

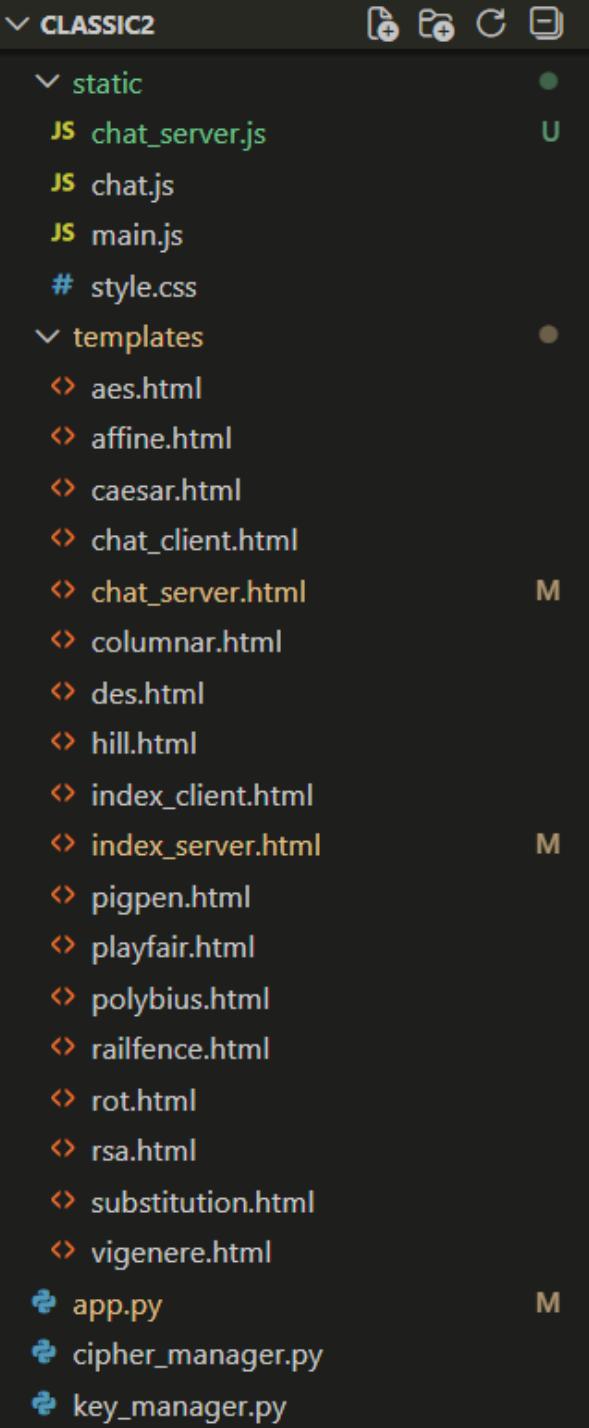
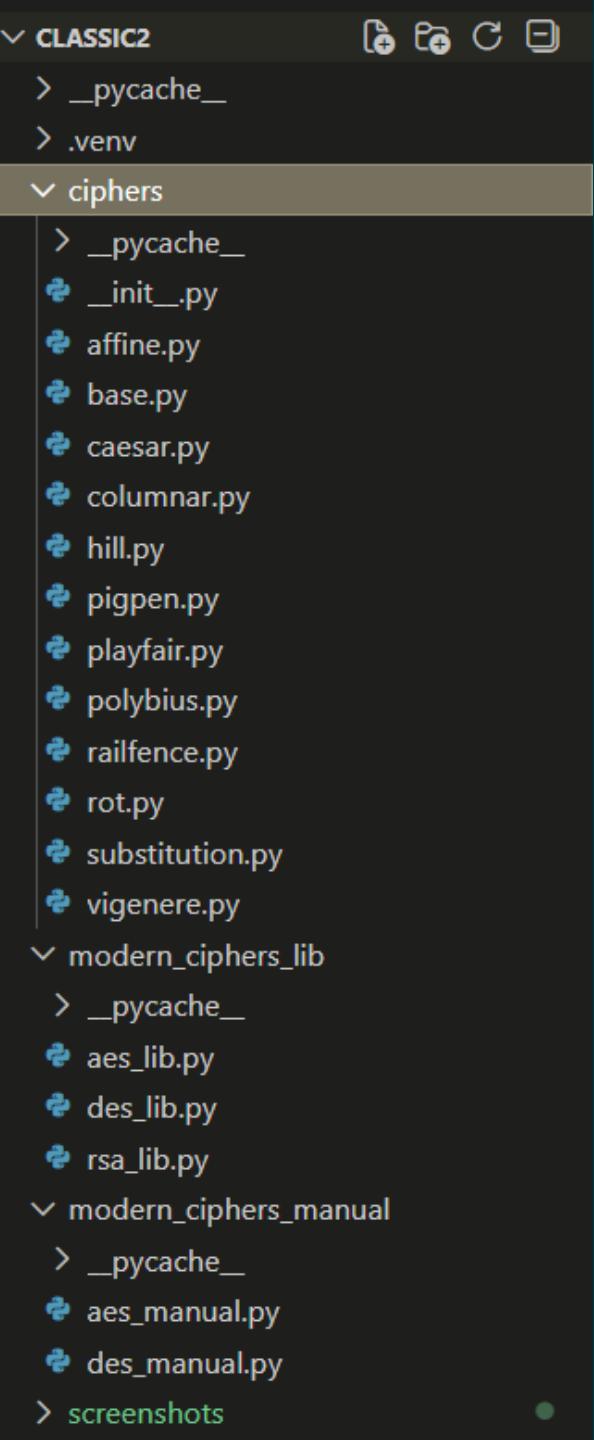
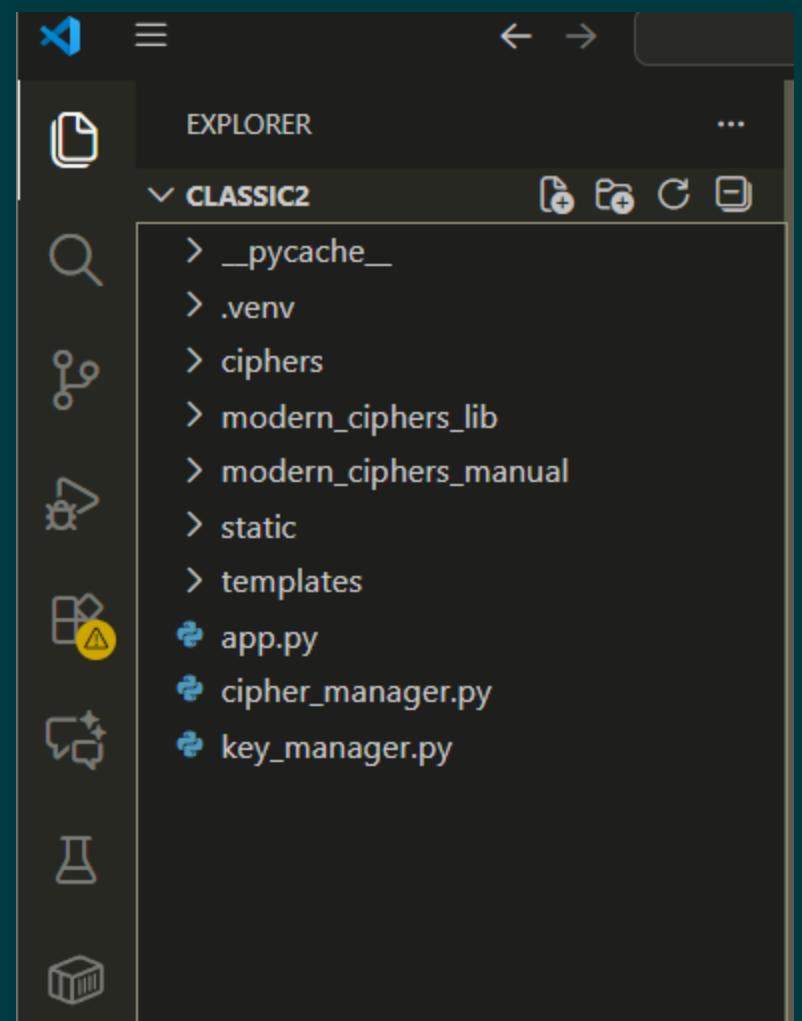
**Hazırlayan:**  
**Fatmanur Zehra Bozkurt**

**Ders / Proje Türü:**  
**Bilgi Güvenliği / Kriptoloji Dersi Projesi**

**Github Bağlantısı:**  
[fatmanurzb/classic2](https://github.com/fatmanurzb/classic2)

# KRIPTOGRAFİ TABANLI GÜVENLİ CHAT VE ÖĞRENMEYE BAŞLANGIÇ UYGULAMASI

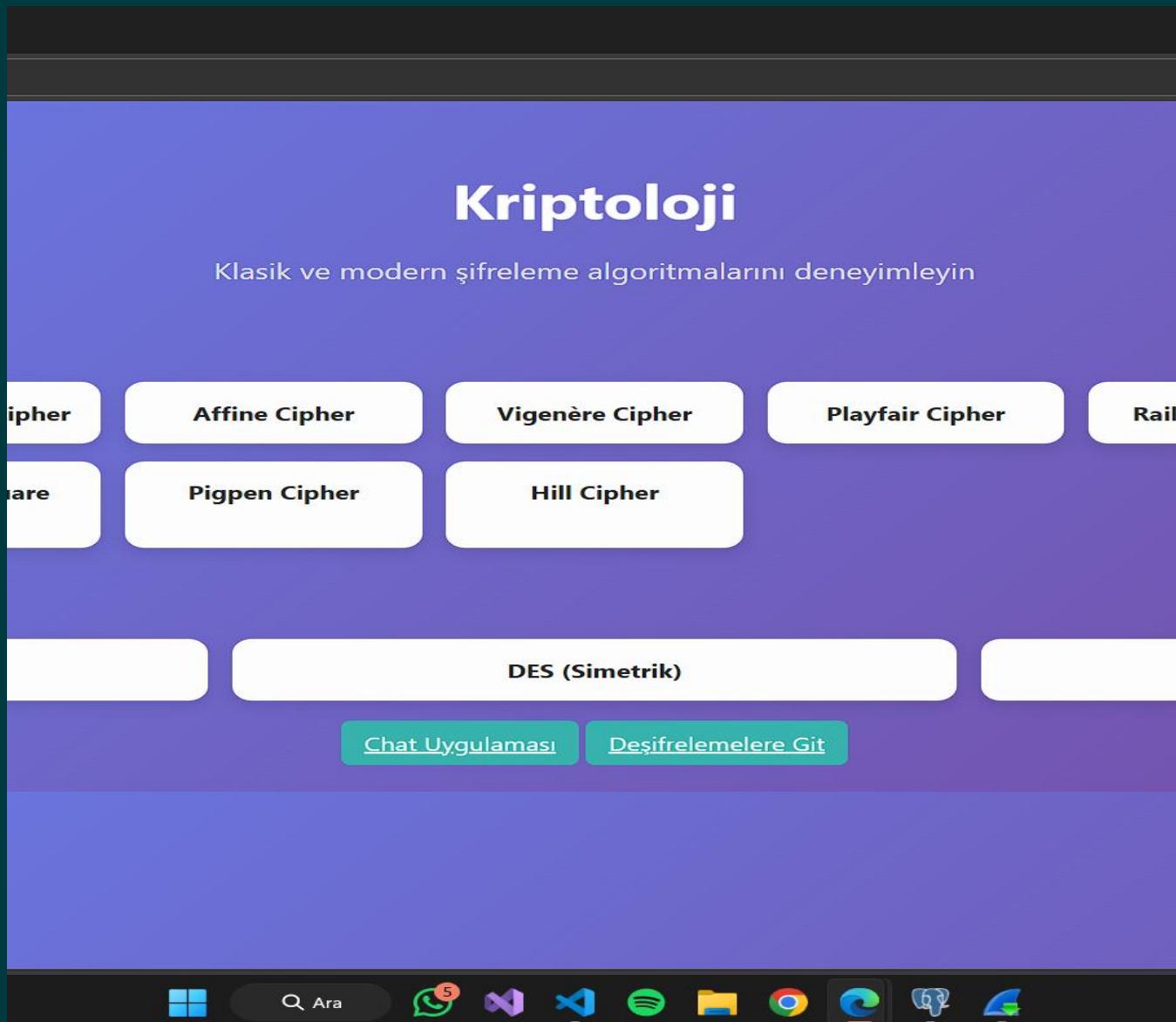
-KLASİK VE MODERN ŞİFRELEME ALGORITMALARININ  
UYGULAMALI ANALİZİ



# 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 iletişimın 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

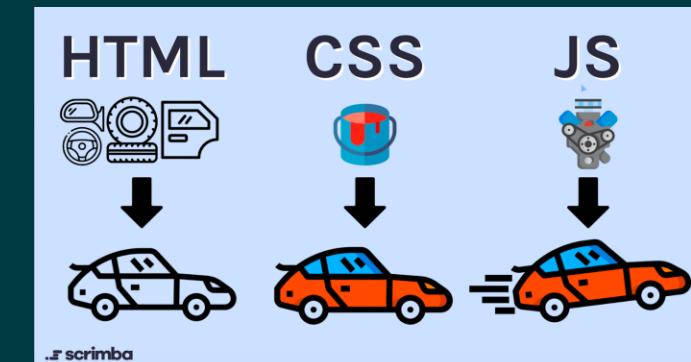
Bu projede, web tabanlı bir istemci–sunucu mimarisi kullanılarak güvenli ve klasik şifreleme algoritmalarını içeren bir haberleşme sistemi geliştirilmiştir.

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



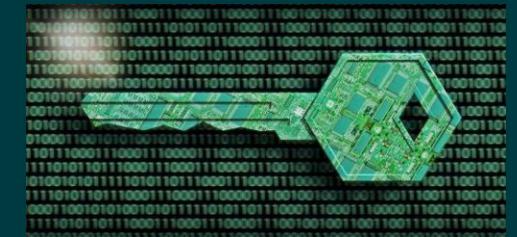
**PyCryptodome kütüphanesi**, modern kriptografi algoritmalarının (AES, DES ve RSA) güvenli ve standartlara uygun şekilde uygulanabilmesi için kullanılmıştır. Bu kütüphane sayesinde gerçek dünyada kullanılan kriptografik işlemler birebir simüle edilmiştir.

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

**Visual Studio Code**, proje geliştirme sürecinde kod editörü olarak tercih edilmiştir. **GitHub** ise proje dosyalarının versyon 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
```

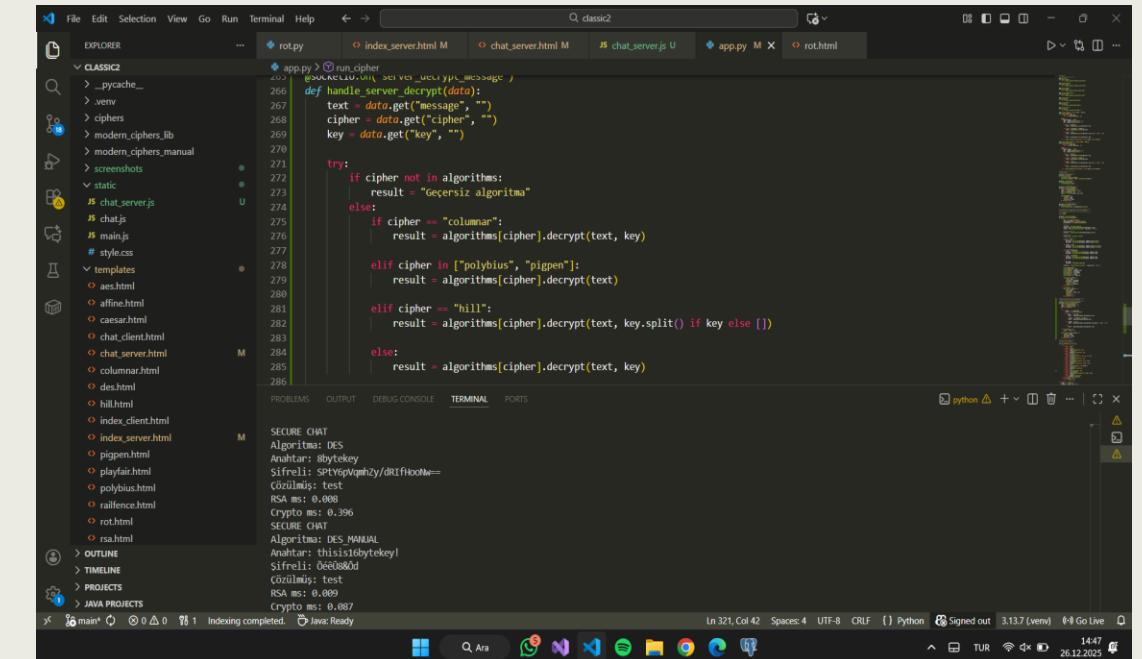
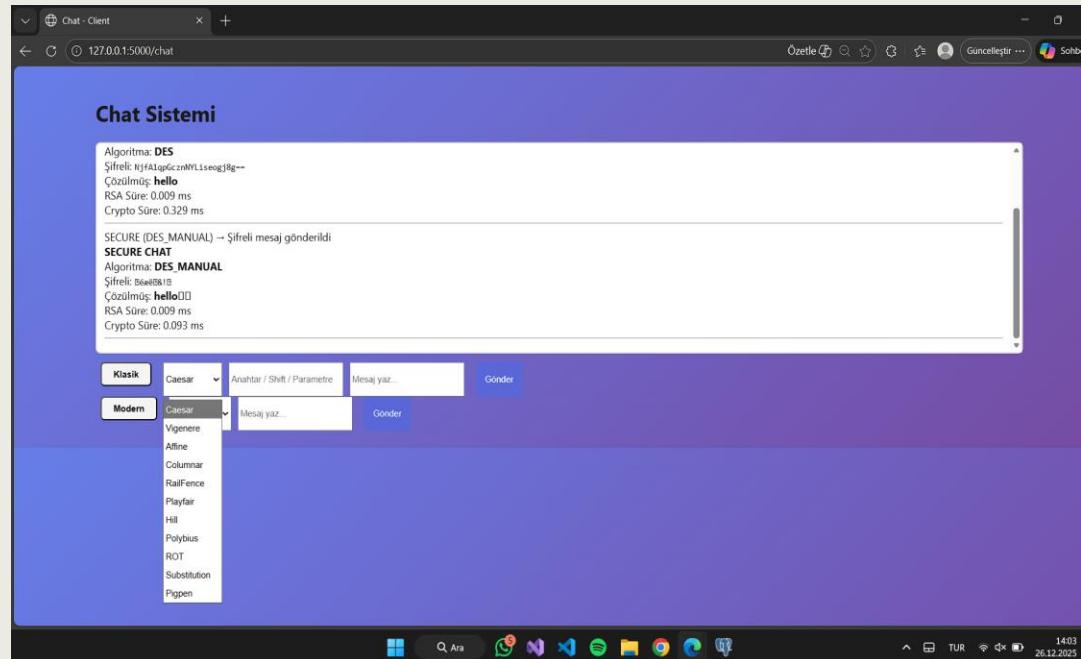


## 4. SİSTEM MİMARİSİ VE GENEL ÇALIŞMA YAPISI

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传递ası 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)

### Chat Sistemi

#### SECURE CHAT

Algoritma: AES

Şifreli: KacZ5naKR9IpQGnq+PiMvFQWAB0t6wv+05qNKy4ndq8=

Çözülmüş: test

RSA Süre: 0.009 ms

Crypto Süre: 0.274 ms

SECURE (AES\_MANUAL) → Şifreli mesaj gönderildi

#### SECURE CHAT

Algoritma: AES\_MANUAL

Şifreli: tμ@tV·TXW6Ö:EÜ+

Çözülmüş: test↑□ ☺☺☺☺☺☺

RSA Süre: 0.149 ms

Crypto Süre: 0.129 ms



```
modern_ciphers_lib > aes_lib.py > aes.decrypt
1  from Crypto.Cipher import AES
2  from Crypto.Random import get_random_bytes
3  from Crypto.Util.Padding import pad, unpad
4  import base64
5
6  def aes_encrypt(plaintext: str, key: bytes) -> str:
7      iv = get_random_bytes(16)
8      cipher = AES.new(key, AES.MODE_CBC, iv)
9      ciphertext = cipher.encrypt(pad(plaintext.encode(), AES.block_size))
10     return base64.b64encode(iv + ciphertext).decode()
11
12 def aes_decrypt(ciphertext_b64: str, key: bytes) -> str:
13     data = base64.b64decode(ciphertext_b64)
14     iv = data[:16]
15     ciphertext = data[16:]
16     cipher = AES.new(key, AES.MODE_CBC, iv)
17     return unpad(cipher.decrypt(ciphertext), AES.block_size).decode()
```

aes\_lib.py

```
modern_ciphers_manual > aes_manual.py > inv_shift_rows
1
2  # Basit S-Box (gerçek AES S-box DEĞİL)
3  S_BOX = {
4      i: (i * 7 + 3) % 256 for i in range(256)
5  }
6
7  INV_S_BOX = {v: k for k, v in S_BOX.items()}
8
9
10 def sub_bytes(block):
11     return [S_BOX[b] for b in block]
12
13
14 def inv_sub_bytes(block):
15     return [INV_S_BOX[b] for b in block]
16
17
18 def shift_rows(block):
19     # 4x4 matris varsayımlı (AES mantığı)
20     return [
21         block[0], block[1], block[2], block[3],
22         block[5], block[6], block[7], block[4],
23         block[10], block[11], block[8], block[9],
24         block[15], block[12], block[13], block[14]
25     ]
26
27
28 def inv_shift_rows(block):
29     return [
30         block[0], block[1], block[2], block[3],
31         block[7], block[4], block[5], block[6],
32         block[10], block[11], block[8], block[9],
33         block[13], block[14], block[15], block[12]
34     ]
35
36
37 def add_round_key(block, key):
```

aes\_manual.py

“Manuel AES uygulaması, algoritmanın eğitimsel açıdan anlaşılması 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 eğitsel açıdan daha öğretici olduğu gözlemlenmiştir.

## 5.3 DES (Data Encryption Standard)

### Chat Sistemi

SECURE (DES) → Şifreli mesaj gönderildi

#### SECURE CHAT

Algoritma: DES

Şifreli: SPtY6pVqmhZy/dRIfHooNw==

Çözülmüş: test

RSA Süre: 0.008 ms

Crypto Süre: 0.396 ms

SECURE (DES\_MANUAL) → Şifreli mesaj gönderildi

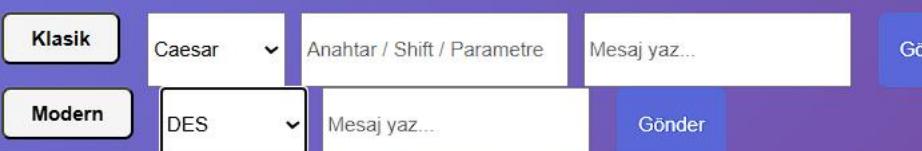
#### SECURE CHAT

Algoritma: DES\_MANUAL

Şifreli: öééÜ8&ôd

Çözülmüş: test

RSA Süre: 0.009 ms



```
modern_ciphers_lib > des_lib.py > ...
1  ✓ from Crypto.Cipher import DES
2   from Crypto.Util.Padding import pad, unpad
3   from Crypto.Random import get_random_bytes
4   import base64
5
6   ✓ def des_encrypt(plaintext: str, key: bytes) -> str:
7     if len(key) != 8:
8       raise ValueError("DES key must be exactly 8 bytes")
9
10    iv = get_random_bytes(8)
11    cipher = DES.new(key, DES.MODE_CBC, iv)
12    ciphertext = cipher.encrypt(pad(plaintext.encode(), 8))
13
14    return base64.b64encode(iv + ciphertext).decode()
15
16
17   ✓ def des_decrypt(ciphertext_b64: str, key: bytes) -> str:
18     if len(key) != 8:
19       raise ValueError("DES key must be exactly 8 bytes")
20
21     raw = base64.b64decode(ciphertext_b64)
22     iv = raw[:8]
23     ciphertext = raw[8:]
24
25     cipher = DES.new(key, DES.MODE_CBC, iv)
26     plaintext = unpad(cipher.decrypt(ciphertext), 8)
27
28
29    return plaintext.decode()
```

des\_lib.py

```
modern_ciphers_manual > des_manual.py > generate_subkeys
1
2  def xor(a, b):
3    return [i ^ j for i, j in zip(a, b)]
4
5
6  def feistel_function(right, subkey):
7    # Basit substitution + XOR
8    return [(r + k) % 256 for r, k in zip(right, subkey)]
9
10
11  def generate_subkeys(key):
12    key_bytes = [ord(c) for c in key.ljust(8)[:8]]
13    return [
14      key_bytes[i:] + key_bytes[:i]
15      for i in range(4)
16    ]
17
18
19  def pad(text):
20    while len(text) % 8 != 0:
21      text += chr(8 - (len(text) % 8))
22    return text
23
24
25  def unpad(text):
26    return text.rstrip(text[-1])
27
28
29  def des_manual_encrypt(text, key):
30    text = pad(text)
31    subkeys = generate_subkeys(key)
32    result = ""
33
34    for i in range(0, len(text), 8):
35      block = [ord(c) for c in text[i:i+8]]
36      left, right = block[:4], block[4:]
37
```

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 anlamayı 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 tasarıımı 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 sayılaraya dayalı matematiksel yapısı sayesinde güvenli anahtar dağıtıımı 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 anahtarları 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() metodları 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.

## Chat Sistemi

SERVER -> Sunucuya bağlandı!  
SERVER -> Sunucuya bağlanıldı.  
(caesar) Şifreli: cccc → Çözülmüş: aaaa  
(railfence) Şifreli: zhaer → Çözülmüş: zehra  
(vigenere) Şifreli: yadfaxnr → Çözülmüş: fatmanur

Klasik

Vigenere ▾ tak

Mesaj yaz...

Gönder

Deşifreleme chat kısmı

## Chat Sistemi

SERVER -> Sunucuya bağlandı!  
Sunucuya bağlanıldı.  
RSA public key alındı.  
(caesar) Orijinal: aaaa → Şifreli: cccc  
(vigenere) Orijinal: fatmanur → Şifreli: yadfaxnr  
(railfence) Orijinal: zehra → Şifreli: zhaer

Klasik

RailFence ▾ 2

Mesaj yaz...

Gönder

Modern

AES ▾

Mesaj yaz...

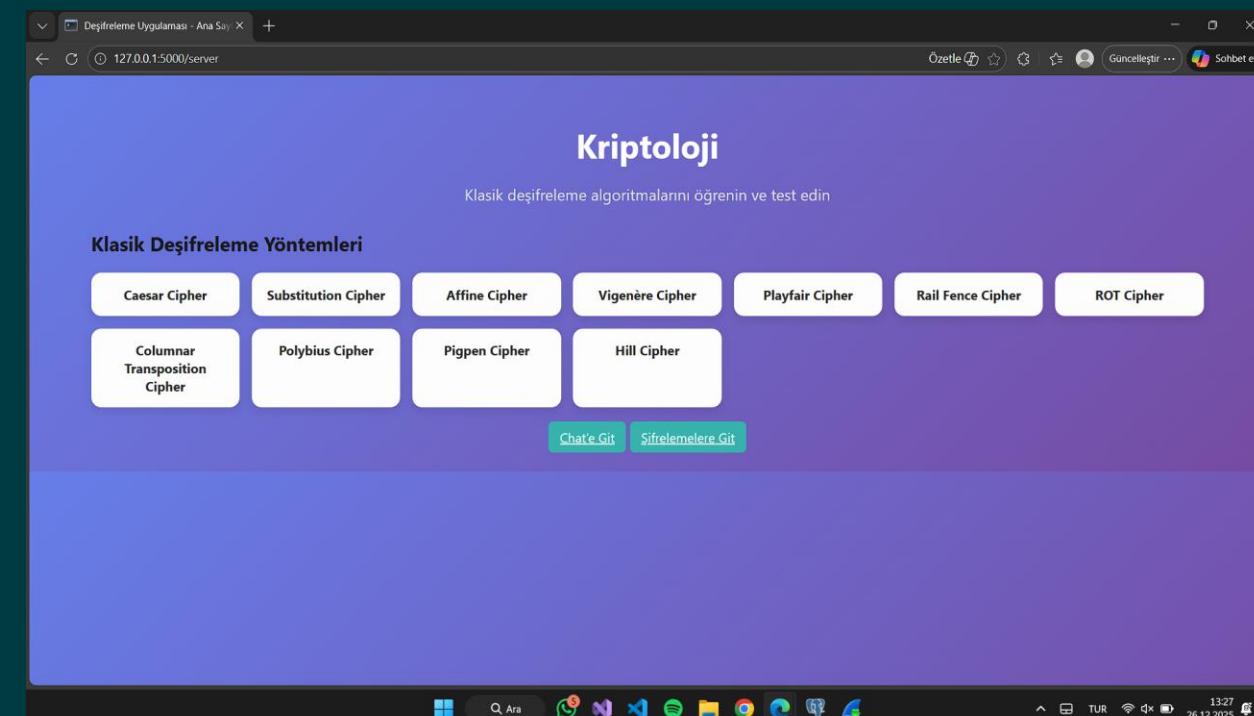
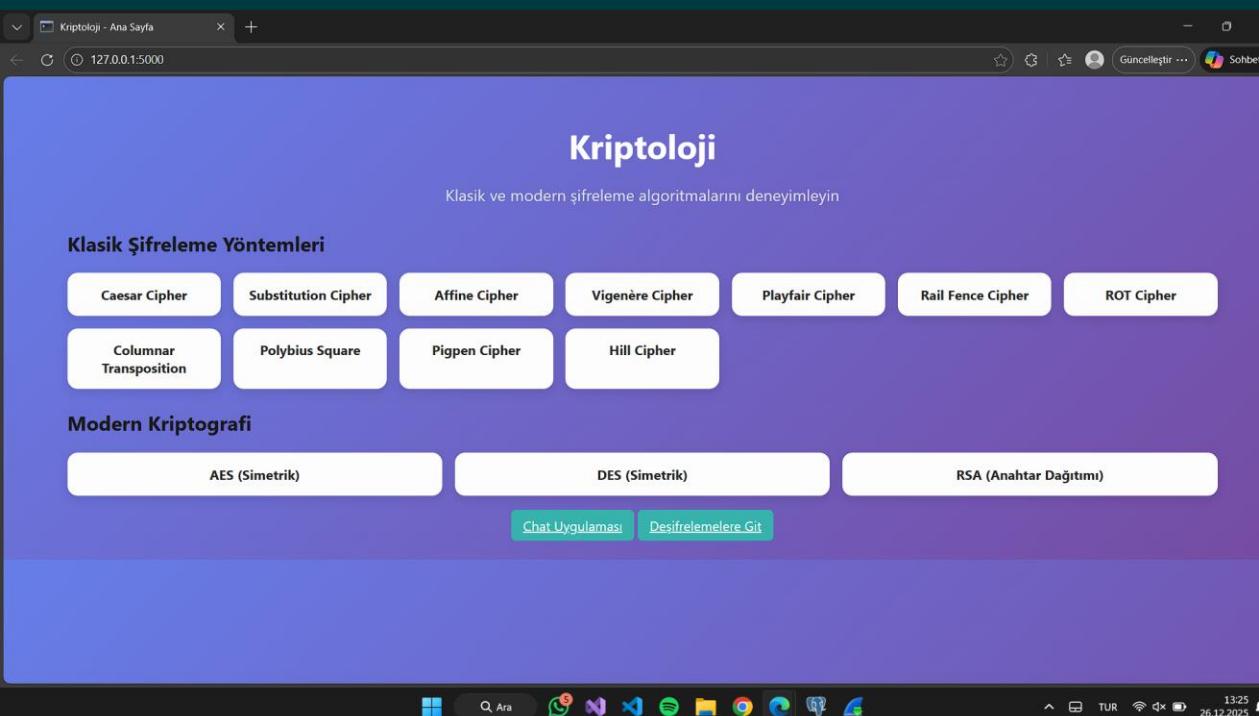
Gönder

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 gerçekleştirılmıştır.

# 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.



## Caesar Cipher – Deşifreleme

Deşifreleme işlemi, Caesar şifrelemenin tersidir. Şifreli metindeki her harf, kullanılan **shift** değeri kadar geriye kaydırılarak orijinal metin elde edilir.

Bu sayfada, daha önce Caesar algoritmasıyla şifrelenmiş bir metni doğru shift değeriyle çözebilirsiniz.

### Örnek:

Şifreli Metin: PHUKDED

Shift: 3

Çözülmüş Metin: MERHABA

Metin:

PHUKDED

Shift:

3

Deşifrele

Sonuç

MERHABA

[← Geri dön](#)

## Caesar Cipher – Şifreleme

Caesar şifreleme algoritması, her harfi alfabede sabit bir sayı kadar ileri kaydırarak şifreleme yapan klasik bir kriptografi yöntemidir.

Bu sayfada girdığınız metin, belirttiğiniz **shift** (kaydırma) değeri kadar ileri kaydırılarak şifrelenir.

### Örnek:

Metin: MERHABA

Shift: 3

Sonuç: PHUKDED

Metin:

MERHABA

Shift:

3

Şifrele

Sonuç

PHUKDED

[← Geri dön](#)

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

## AES – Advanced Encryption Standard

AES (Advanced Encryption Standard), günümüzde yaygın kullanılan **simetrik anahtarlı şifreleme algoritmasıdır**. Hem veriyi şifrelemek hem de çözmek için aynı gizli anahtar kullanılır.

### Temel Özellikler

- Simetrik anahtarlıdır
- Blok şifreleme algoritmasıdır
- Blok boyutu: **128 bit**
- Anahtar uzunlukları:
  - 128 bit
  - 192 bit
  - 256 bit

### AES Nasıl Çalışır?

AES, açık metni (plaintext) sabit boyutlu bloklara ayırır ve her blok üzerinde birden fazla tur (round) uygular. Her turda veri karmaşık matematiksel işlemlerden geçer.

#### Bir AES turunda yapılan işlemler:

1. **SubBytes** – Byte değişimi (S-Box)
2. **ShiftRows** – Satır kaydırma
3. **MixColumns** – Sütün karıştırma
4. **AddRoundKey** – Anahtar ile XOR işlemi

Anahtar uzunluğuna bağlı olarak tur sayısı değişir:

- 128 bit anahtar → 10 tur
- 192 bit anahtar → 12 tur
- 256 bit anahtar → 14 tur

### AES Şifreleme & Deşifreleme Mantığı



14:02 26.12.2025

## RSA – Rivest–Shamir–Adleman

RSA, günümüzde yaygın olarak kullanılan **asimetrik (açık anahtarlı) şifreleme algoritmasıdır**. Şifreleme ve deşifreleme işlemleri farklı anahtarlarla yapılır: **açık anahtar** ile şifrelenir, **gizli anahtar** ile çözülür.

### Temel Özellikler

- Asimetrik anahtarlıdır
- Blok şifreleme algoritmasıdır
- Anahtar uzunluğu genellikle 1024–4096 bit arasında değişir
- Küçük mesajları veya simetrik anahtarları güvenli şekilde iletmek için uygundur

### RSA Nasıl Çalışır?

RSA, büyük asal sayılar ve matematiksel işlemler kullanır. Açık anahtar ( $n, e$ ) ve gizli anahtar ( $n, d$ ) oluşturulur. Mesaj, alıcıının açık anahtarı ile şifrelenir ve yalnızca gizli anahtar ile çözülebilir.

#### RSA Şifreleme Adımları:

1. İki büyük asal sayı  $p$  ve  $q$  seçilir
2.  $n = p \times q$  hesaplanır
3. Euler totient  $\phi(n) = (p-1)(q-1)$  hesaplanır
4. Açık anahtar  $e$  seçilir ( $1 < e < \phi(n)$ ,  $\text{gcd}(e, \phi(n)) = 1$ )
5. Gizli anahtar  $d$  hesaplanır ( $e \times d \equiv 1 \pmod{\phi(n)}$ )
6. Mesaj  $m$ : şifreli metin  $c = m^e \pmod{n}$
7. Şifre çözme:  $m = c^d \pmod{n}$

### RSA Şifreleme & Deşifreleme Mantığı

```
Mesaj m + Alıcıın Açık Anahtarı (n, e)
↓
RSA Algoritması
↓
Şifreli Metin c
↓
Şifreli Metin c + Gizli Anahtar d
```



14:02 26.12.2025

## DES – Data Encryption Standard

DES (Data Encryption Standard), simetrik anahtarlı bir şifreleme algoritmasıdır. Hem veriyi **şifrelemek** hem de **çözmek** için aynı gizli anahtar kullanılır.

### Temel Özellikler

- Simetrik anahtarlıdır
- Blok şifreleme algoritmasıdır
- Blok boyutu: **64 bit**
- Anahtar uzunluğu: 56 bit (parite bittiyle 64 bit)

### DES Nasıl Çalışır?

DES, açık metni 64 bitlik bloklara ayırır ve her blok üzerinde **16 tur** uygular. Her turda veri permütasyon ve substitution işlemlerinden geçer.

#### Bir DES turunda yapılan işlemler:

1. **Expansion** – Sağ blok genişletilir
2. **Substitution** – S-Box ile bitler değiştirilir
3. **Permutation** – Bitler karıştırılır
4. **XOR** – Tur anahtarı ile XOR işlemi yapılır

### DES Şifreleme & Deşifreleme Mantığı

DES'te şifreleme ve deşifreleme işlemleri **aynı anahtar** kullanır. Deşifreleme, şifreleme adımlarının tersidir.

```
Açık Metin + Gizli Anahtar
↓
DES Algoritması
↓
Şifreli Metin
```



14:02 26.12.2025

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 iletiliği gözlemlenmiştir.

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

Kullanılan Filtre:

Http

## 8.1 aes\_lib ve aes\_manuel karşılaştırma

The screenshot shows a Wireshark interface capturing traffic from the Adapter for loopback traffic capture. The main window displays a list of network packets, mostly HTTP requests and responses, originating from and destined to 127.0.0.1. The bottom pane provides a detailed view of the selected packet (Frame 4), showing its hex and ASCII representations. The ASCII dump reveals a series of characters corresponding to the captured HTTP traffic.

Frame 4: Packet, 943 bytes on wire (7544 bits), 943 bytes captured (7544 bits) on interface \Device\Null/Loopback

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Transmission Control Protocol, Src Port: 50422, Dst Port: 5000, Seq: 1, Ack: 1, Len: 899

Hypertext Transfer Protocol

No.	Time	Source	Destination	Protocol	Length	Info
4	0.211719	127.0.0.1	127.0.0.1	HTTP	943	GET /chat HTTP/1.1
11	0.218598	127.0.0.1	127.0.0.1	HTTP	2688	HTTP/1.1 200 OK (text/html)
17	0.226403	127.0.0.1	127.0.0.1	HTTP	890	GET /static/style.css HTTP/1.1
19	0.228546	127.0.0.1	127.0.0.1	HTTP	321	HTTP/1.1 304 NOT MODIFIED
28	0.231974	127.0.0.1	127.0.0.1	HTTP	873	GET /static/main.js HTTP/1.1
30	0.235292	127.0.0.1	127.0.0.1	HTTP	318	HTTP/1.1 304 NOT MODIFIED
39	0.239280	127.0.0.1	127.0.0.1	HTTP	875	GET /static/chat.js HTTP/1.1
41	0.243018	127.0.0.1	127.0.0.1	HTTP	320	HTTP/1.1 304 NOT MODIFIED
50	0.792358	127.0.0.1	127.0.0.1	HTTP	798	GET /socket.io/?EIO=4&transport=polling&t=PjRDHMa HTTP/1.1
62	0.795055	127.0.0.1	127.0.0.1	HTTP	49	HTTP/1.1 200 OK (text/plain)
71	0.799757	127.0.0.1	127.0.0.1	HTTP	916	POST /socket.io/?EIO=4&transport=polling&t=PjRDHm&sid=P5qY8yhXz0CwpuBueAAAA HTTP/1.1 (text/plain)
76	0.800980	127.0.0.1	127.0.0.1	HTTP	823	GET /socket.io/?EIO=4&transport=polling&t=PjRDHmj&sid=P5qY8yhXz0CwpuBueAAAA HTTP/1.1
90	0.803290	127.0.0.1	127.0.0.1	HTTP	49	HTTP/1.1 200 OK (text/plain)
102	0.803643	127.0.0.1	127.0.0.1	HTTP	49	HTTP/1.1 200 OK (text/plain)
113	0.806925	127.0.0.1	127.0.0.1	HTTP	792	GET /socket.io/?EIO=4&transport=websocket&sid=P5qY8yhXz0CwpuBueAAAA HTTP/1.1
115	0.808978	127.0.0.1	127.0.0.1	HTTP	246	HTTP/1.1 101 Switching Protocols
120	0.810903	127.0.0.1	127.0.0.1	HTTP	823	GET /socket.io/?EIO=4&transport=polling&t=PjRDHMs&sid=P5qY8yhXz0CwpuBueAAAA HTTP/1.1
132	0.812101	127.0.0.1	127.0.0.1	HTTP	49	HTTP/1.1 200 OK (text/plain)
139	0.814335	127.0.0.1	127.0.0.1	HTTP	936	POST /socket.io/?EIO=4&transport=polling&t=PjRDHm&sid=P5qY8yhXz0CwpuBueAAAA HTTP/1.1 (text/plain)
148	0.817759	127.0.0.1	127.0.0.1	HTTP	823	GET /socket.io/?EIO=4&transport=polling&t=PjRDHmz&sid=P5qY8yhXz0CwpuBueAAAA HTTP/1.1
162	0.819490	127.0.0.1	127.0.0.1	HTTP	49	HTTP/1.1 200 OK (text/plain)
176	0.819884	127.0.0.1	127.0.0.1	HTTP	49	HTTP/1.1 200 OK (text/plain)
251	32.172306	127.0.0.1	127.0.0.1	HTTP	984	GET /dashboard/dashboard_stats/1/16394?chart_names=session_stats,tps_stats,ti_stats,to_stats,bio_stats HTTP/1.1
262	32.206090	127.0.0.1	127.0.0.1	HTTP/J...	297	HTTP/1.1 200 OK , JSON (application/json)

## **8.2 des\_lib ve des\_manuel Karşılaştırma**

Capturing from Adapter for loopback traffic capture

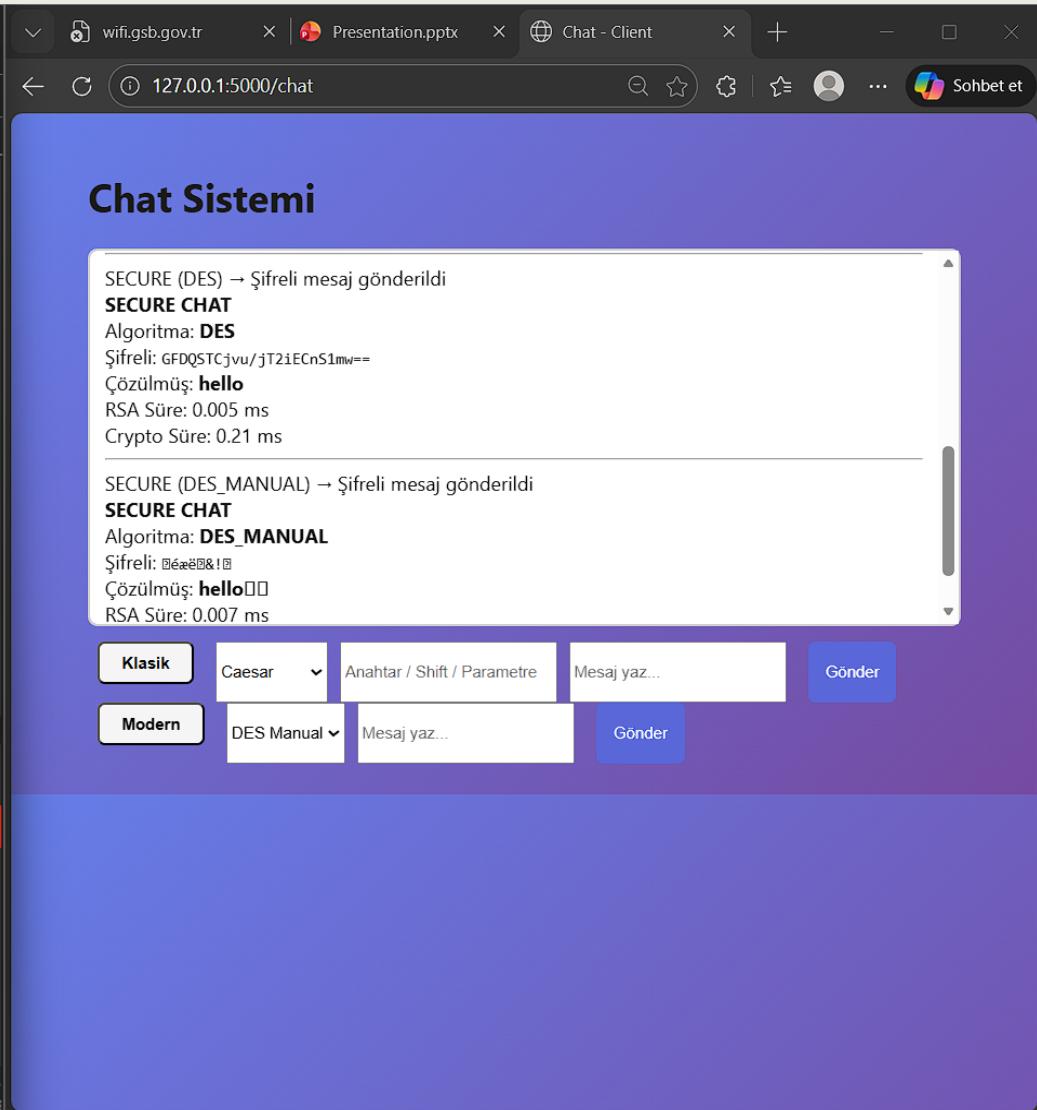
DOSYA Düzleme Görünüm Git Yakala Analiz İstatistikler Telefon Kablosuz Araçlar Yardım

http

No.	Time	Source	Destination	Protocol	Length	Info
36	29.764074	127.0.0.1	127.0.0.1	TCP	44	15287 → 37137 [ACK] Seq=4
37	29.767296	127.0.0.1	127.0.0.1	TCP	44	15287 → 37137 [FIN, ACK]
38	29.767359	127.0.0.1	127.0.0.1	TCP	44	37137 → 15287 [ACK] Seq=9
39	29.769951	::1	::1	PGSQL	2625	>Q
40	29.770005	::1	::1	TCP	64	5432 → 51363 [ACK] Seq=1
41	29.775797	::1	::1	UDP	84	63768 → 63768 Len=32
42	29.788427	::1	::1	UDP	116	63768 → 63768 Len=64
43	29.788475	::1	::1	UDP	228	63768 → 63768 Len=176
44	29.788500	::1	::1	PGSQL	456	<T/D/D/D/D/C/Z
45	29.788647	::1	::1	TCP	64	51363 → 5432 [ACK] Seq=25
46	29.790188	127.0.0.1	127.0.0.1	TCP	715	15287 → 37138 [PSH, ACK]
47	29.790225	127.0.0.1	127.0.0.1	TCP	44	37138 → 15287 [ACK] Seq=9
48	29.790245	127.0.0.1	127.0.0.1	HTTP/J...	297	HTTP/1.1 200 OK , JSON (a
49	29.790257	127.0.0.1	127.0.0.1	TCP	44	37138 → 15287 [ACK] Seq=9
50	29.790907	127.0.0.1	127.0.0.1	TCP	44	37138 → 15287 [FIN, ACK]
51	29.790934	127.0.0.1	127.0.0.1	TCP	44	15287 → 37138 [ACK] Seq=9
52	29.792060	127.0.0.1	127.0.0.1	TCP	44	15287 → 37138 [FIN, ACK]
53	29.792117	127.0.0.1	127.0.0.1	TCP	44	37138 → 15287 [ACK] Seq=9
54	30.367326	::1	::1	UDP	84	63768 → 63768 Len=32
55	30.421800	::1	::1	UDP	76	63768 → 63768 Len=24
56	30.425924	::1	::1	UDP	84	63768 → 63768 Len=32
57	30.449313	::1	::1	UDP	1012	63768 → 63768 Len=960
58	30.449354	::1	::1	UDP	452	63768 → 63768 Len=400
59	30.449369	::1	::1	UDP	564	63768 → 63768 Len=512

Frame 54: Packet, 84 bytes on wire (672 bits), 84 bytes captured (672 bits) on interface 00:00:00:00:00:00 at 2019-01-01 11:28:11 local, transmitted on interface 00:00:00:00:00:00 at 2019-01-01 11:28:11 local  
Null/Loopback  
Internet Protocol Version 6, Src: ::1, Dst: ::1  
User Datagram Protocol, Src Port: 63768, Dst Port: 37138  
Data (32 bytes)

0000	18 00 00 00 60 03 82 15 00 28 11 80 00 00 00
0010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00
0020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 f9 18 f9
0030	00 28 48 fd 01 00 00 00 20 00 00 00 72 76 15
0040	db e9 02 00 52 d5 0d 27 db e9 02 00 00 00 00 00
0050	00 00 00 00



## 8.3 des\_lib ve aes\_lib Karşılaştırma

The screenshot displays two windows side-by-side. On the left is Wireshark capturing traffic from the loopback adapter. The main pane shows an http session with several TCP and UDP packets. The details pane at the bottom shows the raw hex and ASCII data for the selected packet, which is a TCP segment with sequence number 3747 and destination port 5000. The bottom status bar indicates that "htt" is not a valid protocol or protocol field.

The right window is a web browser showing a chat application titled "Chat Sistemi". It compares the performance of two encryption algorithms: AES and DES. The AES section shows:

- Algoritma: AES
- Şifreli: AKN/SREL9xrwmH+BtjkyxMQauK3btSiCE4kuVScSrA0=
- Çözülmüş: zehra
- RSA Süre: 0.006 ms
- Crypto Süre: 0.167 ms

The DES section shows:

- SECURE (DES) → Şifreli mesaj gönderildi
- SECURE CHAT
- Algoritma: DES
- Şifreli: fHTbvKDRPBq7LsxC7mwCw==
- Çözülmüş: zehra
- RSA Süre: 0.006 ms
- Crypto Süre: 0.168 ms

Below the comparison results is a message input field with a dropdown menu for selecting the encryption algorithm (Modern, Klasik, Caesar) and parameters (Anahat / Shift / Parametre), and a "Gönder" (Send) button.