**ABSTRACT**

The security and integrity of medical data are critical in healthcare systems. This project presents a secure patient data management system using AES Encryption and SHA-256 for digital signature generation, integrated into a Python Flask web application. The system facilitates secure data sharing and access for radiologists, doctors, and patients, ensuring robust encryption and data integrity verification through advanced cryptographic techniques.

The application includes three roles: Radiologist, Doctor, and Patient. Radiologists can log in to input patient details such as name, age, gender, phone number, address, and disease type, along with uploading medical images like MRI, CT scans, or X-rays. Upon submission, the system encrypts the data using AES encryption with keys generated by a custom function that produces random binary numbers to simulate quantum-like key generation. The system also generates a digital signature for integrity validation using SHA-256. The encrypted data, along with the signature, is securely stored in a MySQL database.

Doctors and patients receive unique keys for accessing and decrypting the data. When viewing patient records, the application verifies the digital signature to ensure the data's authenticity and checks for tampering. If the signature verification fails, an alert is displayed indicating compromised data. This ensures that only authorized users can access genuine, unmodified medical records.

Although the project does not implement actual Quantum Cryptography, it demonstrates a practical approach to secure communication in healthcare by using AES encryption and SHA-256, coupled with a simulated quantum-like key generation method. This addresses data security, privacy, and trust issues, providing a robust framework for managing sensitive medical information.

### ****INTRODUCTION****

The security and integrity of medical data are critical in healthcare systems. This project presents a secure patient data management system using AES Encryption and SHA-256 for digital signature generation, integrated into a Python Flask web application. The system facilitates secure data sharing and access for radiologists, doctors, and patients, ensuring robust encryption and data integrity verification through advanced cryptographic techniques.

The application includes three roles: Radiologist, Doctor, and Patient. Radiologists can log in to input patient details such as name, age, gender, phone number, address, and disease type, along with uploading medical images like MRI, CT scans, or X-rays. Upon submission, the system encrypts the data using AES encryption with keys generated by a custom function that produces random binary numbers to simulate quantum-like key generation. The system also generates a digital signature for integrity validation using SHA-256. The encrypted data, along with the signature, is securely stored in a MySQL database.

Doctors and patients receive unique keys for accessing and decrypting the data. When viewing patient records, the application verifies the digital signature to ensure the data's authenticity and checks for tampering. If the signature verification fails, an alert is displayed indicating compromised data. This ensures that only authorized users can access genuine, unmodified medical records.

Although the project does not implement actual Quantum Cryptography, it demonstrates a practical approach to secure communication in healthcare by using AES encryption and SHA-256, coupled with a simulated quantum-like key generation method. This addresses data security, privacy, and trust issues, providing a robust framework for managing sensitive medical information.

### ****PROBLEM STATEMENT****

In the healthcare sector, managing sensitive patient data securely while ensuring its authenticity and integrity is a significant challenge. Traditional methods of data encryption and sharing are vulnerable to breaches, unauthorized access, and tampering, putting patients' privacy and trust at risk. Additionally, the increasing use of digital medical records and imaging demands a system that can handle secure data storage and controlled access for authorized personnel only. To address these issues, there is a need for a robust and efficient system that integrates modern encryption techniques and digital signature validation to safeguard sensitive medical information against threats.

### ****OBJECTIVES****

1. To develop a secure patient data management system using AES encryption for data protection and SHA-256 for digital signature generation.
2. To ensure secure storage and controlled access to sensitive medical information for authorized users only, including radiologists, doctors, and patients.
3. To validate the authenticity and integrity of medical data through digital signature verification, alerting users in case of tampering.
4. To simulate quantum-like key generation using a custom random binary number generator to enhance the encryption process and demonstrate advanced cryptographic techniques.

**PROPOSED SYSTEM**

The proposed system is a secure patient data management application designed to address the challenges of protecting sensitive medical information. The system is developed using Python Flask, AES encryption for data security, and SHA-256 for digital signature generation. A simulated quantum-like key generation method is employed to enhance encryption robustness.

**Key Features of the Proposed System:**

1. **User Roles and Access Control:**  
   The application includes three roles: Radiologist, Doctor, and Patient. Each role has specific permissions:
   * **Radiologists** can input patient details and upload medical images such as MRI, CT scans, or X-rays.
   * **Doctors** can view patient records for diagnosis and treatment planning.
   * **Patients** can securely access their own medical data.
2. **Data Encryption:**  
   Patient data, including personal details and medical images, is encrypted using AES encryption with unique keys generated by a custom random binary number generator. This ensures that the data remains secure during storage and transmission.
3. **Digital Signature for Data Integrity:**  
   SHA-256 is used to generate a digital signature for each record, ensuring the integrity and authenticity of the data. Any modification to the stored data will result in signature verification failure.
4. **Secure Data Storage:**  
   The encrypted patient data and corresponding digital signatures are securely stored in a MySQL database, ensuring data privacy and preventing unauthorized access.
5. **Data Decryption and Verification:**  
   Doctors and patients are provided unique keys to decrypt the encrypted data. Before decryption, the system verifies the digital signature to confirm data integrity. If tampering is detected, the system alerts the user and denies access to the compromised data.
6. **Simulated Quantum-like Key Generation:**  
   Although not employing actual quantum cryptography, the system uses a custom function to generate random binary numbers, simulating a quantum-like key generation mechanism for enhanced encryption security.

**Benefits of the Proposed System:**

* Ensures confidentiality, integrity, and authenticity of sensitive medical data.
* Provides a secure platform for sharing medical records among authorized personnel.
* Enhances patient trust by protecting their data from unauthorized access and tampering.
* Demonstrates the practical application of advanced cryptographic techniques in healthcare systems.

**SYSTEM REQUIREMENTS**

SOFTWARE REQUIREMENTS

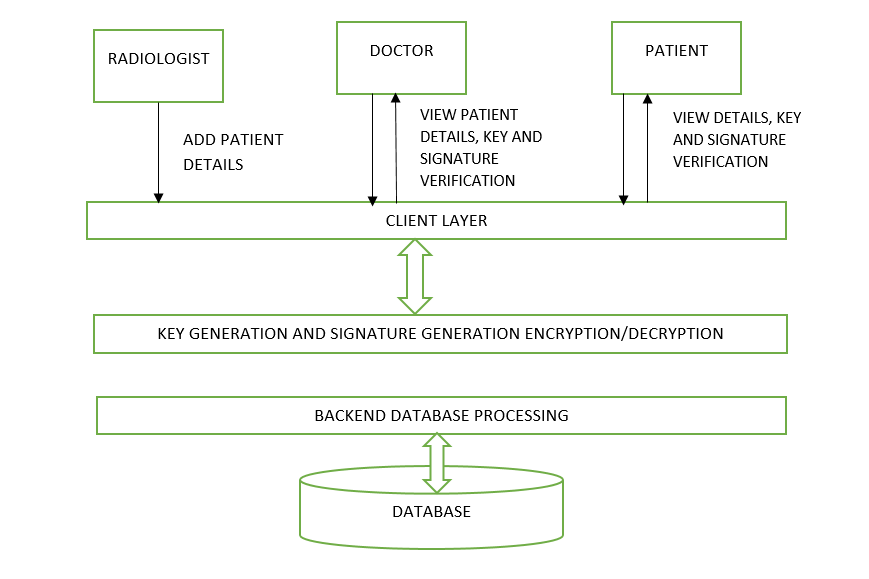
|  |  |
| --- | --- |
| OPERATING SYSTEM | WINDOWS 10+ |
| LANGUAGE | PYTHON |
| IDE | VS CODE |
| FRONT END | HTML,CSS |
| FRAMEWORK | FLASK |
| DATABASE | MYSQL |

HARDWARE REQUIREMENTS

|  |  |
| --- | --- |
| RAM | 4GB |
| HDD | 20GB |
| PROCESSOR | I5 |

**DESIGN**

ARCHITECTURE

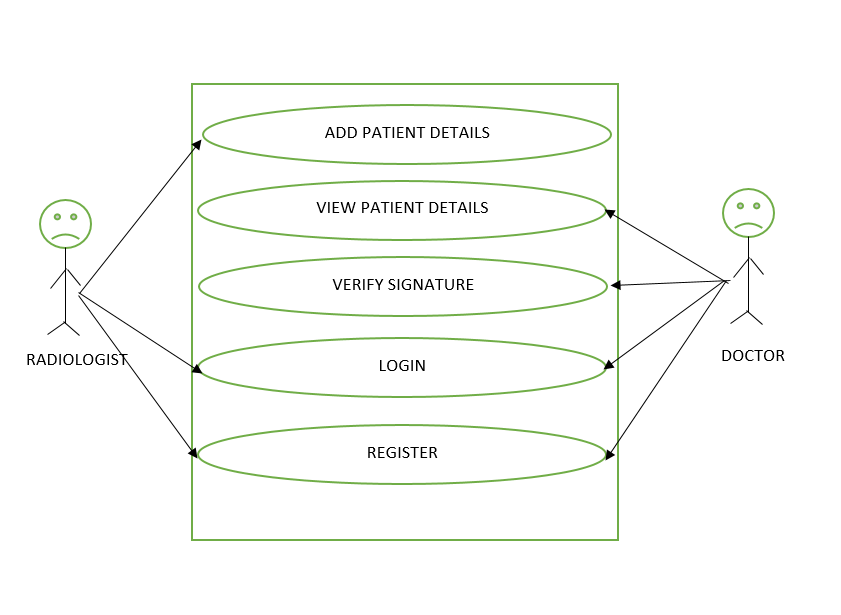


The architecture diagram depicts the flow and interactions in the secure patient data management system. It is structured into multiple layers:

1. **Client Layer:**  
   This layer serves as the interface for three user roles: Radiologists, Doctors, and Patients. Radiologists can add patient details and upload medical images, while Doctors and Patients can view encrypted patient records, access unique decryption keys, and verify data authenticity via digital signature validation.
2. **Key Generation and Encryption/Decryption Layer:**  
   This layer handles the core cryptographic functionalities. It generates keys using a custom random binary number function, encrypts sensitive patient data using AES, and creates digital signatures using SHA-256 for integrity verification. It also facilitates data decryption and signature validation.
3. **Backend Database Processing Layer:**  
   This layer interacts with the database to securely store and retrieve encrypted patient data and digital signatures. It ensures that data remains protected from unauthorized access while enabling efficient query handling.

This layered architecture ensures secure, streamlined data handling while maintaining data integrity and controlled access.

**USE CASE DIAGRAM**



The use case diagram illustrates the key functionalities and interactions between the system and its users: Radiologist and Doctor.

1. **Radiologist:**
   * Can register and log in to the system.
   * Has the capability to add patient details, including medical information and images.
   * Can view patient details and verify the integrity of the data using the digital signature.
2. **Doctor:**
   * Can register and log in to access the system.
   * Can view patient details provided by the radiologist.
   * Verifies the data's authenticity through signature validation to ensure its integrity.

**FUNCTIONAL REQUIREMENTS**

1. **User Authentication and Authorization**:
   * Radiologists, Doctors, and Patients must register and log in to access the system.
   * Each user role has specific access privileges.
2. **Patient Data Management**:
   * Radiologists can input patient details, including name, age, gender, phone number, address, disease type, and upload medical images (e.g., MRI, CT scans, X-rays).
3. **Data Encryption and Decryption**:
   * Patient data is encrypted using AES before being stored in the database.
   * Authorized users (Doctors and Patients) can decrypt data using unique keys.
4. **Digital Signature Generation and Verification**:
   * A digital signature is generated using SHA-256 for each record to ensure data integrity.
   * The system verifies the signature before allowing users to access the data. If the verification fails, a notification about compromised data is displayed.
5. **Secure Database Management**:
   * Encrypted patient data and corresponding digital signatures are stored securely in a MySQL database.
6. **Role-Based Access**:
   * Radiologists can add and view patient data.
   * Doctors and Patients can view patient details and verify the data's authenticity.
7. **Alerts for Compromised Data**:
   * If data integrity verification fails, an alert is generated to notify users of potential tampering.

**NON-FUNCTIONAL REQUIREMENTS**

1. **Security**:
   * AES encryption ensures data confidentiality.
   * SHA-256 digital signature prevents unauthorized modifications and ensures data authenticity.
2. **Performance**:
   * The system should handle encryption, decryption, and signature verification processes efficiently without noticeable delays.
3. **Scalability**:
   * The system must support an increasing number of users and records without performance degradation.
4. **Reliability**:
   * The application must ensure high reliability in encryption, decryption, and signature verification to maintain trust.
5. **Usability**:
   * The interface should be user-friendly, making it easy for Radiologists, Doctors, and Patients to navigate and perform their respective tasks.
6. **Availability**:
   * The system must be accessible 24/7 to ensure seamless operation for users.
7. **Data Integrity**:
   * The system must guarantee the integrity of patient data through robust cryptographic techniques.
8. **Compliance**:
   * The system should comply with healthcare data regulations like HIPAA (if applicable) to ensure data privacy and security.
9. **Portability**:
   * The application should be deployable across various platforms without significant changes.
10. **Maintainability**:
    * The system must be easy to update and maintain for incorporating future enhancements or addressing bugs.

**TOOLS AND TECHNOLOGIES USED**

**ABOUT PYTHON**

Python offers concise and readable code. While complex algorithms and versatile workflows stand behind machine learning and AI, Python’s simplicity allows developers to write reliable systems. Developers get to put all their effort into solving an ML problem instead of focusing on the technical nuances of the language.

Additionally, Python is appealing to many developers as it’s easy to learn. Python code is understandable by humans, which makes it easier to build models for machine learning.

Many programmers say that Python is more intuitive than other programming languages. Others point out the many frameworks, libraries, and extensions that simplify the implementation of different functionalities. It’s generally accepted that Python is suitable for collaborative implementation when multiple developers are involved. Since Python is a general-purpose language, it can do a set of complex machine learning tasks and enable you to build prototypes quickly that allow you to test your product for machine learning purposes.

**Platform independence**

Platform independence refers to a programming language or framework allowing developers to implement things on one machine and use them on another machine without any (or with only minimal) changes. One key to Python’s popularity is that it’s a platform independent language. Python is supported by many platforms including Linux, Windows, and macOS. Python code can be used to create standalone executable programs for most common operating systems, which means that Python software can be easily distributed and used on those operating systems without a Python interpreter.

What’s more, developers usually use services such as Google or Amazon for their computing needs. However, you can often find companies and data scientists who use their own machines with powerful Graphics Processing Units (GPUs) to train their ML models. And the fact that Python is platform independent makes this training a lot cheaper and easier.

**Great community and popularity**

In the Developer Survey 2018 by Stack Overflow, Python was among the top 10 most popular programming languages, which ultimately means that you can find and hire a development company with the necessary skill set to build your AI-based project.

If you look closely at the image below, you’ll see that Python is the language that people Google more than any other.

**PYTHON INSTALLATION**

Python is a widely used high-level programming language. To write and execute code in python, we first need to install Python on our system.

Installing Python on Windows takes a series of few easy steps.

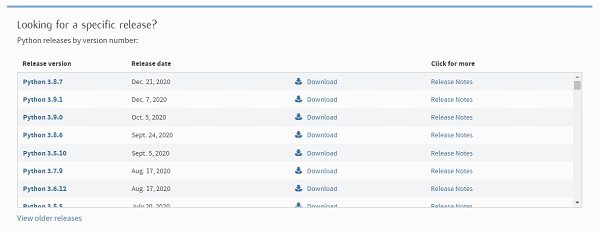
Step 1 − Select Version of Python to Install

Python has various versions available with differences between the syntax and working of different versions of the language. We need to choose the version which we want to use or need. There are different versions of Python 2 and Python 3 available.

Step 2 − Download Python Executable Installer

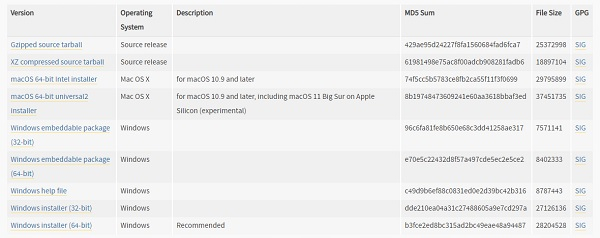
On the web browser, in the official site of python ([www.python.org](https://www.tutorialspoint.com/www.python.org)), move to the Download for Windows section.

All the available versions of Python will be listed. Select the version required by you and click on Download. Let suppose, we chose the Python 3.9.1 version.



On clicking download, various available executable installers shall be visible with different operating system specifications. Choose the installer which suits your system operating system and download the instlaller. Let suppose, we select the Windows installer(64 bits).

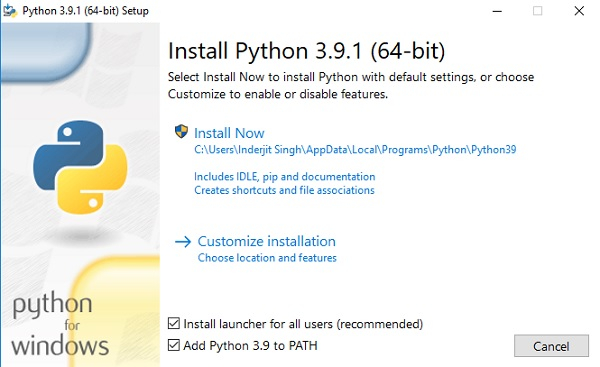
The download size is less than 30MB.



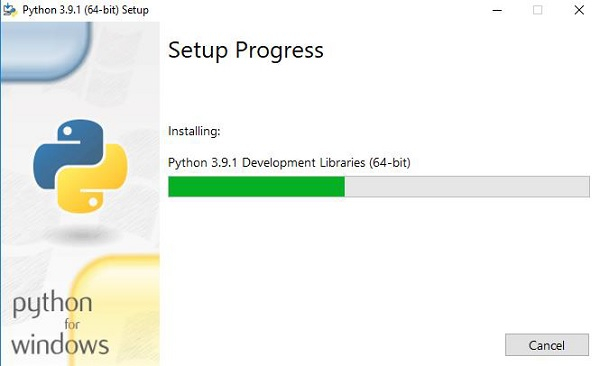
Step 3 − Run Executable Installer

We downloaded the Python 3.9.1 Windows 64 bit installer.

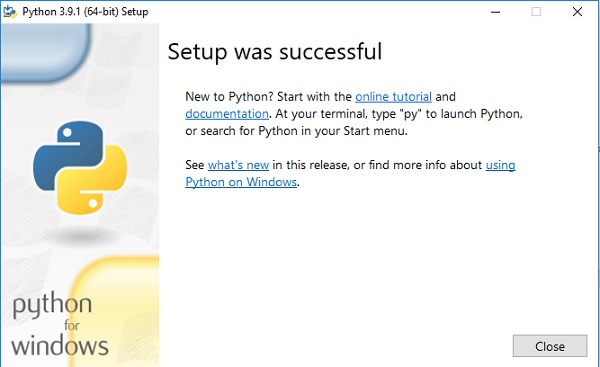
Run the installer. Make sure to select both the checkboxes at the bottom and then click Install New.



On clicking the Install Now, The installation process starts.



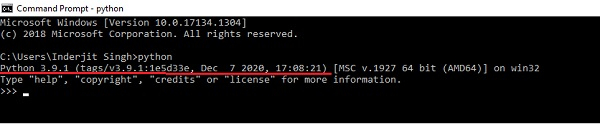
The installation process will take few minutes to complete and once the installation is successful, the following screen is displayed.



Step 4 − Verify Python is installed on Windows

To ensure if Python is succesfully installed on your system. Follow the given steps −

* Open the command prompt.
* Type ‘python’ and press enter.
* The version of the python which you have installed will be displayed if the python is successfully installed on your windows.

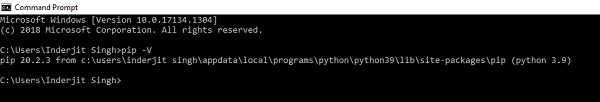


Step 5 − Verify Pip was installed

Pip is a powerful package management system for Python software packages. Thus, make sure that you have it installed.

To verify if pip was installed, follow the given steps −

* Open the command prompt.
* Enter pip –V to check if pip was installed.
* The following output appears if pip is installed successfully.



We have successfully installed python and pip on our Windows system.

To install any libraries

Go to command prompt

Type

Pip install libraryname

**FLASK**

**What is Flask Python**

Flask is a web framework, it’s a Python module that lets you develop web applications easily. It’s has a small and easy-to-extend core: it’s a microframework that doesn’t include an ORM (Object Relational Manager) or such features.

It does have many cool features like url routing, template engine. It is a WSGI web app framework.

**What is a Web Framework?**

A Web Application Framework or a simply a Web Framework represents a collection of libraries and modules that enable web application developers to write applications without worrying about low-level details such as protocol, thread management, and so on.

**What is Flask?**

Flask is a web application framework written in Python. It was developed by Armin Ronacher, who led a team of international Python enthusiasts called Poocco. Flask is based on the Werkzeg WSGI toolkit and the Jinja2 template engine.Both are Pocco projects.

**WSGI**

The Web Server Gateway Interface (Web Server Gateway Interface, WSGI) has been used as a standard for Python web application development. WSGI is the specification of a common interface between web servers and web applications.

**Werkzeug**

Werkzeug is a WSGI toolkit that implements requests, response objects, and utility functions. This enables a web frame to be built on it. The Flask framework uses Werkzeg as one of its bases.

**jinja2**

jinja2 is a popular template engine for Python.A web template system combines a template with a specific data source to render a dynamic web page.

This allows you to pass Python variables into HTML templates like this:

|  |
| --- |
| <html>  <head>  <title>{{ title }}</title>  </head>  <body>  <h1>Hello {{ username }}</h1>  </body> </html> |

**Microframework**

Flask is often referred to as a microframework. It is designed to keep the core of the application simple and scalable.

Instead of an abstraction layer for database support, Flask supports extensions to add such capabilities to the application.

**Why is Flask a good web framework choice?**

Unlike the Django framework, Flask is very Pythonic. It’s easy to get started with Flask, because it doesn’t have a huge learning curve.

On top of that it’s very explicit, which increases readability. To create the “Hello World” app, you only need a few lines of code.

This is a boilerplate code example.

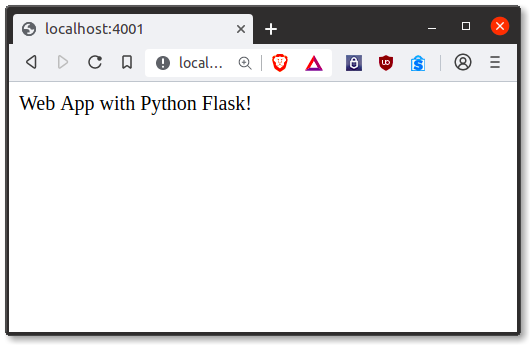
|  |
| --- |
| from flask import Flask app = Flask(\_\_name\_\_)  @app.route('/') def hello\_world():  return 'Hello World!'  if \_\_name\_\_ == '\_\_main\_\_':  app.run() |

If you want to develop on your local computer, you can do so easily. Save this program as server.py and run it with python server.py.

|  |
| --- |
| $ python server.py  \* Serving Flask app "hello"  \* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit) |

It then starts a web server which is available only on your computer. In a web browser open localhost on port 5000 (the url) and you’ll see “Hello World” show up.  
To host and develop online, you can use [PythonAnywhere](https://www.pythonanywhere.com/?affiliate_id=00535ced" \t "_blank)

Some example output:



It’s a microframework, but that doesn’t mean your whole app should be inside one single Python file. You can and should use many files for larger programs, to handle complexity.

Micro means that the Flask framework is simple but extensible. You may all the decisions: which database to use, do you want an ORM etc, Flask doesn’t decide for you.

Flask is one of the most popular web frameworks, meaning it’s up-to-date and modern. You can easily extend it’s functionality. You can scale it up for complex applications

**HTML**

**Hypertext Markup Language** (**HTML**) is the standard markup language for creating web pages and web applications. With Cascading Style Sheets (CSS) and JavaScript, it forms a triad of cornerstone technologies for the World Wide Web.

Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by *tags*, written using angle brackets. Tags such as <**img** /> and <**input** /> directly introduce content into the page. Other tags such as <**p**> surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page.

HTML can embed programs written in a scripting language such as JavaScript, which affects the behavior and content of web pages. Inclusion of CSS defines the look and layout of content. The World Wide Web Consortium (W3C), maintainer of both the HTML and the CSS standards, has encouraged the use of CSS over explicit presentational HTML since 1997.

**CSS**

**Cascading Style Sheets** (**CSS**) is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file, and reduce complexity and repetition in the structural content.

Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or screen reader), and on Braille-basedtactile devices. CSS also has rules for alternate formatting if the content is accessed on a mobile device.

The name *cascading* comes from the specified priority scheme to determine which style rule applies if more than one rule matches a particular element. This cascading priority scheme is predictable.

The CSS specifications are maintained by the World Wide Web Consortium (W3C). Internet media type (MIME type) text/css is registered for use with CSS by RFC 2318 (March 1998). The W3C operates a free CSS validation service for CSS documents.

In addition to HTML, other markup languages support the use of CSS, including XHTML, plain XML, SVG, and XUL.

**AES ALGORITHM**

The **AES (Advanced Encryption Standard)** algorithm is a symmetric key encryption standard widely used for securing data. It operates on fixed-size blocks of data (128 bits) and supports key sizes of 128, 192, or 256 bits. AES is used in various security protocols like SSL/TLS, VPNs, and disk encryption due to its efficiency and strong security.

### Key Concepts:

* **Symmetric Key**: The same key is used for both encryption and decryption.
* **Block Cipher**: AES encrypts data in blocks of 128 bits (16 bytes) at a time.
* **Rounds**: AES performs multiple rounds of processing (10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys) to enhance security.

### AES Encryption Process:

1. **Initial Round**:
   * AddRoundKey: XOR the plaintext with the round key.
2. **Main Rounds** (repeated for the number of rounds):
   * SubBytes: Substitute bytes in the state using a substitution box (S-Box).
   * ShiftRows: Shift the rows of the state to the left.
   * MixColumns: Mix the columns of the state to make the ciphertext more complex (except in the last round).
   * AddRoundKey: XOR the state with the round key.
3. **Final Round**:
   * SubBytes
   * ShiftRows
   * AddRoundKey (no MixColumns in the final round)

### AES Decryption Process:

The decryption process is similar to encryption but involves reversing the steps. The operations are reversed, and the round keys are applied in reverse order.

### Example of AES Encryption and Decryption in Python:

Here’s an example using the Python library pycryptodome to perform AES encryption and decryption.

#### 1. **Install pycryptodome**:

pip install pycryptodome

#### 2. **Python Code Example**:

from Crypto.Cipher import AES

from Crypto.Random import get\_random\_bytes

from Crypto.Util.Padding import pad, unpad

# Key and IV generation

key = get\_random\_bytes(16) # AES-128 (16 bytes key)

iv = get\_random\_bytes(16) # Initialization vector (16 bytes)

# AES Encryption

def aes\_encrypt(plaintext, key, iv):

cipher = AES.new(key, AES.MODE\_CBC, iv)

ciphertext = cipher.encrypt(pad(plaintext.encode(), AES.block\_size))

return ciphertext

# AES Decryption

def aes\_decrypt(ciphertext, key, iv):

cipher = AES.new(key, AES.MODE\_CBC, iv)

decrypted\_data = unpad(cipher.decrypt(ciphertext), AES.block\_size)

return decrypted\_data.decode()

# Example Usage

plaintext = "This is a secret message"

print(f"Original message: {plaintext}")

# Encrypting the message

ciphertext = aes\_encrypt(plaintext, key, iv)

print(f"Encrypted message: {ciphertext.hex()}")

# Decrypting the message

decrypted\_message = aes\_decrypt(ciphertext, key, iv)

print(f"Decrypted message: {decrypted\_message}")

### Explanation:

* **AES.new(key, AES.MODE\_CBC, iv)**: Creates a new AES cipher object in CBC mode, which requires an initialization vector (IV).
* **pad()**: Ensures the plaintext length is a multiple of the block size (16 bytes).
* **unpad()**: Removes padding after decryption to return the original message.

### Sample Output:

Original message: This is a secret message

Encrypted message: 7c4f7e9c4f0fc9bbad1e4f0436a6c50a

Decrypted message: This is a secret message

### Key Points:

* **CBC (Cipher Block Chaining)** mode is used here, which requires an IV for encryption and decryption.
* **Padding** ensures the plaintext fits into a multiple of 128-bit blocks.

### Security Considerations:

* The key must be kept secret. If someone knows the key, they can easily decrypt the ciphertext.
* IV should be random for each encryption to prevent attacks like replay attacks.

**SHA 256 FOR SIGNATURE GENERATION AND VERIFICATION**

**SHA-256** (Secure Hash Algorithm 256-bit) is a cryptographic hash function that produces a fixed-size (256-bit) hash value from an input of any size. It's part of the SHA-2 family and is widely used for integrity checking, digital signatures, and password hashing. SHA-256 is considered secure because it produces a unique hash for each unique input, and it's computationally infeasible to reverse the process (i.e., find the original input from the hash).

### Signature Generation Using SHA-256:

In the context of digital signatures, **SHA-256** is often used to generate a hash of the data, which is then signed using a private key. The process involves:

1. **Hashing the Data**: First, the data (e.g., a document or message) is passed through the SHA-256 function to produce a hash.
2. **Signing the Hash**: The resulting hash is then signed with a private key using a public key cryptosystem like RSA, DSA, or ECDSA. This signed hash (the digital signature) can be verified by anyone using the corresponding public key.

### Example of SHA-256 Signature Generation (RSA-based):

Let’s see how you can generate a SHA-256 signature using Python and the pycryptodome library. In this example, we will:

* Generate a SHA-256 hash of some data (message).
* Sign the hash using an RSA private key.
* Verify the signature using the corresponding RSA public key.

#### 1. **Install the Required Library**:

pip install pycryptodome

#### 2. **Python Code Example**:

from Crypto.PublicKey import RSA

from Crypto.Signature import pkcs1\_15

from Crypto.Hash import SHA256

from Crypto.Random import get\_random\_bytes

# Step 1: Generate RSA Keys (Private and Public)

key = RSA.generate(2048)

private\_key = key

public\_key = key.publickey()

# Step 2: Prepare the message to sign

message = "This is a secret message to sign."

message\_bytes = message.encode()

# Step 3: Generate SHA-256 hash of the message

hash\_object = SHA256.new(message\_bytes)

# Step 4: Sign the hash with the private key

signer = pkcs1\_15.new(private\_key)

signature = signer.sign(hash\_object)

print(f"Original message: {message}")

print(f"SHA-256 Hash of message: {hash\_object.hexdigest()}")

print(f"Generated Signature: {signature.hex()}")

# Step 5: Verifying the Signature (using the public key)

try:

verifier = pkcs1\_15.new(public\_key)

verifier.verify(hash\_object, signature)

print("Signature is valid!")

except (ValueError, TypeError):

print("Signature is invalid!")

### Explanation of Steps:

1. **RSA Key Generation**:
   * A 2048-bit RSA key pair (private and public) is generated using RSA.generate(). The private key is used for signing, and the public key is used for verification.
2. **Message and Hash**:
   * The message to be signed is encoded into bytes and passed to SHA256.new() to generate the hash of the message.
3. **Signature Creation**:
   * The generated SHA-256 hash is signed using the private key with the help of pkcs1\_15.new(private\_key) and the sign() method. This creates the digital signature.
4. **Signature Verification**:
   * To verify the signature, the public key is used along with the verify() method of the RSA signature scheme. If the signature is valid, it prints Signature is valid!, otherwise, it prints Signature is invalid!.

### Sample Output:

Original message: This is a secret message to sign.

SHA-256 Hash of message: 4a656d6d8f1c8d8b295bd2b2d7fcd1cf7d81e38f7ffbb34bb9f93b6a4ea5393f

Generated Signature: 8c4a7d6a251acc7ff574fb0d6c5b9250b0cb2f394a2d92bb3345f3f4bcfd4fcd6b5041b2b0e6b63d7f6f8cf99492d3bada60e29f0809e5c7c8f303876771db02bc1b46ff340b1cdb76d16ac2ba46a51ac3b8cf62e3d51076ffcb94dbcd105a1ab0b9a2e7d727f67e619598e3b420c7f15a8b8cb62de702b5b8009f6b47e90e5b988575d0b4d9f4f3d99ffab7ae9b43b53b6be95dff6cb99ac8907d6a5a74b8710dff94e2f509db503bc1de383edee06202e90ab43cd640b6b020a92c364fddeafffb033ff7b3f5821f14019f07d4f1e464

Signature is valid!

### Explanation of the Output:

* **SHA-256 Hash**: A fixed-size 256-bit (64 hexadecimal characters) hash is generated for the original message.
* **Signature**: A long hexadecimal string representing the signed hash of the message.
* **Signature Verification**: The public key successfully verifies that the signature corresponds to the hash of the original message.

### Key Points:

* **Digital Signature**: In this example, the signature is the result of applying RSA encryption to the SHA-256 hash of the message.
* **Public Key Verification**: The public key can be used to verify that the message has not been tampered with and that the signature matches the message.
* **Security**: The private key used to sign the message must be kept secret, while the public key can be shared for signature verification.

**CONCLUSION**

The Secure Patient Data Management System using AES encryption and SHA-256 for digital signature generation provides a robust solution for safeguarding sensitive medical information within healthcare systems. By integrating advanced cryptographic techniques such as symmetric encryption and digital signature validation, this system ensures the confidentiality, integrity, and authenticity of patient data. The use of AES encryption secures patient details and medical images from unauthorized access, while SHA-256 guarantees the integrity of the data, preventing tampering and ensuring trust between healthcare professionals and patients.

Although the project simulates quantum-like key generation, it effectively demonstrates a secure approach to managing medical data in a web application environment. By implementing role-based access control, the system ensures that only authorized users, such as radiologists, doctors, and patients, can access and decrypt the data, maintaining strict privacy standards. The inclusion of digital signature verification further strengthens data integrity by alerting users in case of any compromise.

This project addresses critical issues related to data security, privacy, and trust, providing a practical and scalable framework for secure medical data management in healthcare systems. It highlights the potential for future integration of more advanced cryptographic methods, such as quantum cryptography, to further enhance the security of healthcare data in the evolving technological landscape.

**FUTURE SCOPE**

The Secure Patient Data Management System, while already robust in its use of AES encryption and SHA-256 for digital signature generation, has several avenues for future development and enhancement. These improvements can address emerging security challenges, enhance user experience, and incorporate advanced technologies to ensure the system remains adaptable and secure in the evolving healthcare landscape.

1. **Blockchain for Immutable Data Records**: Implementing blockchain technology could provide a decentralized and tamper-proof ledger for medical records. By storing digital signatures and encrypted patient data in a blockchain, healthcare providers can ensure that the data remains immutable and can be audited without fear of tampering or unauthorized modification.
2. **Biometric Authentication**: Adding biometric authentication, such as fingerprint or facial recognition, can further enhance user authentication and access control. This would prevent unauthorized access by ensuring that only the legitimate person (radiologist, doctor, or patient) can view or modify sensitive medical records.
3. **AI/ML for Anomaly Detection**: Artificial intelligence (AI) and machine learning (ML) algorithms could be integrated into the system to monitor and detect any unusual activity or anomalies in data access patterns. This could help identify potential security threats, such as unauthorized access attempts or data tampering, and alert administrators in real-time.
4. **Multi-Factor Authentication (MFA)**: Implementing multi-factor authentication (MFA) for all user roles would add an extra layer of security. In addition to passwords, MFA could require secondary factors like OTPs, security tokens, or biometric scans, ensuring only authorized users can access patient data.
5. **Real-Time Data Sharing with Secure Channels**: Enhancing the system to support real-time medical data sharing between radiologists, doctors, and patients using secure channels would improve collaboration. The integration of secure messaging protocols or even video consultations could ensure that sensitive data is transmitted safely.

**REFERENCES**

### Books:

1. **"Cryptography and Network Security: Principles and Practice" by William Stallings**
   * This book provides a comprehensive overview of cryptography, including encryption algorithms like AES and hash functions like SHA-256. It's a great resource for understanding the theory and application of cryptographic techniques in various domains, including healthcare.
2. **"Applied Cryptography: Protocols, Algorithms, and Source Code in C" by Bruce Schneier**
   * A classic in the field of cryptography, this book covers a wide range of cryptographic techniques, including AES encryption, and provides practical examples of implementing secure systems.

### Research Papers & Articles:

1. **"A Survey on Blockchain Technology and Its Applications in Healthcare"**
   * This paper discusses how blockchain can be integrated into healthcare for data security, immutability, and privacy, providing insights into how blockchain could complement cryptographic methods in healthcare systems.
   * Source: MDPI - Healthcare Journal
2. **"The Role of Cryptography in Electronic Health Record Security" by S. S. R. K. Reddy, M. S. Rajasekaran, and M. P. R. Reddy**
   * This paper explores the role of cryptographic methods such as AES encryption and digital signatures in ensuring the security and privacy of electronic health records (EHR).
   * Source: Journal of Medical Systems
3. **"Quantum Cryptography: A Practical Perspective" by C. F. Roos, P. M. K. Mathews, and S. K. Sinha**
   * Discusses the principles and future potential of quantum cryptography, including Quantum Key Distribution (QKD), which could revolutionize data security in sensitive areas like healthcare.
   * Source: Springer

### Websites & Online Resources:

1. **National Institute of Standards and Technology (NIST) - AES Encryption Standard**
   * The official documentation of the AES encryption standard provides detailed insights into the algorithm's security and implementation.
   * Website: <https://csrc.nist.gov/publications/detail/fips/197/final>
2. **SHA-256 Information from the Cryptographic Module Validation Program (CMVP)**
   * Official documentation on SHA-256 and its use in secure systems, detailing the standards and validation processes for cryptographic algorithms.
   * Website: <https://csrc.nist.gov/projects/cryptographic-module-validation-program>
3. **"Blockchain Technology in Healthcare: A Comprehensive Review and Directions for Future Research" by Alaa Khamis, Ahmed S. F. H. Ali, and Mohamed Ali Hamouda**
   * This review discusses the application of blockchain in healthcare, focusing on its integration with cryptographic methods like AES for enhanced security and privacy.
   * Source: IEEE Xplore
   * Website: <https://ieeexplore.ieee.org/document/9044557>

### Journals and Conferences:

1. **IEEE Transactions on Information Forensics and Security**
   * This journal publishes cutting-edge research in the fields of cryptography, data security, and forensics, including topics related to healthcare and medical data security.
   * Website: <https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=10206>
2. **Journal of Medical Internet Research (JMIR)**
   * A journal that frequently publishes research related to healthcare systems, data security, and cryptography in digital health applications.
   * Website: <https://www.jmir.org/>

These references will provide a solid foundation for understanding the cryptographic principles behind AES encryption, SHA-256 digital signatures, and their applications in healthcare systems. Additionally, you can find valuable insights into integrating advanced technologies like blockchain and quantum cryptography for enhanced data security and privacy in the healthcare domain.