.

HTML, CSS ve JavaScript, istemci arayüzünün oluşturulmasında kullanılmıştır. HTML sayfa yapısını oluştururken, CSS görsel tasarım ve düzenleme işlemlerini sağlamış, JavaScript ise kullanıcı etkileşimleri ve Socket.IO haberleşmesini yönetmiştir.



PyCryptodome kütüphanesi, modern kriptografi algoritmalarının (AES, DES ve RSA) güvenli ve standartlara uygun şekilde uygulanabilmesi için kullanılmıştır. Bu kütüphane sayesinde gerçek dünyada kullanılan kriptografik işlemler birebir simüle edilmiştir.

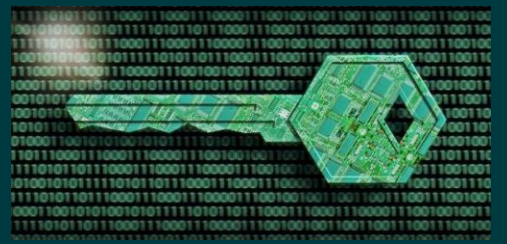
Manuel (kütüphanesiz) şifreleme implementasyonları, AES ve DES algoritmalarının sadeleştirilmiş versiyonları kullanılarak geliştirilmiştir. Bu sayede algoritmaların round yapıları, anahtar kullanımı ve dönüşüm adımları uygulamalı olarak deneyimlenmiştir.

Wireshark, ağ trafiğinin analiz edilmesi amacıyla kullanılmıştır. Chat uygulaması üzerinden gönderilen şifreli ve şifresiz mesajlar Wireshark ile yakalanmış, paket içerikleri incelenmiş ve şifreleme algoritmalarının ağ trafiğine etkileri gözlemlenmiştir.

Visual Studio Code, proje geliştirme sürecinde kod editörü olarak tercih edilmiştir. **GitHub** ise proje dosyalarının versiyon kontrolü ve paylaşımı amacıyla kullanılmıştır.

```
PS E:\pachong> pip install pycryptodome
Collecting pycryptodome
  Downloading pycryptodome-3.15.0-cp35-abi3-win_amd64.whl (1.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 22.3 kB/s eta 0:00:00
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.15.0
```

CSDN @mlws1900

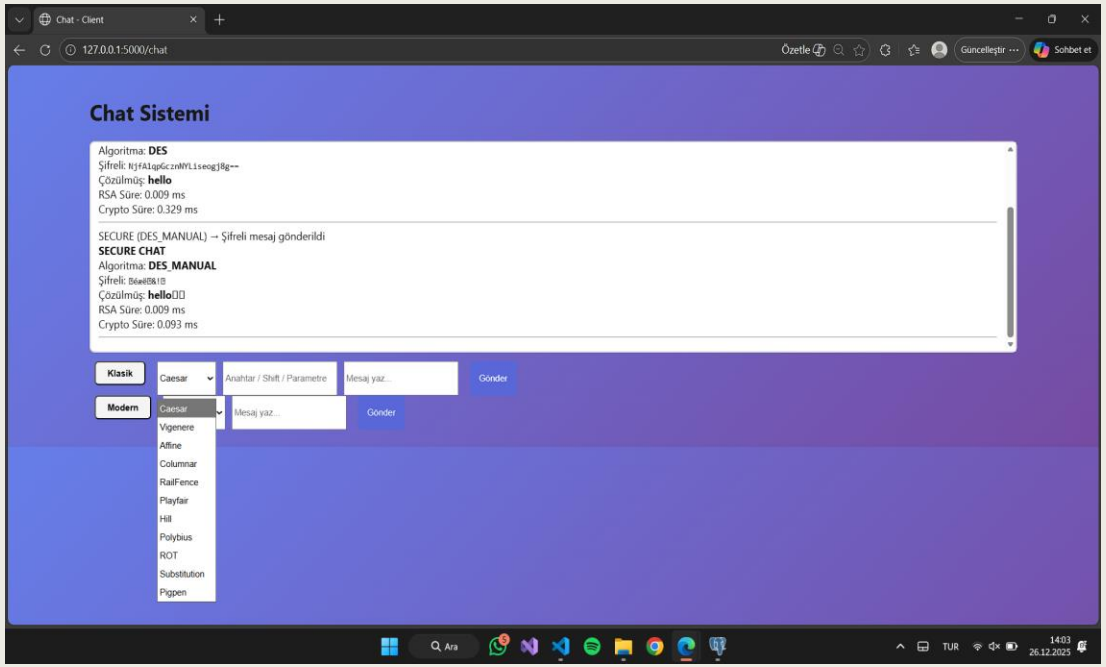


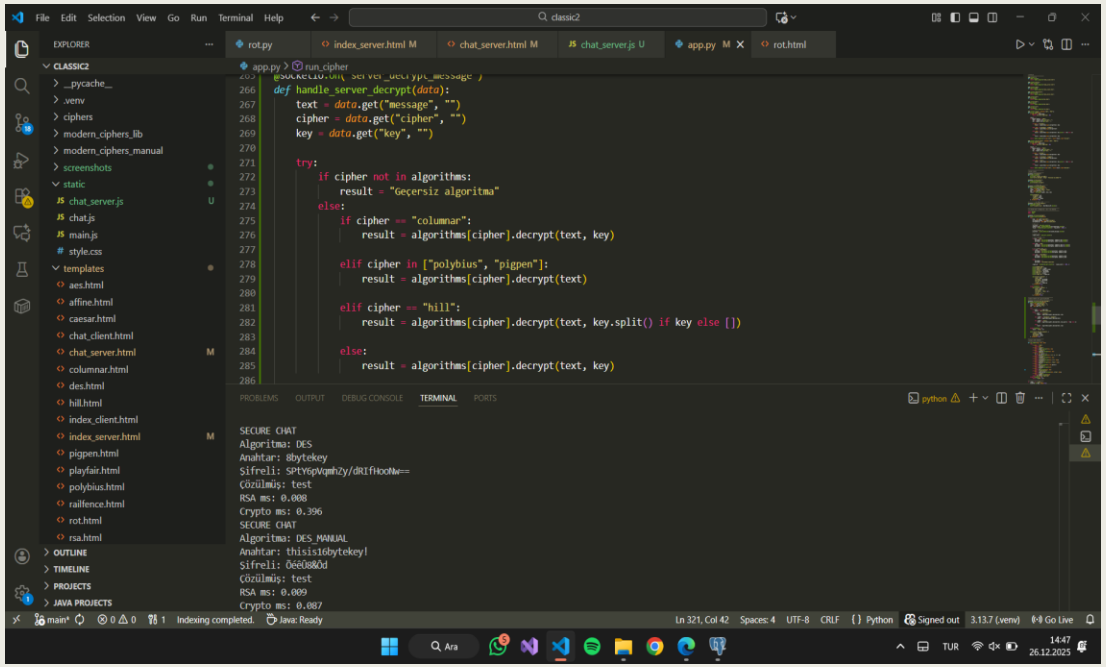
4. SİSTEM MİMARİSİ VE GENEL ÇALIŞMA YAPISI

Geliştirilen sistem, istemci–sunucu mimarisi temel alınarak tasarlanmıştır. Sistem; istemci arayüzü, sunucu uygulaması ve kriptografik işlem modüllerinden oluşmaktadır. Bu yapı sayesinde hem klasik hem de modern şifreleme algoritmaları kullanılarak güvenli veri iletişimi sağlanmıştır.

İstemci tarafı, web tarayıcısı üzerinden çalışan HTML, CSS ve JavaScript bileşenlerinden oluşmaktadır. Kullanıcı, istemci arayüzü üzerinden göndermek istediği mesajı ve kullanılacak şifreleme algoritmasını seçmektedir. Seçilen algoritmaya göre mesaj, klasik veya modern kriptografi modüllerine yönlendirilmektedir.

Sunucu tarafı, Flask ve Flask-SocketIO kullanılarak geliştirilmiştir. Sunucu, istemciden gelen mesajları Socket.IO aracılığıyla almakta ve ilgili şifreleme/çözme işlemlerini gerçekleştirmektedir. İşlenen mesajlar, tekrar istemciye gönderilerek sonuçların anlık olarak görüntülenmesi sağlanmaktadır.





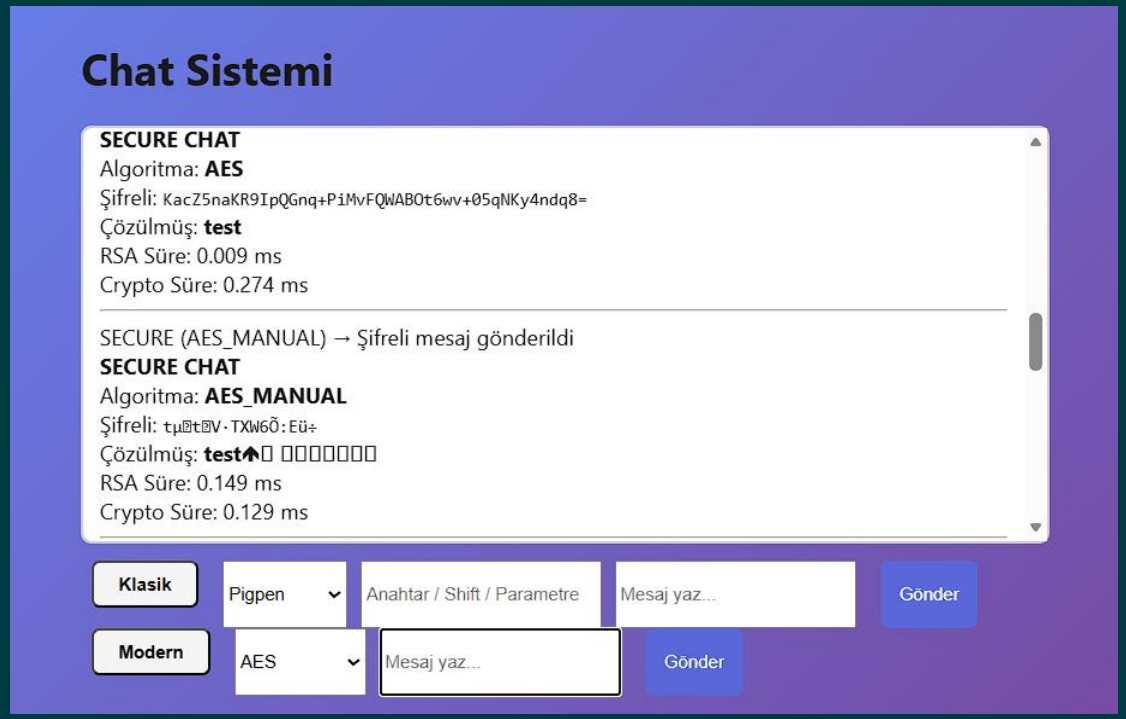
Sistem iki ana çalışma moduna sahiptir:

Klasik Şifreleme Modu:
Bu modda Caesar, Vigenere, Affine, Rail Fence, Playfair, Hill, Polybius, ROT, Substitution ve Pigpen gibi klasik kriptografi algoritmaları kullanılmaktadır. Mesajlar istemci tarafından sunucuya iletilir, sunucuda şifrelenir veya çözülür ve sonuç tüm istemcilere yayınlanır.

Güvenli (Secure) Şifreleme Modu:
Bu modda AES, DES ve RSA algoritmaları birlikte kullanılmaktadır. Simetrik şifreleme algoritmaları (AES, DES) mesajın gizliliğini sağlarken, RSA algoritması simetrik anahtarın güvenli şekilde paylaşılması amacıyla kullanılmaktadır. Böylece hibrit bir kriptografi yapısı oluşturulmuştur.
AES ve DES algoritmaları hem kütüphane tabanlı hem de manuel implementasyon olarak iki farklı şekilde uygulanmıştır. Kütüphane tabanlı yöntemler gerçek dünyada kullanılan standartları temsil ederken, manuel yöntemler algoritmaların temel mantığını öğretici bir yaklaşımla sunmaktadır.
Sistem mimarisinde kriptografik işlemler modüler bir yapıda geliştirilmiştir. Her algoritma ayrı bir dosya ve sınıf içerisinde tanımlanmış, böylece kodun okunabilirliği ve sürdürülebilirliği artırılmıştır.
İstemci–sunucu arasındaki tüm haberleşme Socket.IO üzerinden gerçekleştiği için mesajlar gerçek zamanlı olarak iletilmektedir.

5. AES, DES VE RSA ALGORİTMALARININ KULLANIMI VE KARŞILAŞTIRILMASI

5.1 AES (Advanced Encryption Standard)



modern_ciphers_lib > aes_lib.py > aes_decrypt

```python
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad
import base64

def aes_encrypt(plaintext: str, key: bytes) -> str:
    iv = get_random_bytes(16)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    ciphertext = cipher.encrypt(pad(plaintext.encode(), AES.block_size))
    return base64.b64encode(iv + ciphertext).decode()

def aes_decrypt(ciphertext_b64: str, key: bytes) -> str:
    data = base64.b64decode(ciphertext_b64)
    iv = data[:16]
    ciphertext = data[16:]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    return unpad(cipher.decrypt(ciphertext), AES.block_size).decode()

```

aes_lib.py

modern_ciphers_manual > aes_manual.py > inv_shift_rows

```python

# Basit S-Box (gerçek AES S-box DEĞİL)
S_BOX = {
    i: (i * 7 + 3) % 256 for i in range(256)
}

INV_S_BOX = {v: k for k, v in S_BOX.items()}


def sub_bytes(block):
    return [S_BOX[b] for b in block]


def inv_sub_bytes(block):
    return [INV_S_BOX[b] for b in block]


def shift_rows(block):
    # 4x4 matris varsayımı (AES mantığı)
    return [
        block[0],  block[1],  block[2],  block[3],
        block[5],  block[6],  block[7],  block[4],
        block[10], block[11], block[8],  block[9],
        block[15], block[12], block[13], block[14]
    ]


def inv_shift_rows(block):
    return [
        block[0],  block[1],  block[2],  block[3],
        block[7],  block[4],  block[5],  block[6],
        block[10], block[11], block[8],  block[9],
        block[13], block[14], block[15], block[12]
    ]


def add_round_key(block, key):
```

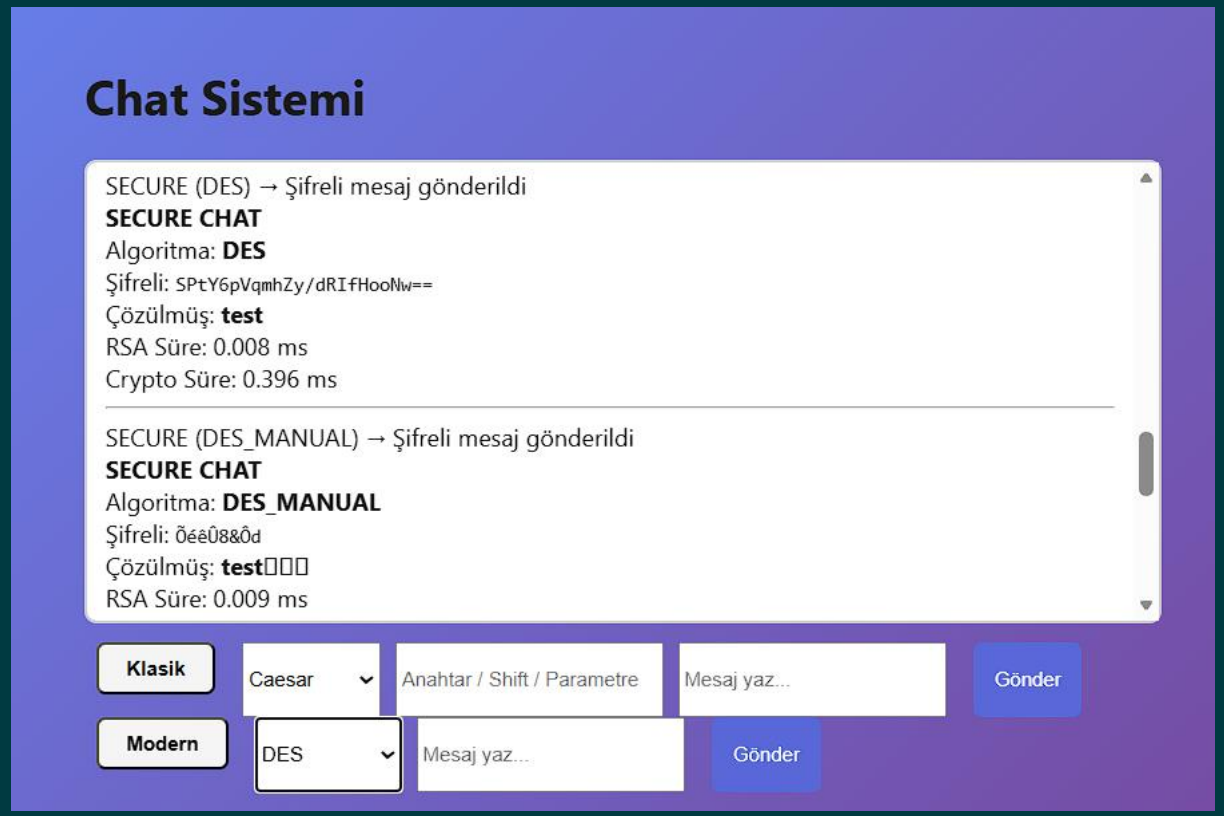
aes_manual.py

“Manuel AES uygulaması, algoritmanın eğitimsel açıdan anlaşılmasını sağlarken; kütüphane tabanlı AES uygulaması, gerçek dünya güvenlik gereksinimlerini karşılayan profesyonel bir çözümdür.”

•AES, günümüzde en yaygın kullanılan simetrik şifreleme algoritmalarından biridir. Bu çalışmada AES-128 kullanılmıştır ve anahtar uzunluğu 16 byte olarak belirlenmiştir.

- AES algoritması blok şifreleme mantığıyla çalışmakta olup, veriyi 128-bit bloklar halinde işlemektedir. Her blok üzerinde SubBytes, ShiftRows, MixColumns ve AddRoundKey adımlarından oluşan çok turlu (round) bir yapı uygulanmaktadır.
- Projede AES algoritması iki farklı şekilde uygulanmıştır:
- **Kütüphane Tabanlı AES:** PyCryptodome kütüphanesi kullanılarak gerçek dünya standartlarına uygun bir AES uygulaması gerçekleştirilmiştir.
- **Manuel AES:** Algoritmanın sadeleştirilmiş bir versiyonu manuel olarak kodlanmış, böylece AES'in temel çalışma mantığı deneyimlenmiştir.
- Yapılan testlerde kütüphane tabanlı AES'in manuel implementasyona göre daha güvenli ve stabil olduğu, ancak manuel implementasyonun eğitimsel açıdan daha öğretici olduğu gözlemlenmiştir.

5.3 DES (Data Encryption Standard)



modern_ciphers_lib > des_lib.py > ...

```python
from Crypto.Cipher import DES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes
import base64

def des_encrypt(plaintext: str, key: bytes) -> str:
    if len(key) != 8:
        raise ValueError("DES key must be exactly 8 bytes")

    iv = get_random_bytes(8)
    cipher = DES.new(key, DES.MODE_CBC, iv)
    ciphertext = cipher.encrypt(pad(plaintext.encode(), 8))

    return base64.b64encode(iv + ciphertext).decode()


def des_decrypt(ciphertext_b64: str, key: bytes) -> str:
    if len(key) != 8:
        raise ValueError("DES key must be exactly 8 bytes")

    raw = base64.b64decode(ciphertext_b64)
    iv = raw[:8]
    ciphertext = raw[8:]

    cipher = DES.new(key, DES.MODE_CBC, iv)
    plaintext = unpad(cipher.decrypt(ciphertext), 8)

    return plaintext.decode()
```

des_lib.py

modern_ciphers_manual > des_manual.py > generate_subkeys

```python
def xor(a, b):
    return [i ^ j for i, j in zip(a, b)]


def feistel_function(right, subkey):
    # Basit substitution + XOR
    return [(r + k) % 256 for r, k in zip(right, subkey)]


def generate_subkeys(key):
    key_bytes = [ord(c) for c in key.ljust(8)[:8]]
    return [
        key_bytes[i:] + key_bytes[:i]
        for i in range(4)
    ]


def pad(text):
    while len(text) % 8 != 0:
        text += chr(8 - (len(text) % 8))
    return text


def unpad(text):
    return text.rstrip(text[-1])


def des_manual_encrypt(text, key):
    text = pad(text)
    subkeys = generate_subkeys(key)
    result = ""

    for i in range(0, len(text), 8):
        block = [ord(c) for c in text[i:i+8]]
        left, right = block[:4], block[4:]
```

des_manual.py

Sonuç olarak, AES ve DES algoritmalarında kütüphane tabanlı ve manuel uygulamaların farklı şifreli çıktılar üretmesi beklenen ve normal bir durumdur. Kütüphane tabanlı uygulamalar gerçek dünya güvenlik standartlarına uygunken, manuel uygulamalar eğitim ve algoritma yapısını anlama amacı taşımaktadır.
Bu çalışma sayesinde, modern kriptografik sistemlerde yalnızca algoritmanın değil, kullanılan modların, IV yapısının ve padding mekanizmalarının da güvenlik açısından ne kadar kritik olduğu uygulamalı olarak gözlemlenmiştir.

DES, 56-bit anahtar uzunluğuna sahip, günümüzde güvenliği zayıf kabul edilen bir simetrik şifreleme algoritmasıdır. Ancak kriptografinin tarihsel gelişimi açısından önemli bir yere sahiptir.
DES algoritması 64-bit bloklar üzerinde çalışmakta ve Feistel yapısını kullanmaktadır. Bu çalışmada DES, karşılaştırma ve eğitim amaçlı olarak kullanılmıştır.
AES'e benzer şekilde, DES algoritması da projede iki farklı biçimde uygulanmıştır:

- **Kütüphane Tabanlı DES**
- **Manuel DES**

Yapılan deneylerde DES'in AES'e kıyasla daha kısa anahtar uzunluğu ve eski tasarımı nedeniyle güvenlik açısından zayıf olduğu gözlemlenmiştir.

5.3 RSA (Rivest–Shamir–Adleman)

RSA, asimetrik şifreleme algoritmaları arasında en yaygın kullanılan yöntemlerden biridir. RSA, büyük asal sayılara dayalı matematiksel yapısı sayesinde güvenli anahtar dağıtımı sağlamaktadır.
Bu projede RSA algoritması, doğrudan mesaj şifreleme amacıyla değil, **simetrik anahtarın güvenli bir şekilde paylaşılması** amacıyla kullanılmıştır. Bu yaklaşım, gerçek dünya uygulamalarında yaygın olarak kullanılan hibrit kriptografi modelini temsil etmektedir.
İstemci tarafında oluşturulan simetrik anahtar, RSA açık anahtarı ile şifrelenerek sunucuya gönderilmekte, sunucu tarafında ise RSA gizli anahtarı kullanılarak çözülmektedir.

6. KLASİK ŞİFRELEME ALGORİTMALARI

Bu projede aşağıda listelenen klasik şifreleme algoritmaları uygulanmıştır:

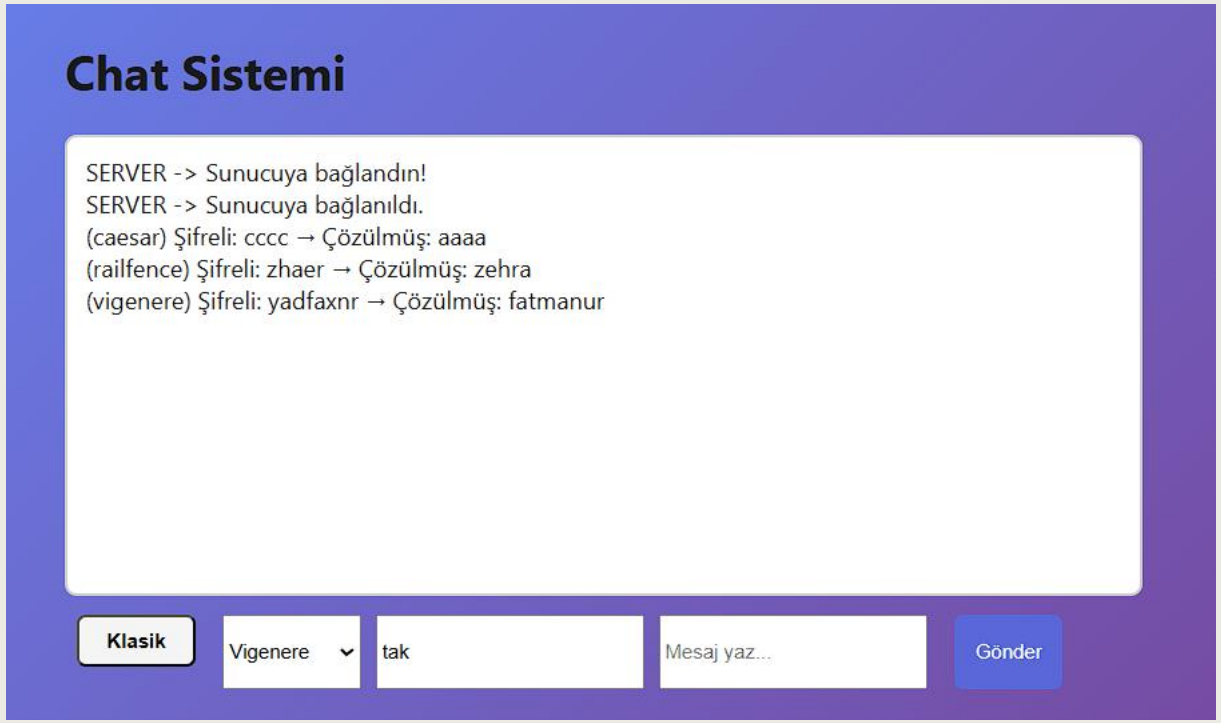
- **Caesar Cipher**
- **Vigenère Cipher**
- **Affine Cipher**
- **Substitution Cipher**
- **Columnar Transposition Cipher**
- **Rail Fence Cipher**
- **Playfair Cipher**
- **Hill Cipher**
- **ROT Cipher**
- **Polybius Square Cipher**
- **Pigpen Cipher**

Projede klasik şifreleme algoritmaları,

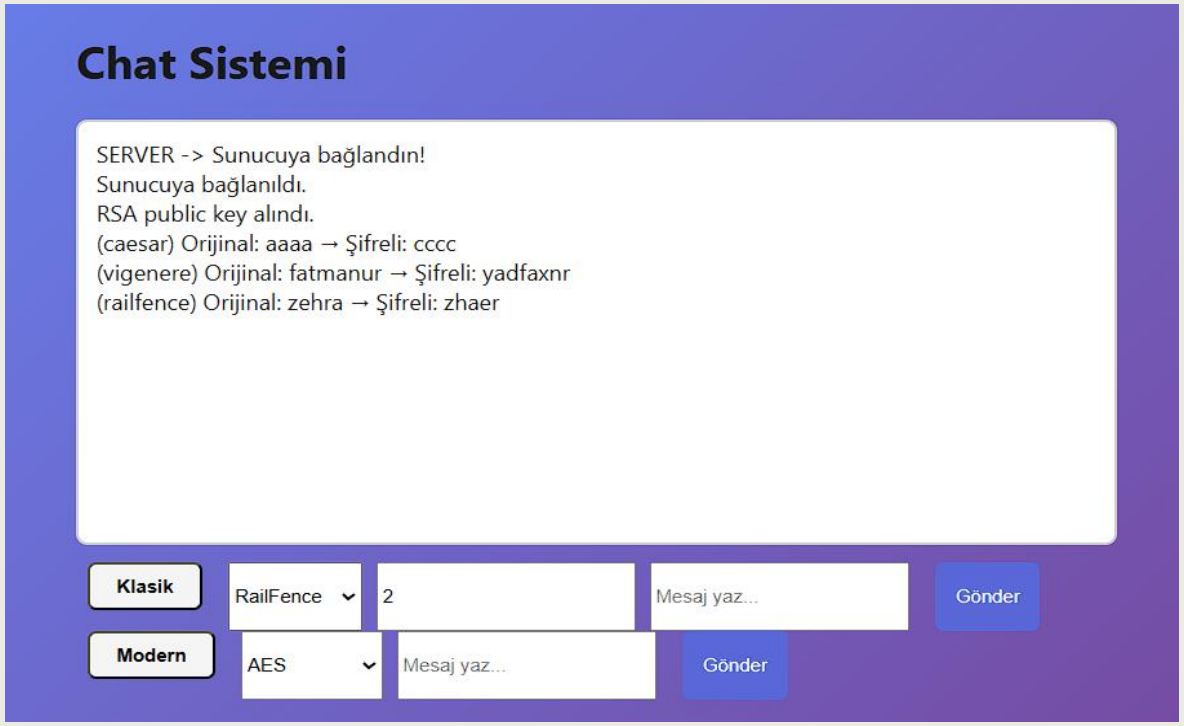
Her bir algoritma, kullanıcıdan alınan düz metin ve anahtar bilgisine göre şifreleme ve deşifreleme işlemlerini gerçekleştirecek şekilde tasarlanmıştır.

Python dili kullanılarak sınıf tabanlı bir yapı ile geliştirilmiştir. Her algoritma için ayrı bir sınıf oluşturulmuş ve bu sınıflar içerisinde encrypt() ve decrypt() metotları tanımlanmıştır.

Flask tabanlı web arayüzü sayesinde kullanıcılar, seçtikleri klasik şifreleme algoritmasını kullanarak metinlerini kolayca şifreleyebilmekte veya çözebilmektedir. Ayrıca Socket.IO destekli sohbet sistemi aracılığıyla, klasik şifreleme algoritmaları gerçek zamanlı mesajlaşma senaryosu içerisinde de test edilmiştir.



Deşifreleme chat kısmı

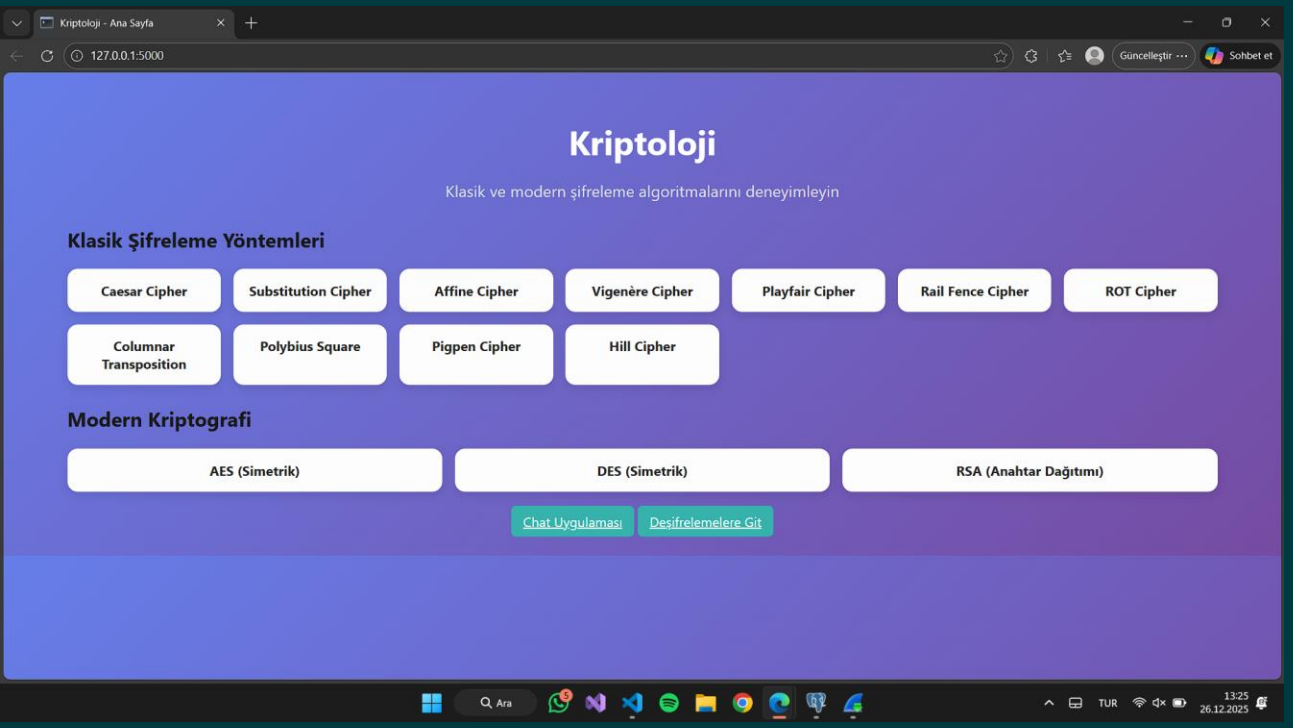


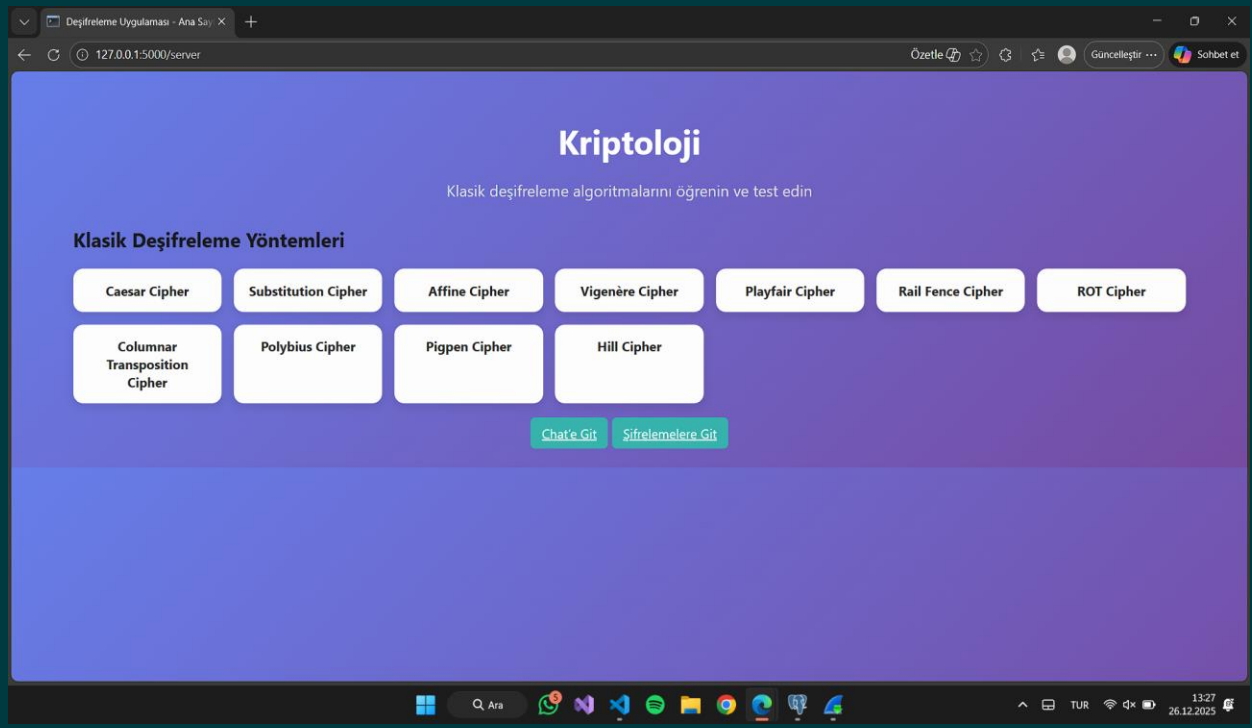
Şifreleme chat kısmı

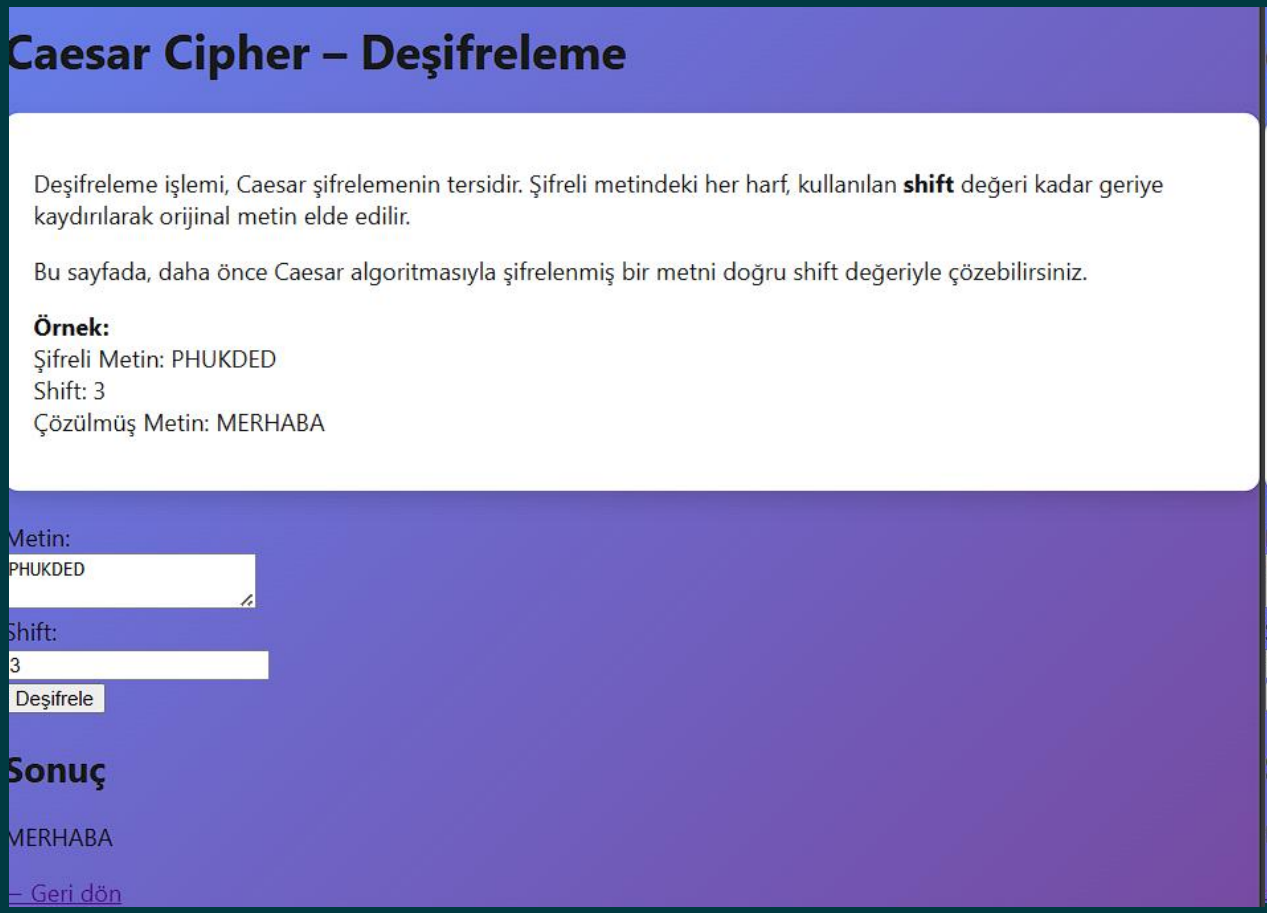
Günümüzde tek başına güvenli kabul edilmeseler de, klasik şifreleme algoritmaları kriptografinin temel mantığını anlamak açısından büyük öneme sahiptir. Bu projede, farklı mantıklara sahip klasik şifreleme algoritmaları hem şifreleme hem de deşifreleme işlemleriyle uygulamalı olarak gerçekleştirilmiştir.

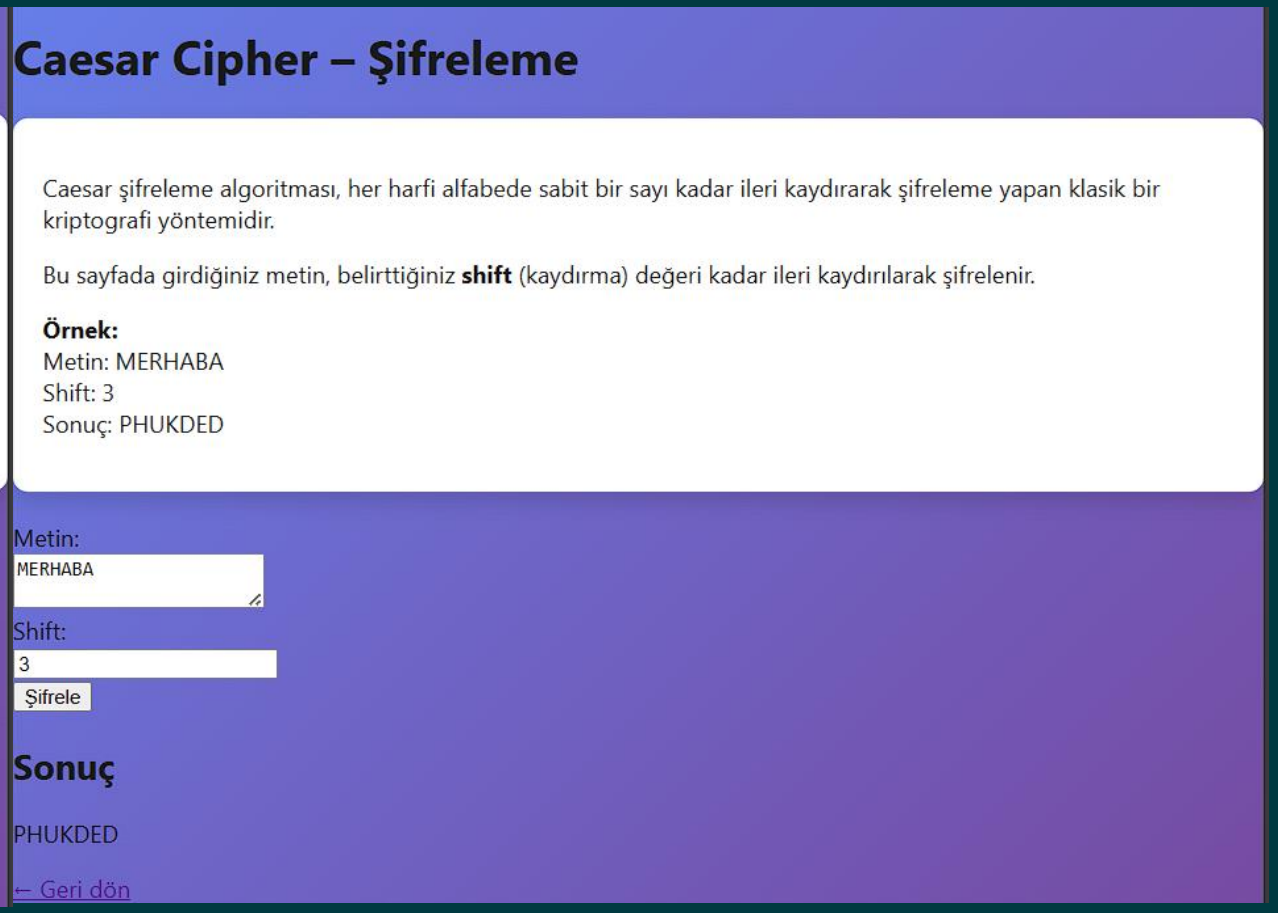
7.KULLANILAN ŞİFRELEMELER İÇİN ÖĞRENİM SAYFALARI

Eğiticilik ve öğrenim kısmı için projenin chat kısmının yetersizliğinden ötürü projeye her şifreleme için ayrı açıklama/blog sayfaları ve şifrelemeyi denemek adına blogların altına formlar oluşturulmuştur. Böylelikle bir şifrelemenin aslında bu projede nasıl işlediği anlatılmış ve uygulaması kullanıcıya bırakılmıştır.

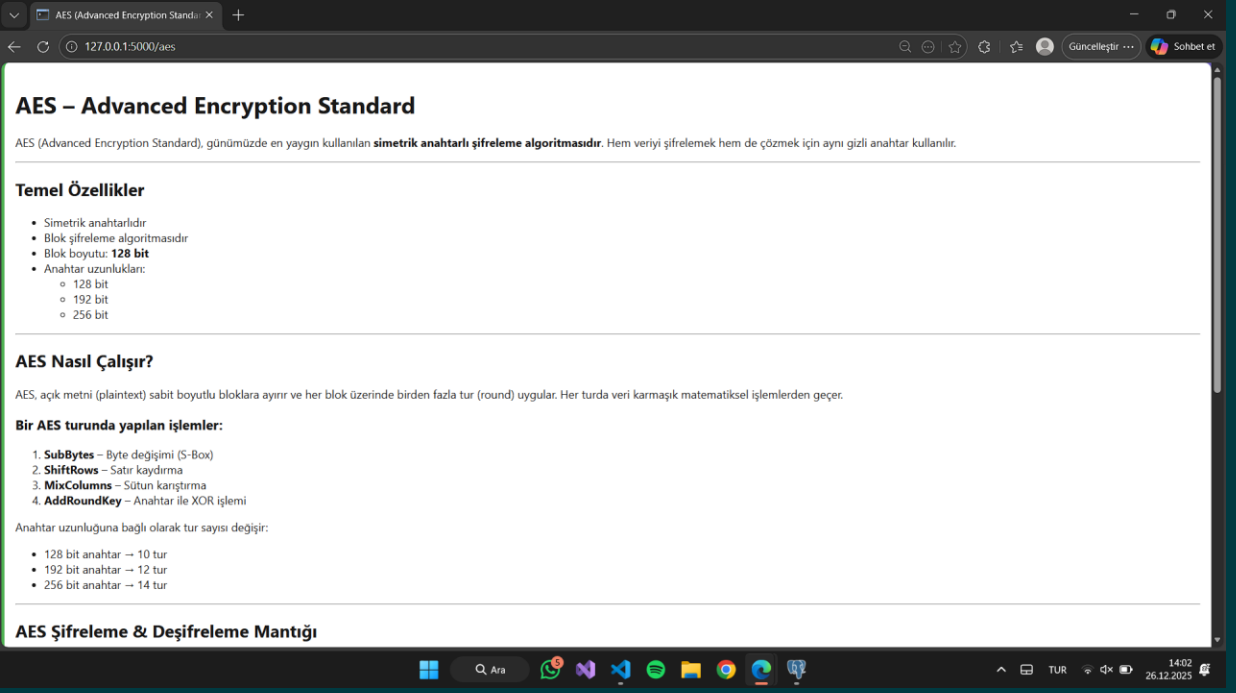


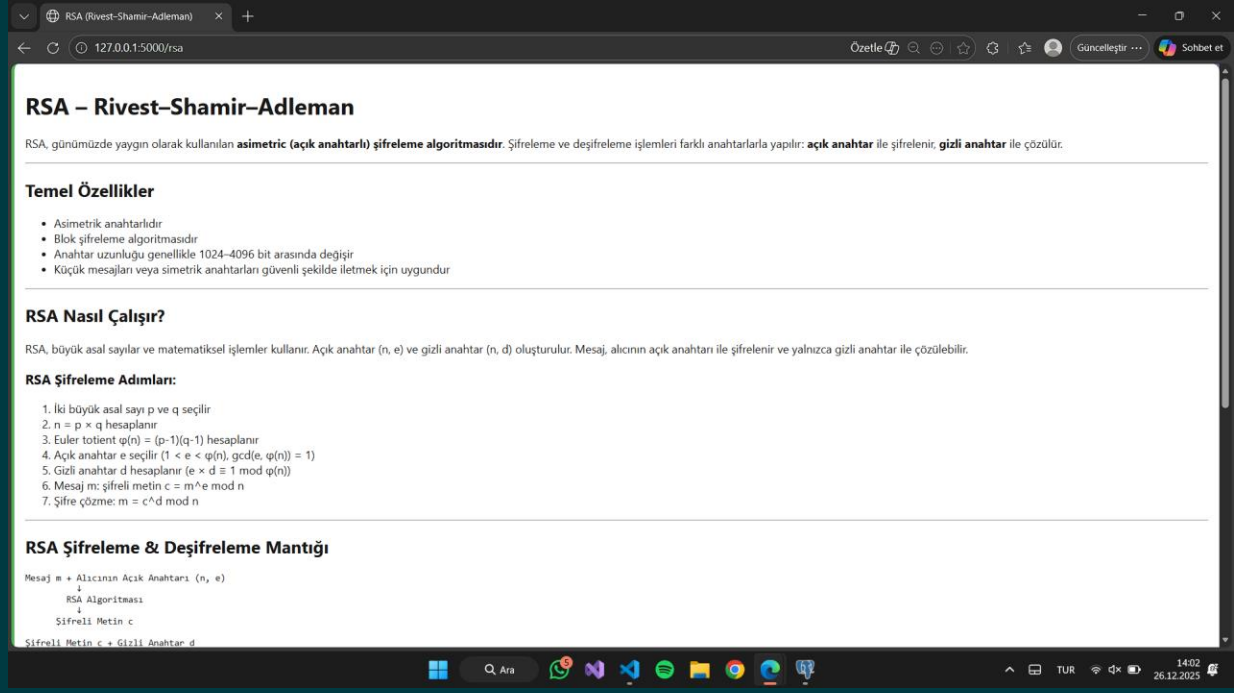


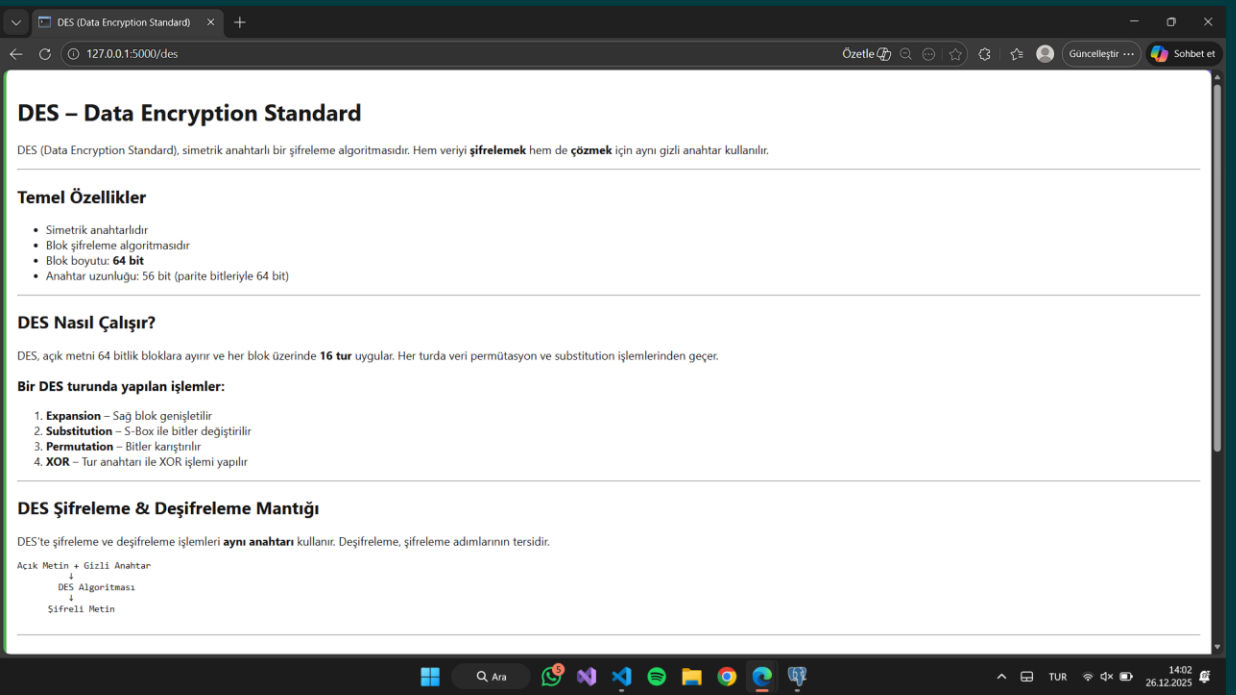




Her şifreleme için templates/ dizini altına kendi html dosyaları ve ciphers/ dizini altına şifreleme algoritmalarını içeren py uzantılı dosya açıklmıştır. Her ikisi de dinamik olarak şifreleme ve deşifreleme kısımlarına göre şekillenmektedir.







Modern şifrelemeler bu adımda unutulmamıştır. Onlarında genel mantık ve öğrenimi kendi sayfalarında gerçekleştirilebilmektedir.

8. WIRESHARK AĞ TRAFİĞİ ANALİZİ

Wireshark kurulumu sırasında, ağ trafiğinin yakalanabilmesi için gerekli sürücüler (Npcap) sisteme dahil edilmiştir. Kurulum tamamlandıktan sonra aktif ağ bağlantısı seçilerek paket yakalama işlemi başlatılmıştır.
Paket yakalama işlemi sırasında proje çalıştırılmış, istemci ve sunucu arasında modern şifreleme algoritmaları kullanılarak mesajlar gönderilmiştir. Bu süreçte Wireshark arka planda tüm ağ trafiğini kaydetmiştir.

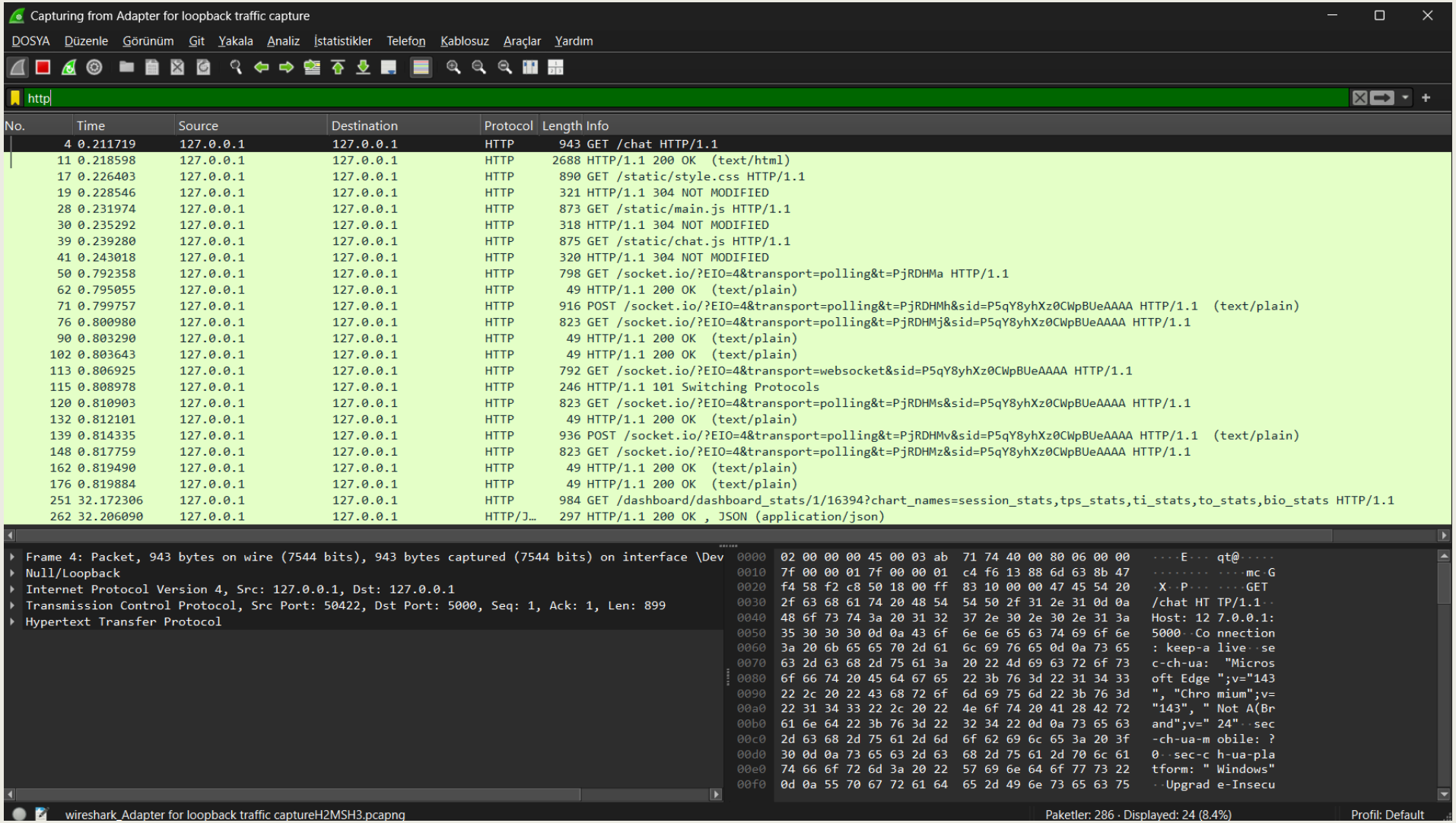
Secure Chat modunda, mesajlar AES ve DES algoritmaları kullanılarak şifrelenmiş ve simetrik anahtar RSA ile korunmuştur. Bu sayede ağ üzerinden iletilen veri, tamamen anlamsız ve rastgele görünen byte dizileri şeklinde taşınmıştır.

Wireshark üzerinde incelenen paketlerde, mesaj içeriğinin okunabilir olmadığı ve Base64 formatında kodlanmış şifreli veriler halinde iletildiği gözlemlenmiştir.

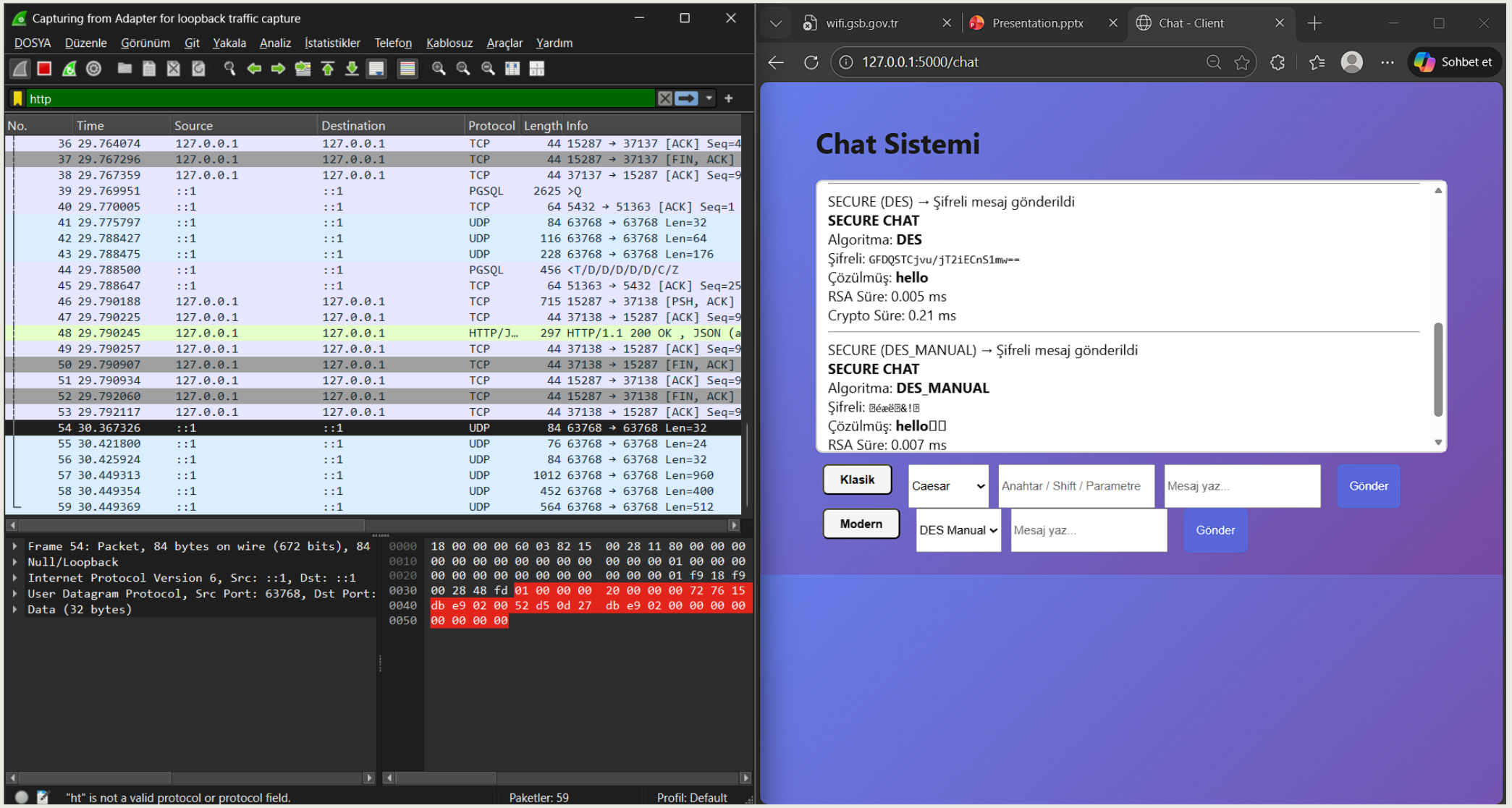
AES ve DES algoritmaları kullanılarak gönderilen mesajlara ait paketler incelendiğinde, veri içeriğinin tamamen anlamsız byte dizileri halinde iletildiği gözlemlenmiştir. Paket içeriğinde düz metne ait herhangi bir okunabilir ifade bulunmamaktadır.

Kullanılan Filtre:
`Http`

8.1 aes_lib ve aes_manuel karşılaştırma



8.2 des_lib ve des_manuel Karşılaştırma



8.3 des_lib ve aes_lib Karşılaştırma