Improving data security in Iot healthcare offerings the use of fog computing

Podilapu Chakradharnaidu, Ippili Sri Sravya, Manoj Kumar Kar, Bobbili Karthik, Lakkoju Eswara Sai, Mandala Varun Kumar

Department of Computer Science and Engineering Aditya Institute of Technology and Management Srikakulam, India

***Abstract*— By connecting physical items, the Internet of Things (IoT) enables data transmission and reception over the internet without human interaction. Massive volumes of real-time data are produced in the medical field leveraging Iot devices such as medical sensors and wearable monitors, often stored in the cloud. Managing such data presents challenges like latency and security, especially for real-time healthcare applications.**

Fog computing handles these issues by shifting processing and storage towards the edge of the network near data sources. Fog nodes process, encrypt, and store data locally, reducing reliance on cloud services. This approach minimizes latency and ensures real-time computation. Security is strengthened as data encrypted with the Advanced Encryption Standard (AES) at the fog node is further protected with an access token mechanism, allowing only authorized entities to encrypt or access files directly from storage drives. This safeguards sensitive information and ensures compliance with regulations like HIPAA and GDPR.

By handling data at a local level, fog computing reduces communication errors and improves consistency and accuracy. It offers a reliable framework for healthcare IoT data processing, guaranteeing better patient care, improved decision-making, and enhanced data privacy.

Keywords—Internet of Things, fog computing, Access Token, Advanced Encryption Standard (AES) Algorithm

I. INTRODUCTION

By linking commonplace items to the internet for data gathering and analysis, the explosive growth of IoT (Internet of Things) devices has revolutionized how industries function. However, real-time handling of data is a major barrier for typical IoT-cloud infrastructures, especially in vital industries like healthcare. Timely data evaluation and decision-making are vital for applications like patient monitoring and emergency response; yet, cloud-based models frequently have significant latency, bandwidth restrictions, and security flaws. When quick decisions are needed, these difficulties may reduce the efficacy of healthcare apps.

One revolutionary method for getting over these restrictions is fog computing. Fog computing diminishes the amount of data that needs to be transfer to the cloud for analysis by enabling localized data processing nearer the source (i.e., edge devices as shown in Fig. 1). This significantly lowers latency while also enhancing real-time performance. Fog computing improves the responsiveness and efficiency of systems by bridging the gap between cloud and edge devices. Fog computing's decentralized architecture also improves security, scalability, and quality of service (QoS), guaranteeing that vital data is handled quickly and safely without taxing cloud servers. The ability of fog computing to react quickly to changes in patient circumstances is one of its biggest benefits in the healthcare industry. For example, vital signs can be locally collected and analyzed by wearable health devices, which can promptly alert medical specialists to any irregularities found. This capacity improves patient care and results by enabling healthcare systems to respond swiftly. Fog computing does, however, still have issues with deployment costs, complexity, and the requirement for continuous management, despite its benefits. These challenges are still being addressed by technological developments, which makes fog computing a more viable option for next-generation Internet of Things applications (Fig. 2).

Alongside these developments in fog computing and the Internet of Things, network security is still a major worry.

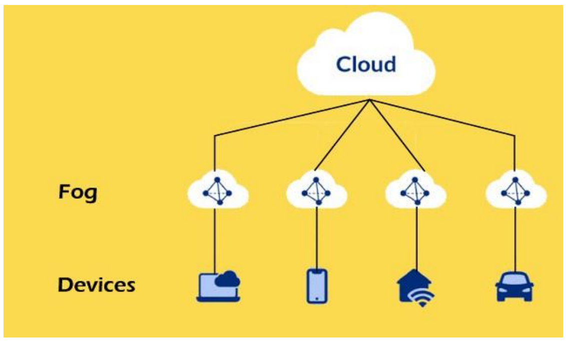


Fig. 1

Ensuring the security of the data being transferred is crucial as more gadgets join the internet. A variety of technologies, procedures, and regulations are used in network security to shield data from disturbances, illegal access, and cyberattacks. Firewalls, intrusion detection systems (IDS), encryption, and access control systems are important parts. Together, these capabilities protect sensitive data's availability, confidentiality, and integrity across networks.

Advanced security solutions are required as cyber threats get more complex. Zero-trust security models and AI-based threat detection are becoming more and more popular in network security. These solutions guarantee that no entity, either inside or outside the network, is trusted by default and aid in the more efficient identification and response to threats. Because of its effectiveness and robust cryptographic features, AES (Advanced Encryption Standard) has emerged as the preferred method for data security. AES offers a high degree of security and works well for a number of applications, such as disc encryption, secure data transmission, and secure communication. It is perfect for securing sensitive data because it is quicker and more dependable than more traditional algorithms like DES and 3DES.

Support for several key lengths (128, 192, and 256 bits) is one of AES's finest qualities; it allows for flexibility in security levels based on the specific needs of the application. It is appropriate for settings with high data throughput, such cloud-based apps and Internet of Things systems, due to its speed and resilience. AES is the recommended option for encrypting huge amounts of data since it functions effectively in real-time, unlike asymmetric encryption algorithms like RSA, which are slower and more computationally costly. Its reputation as the gold standard for encryption has been cemented by its ability to protect sensitive data and by its extensive use in sectors like government, healthcare, and finance.

Additionally, by guaranteeing that only verified individuals or apps are able to encrypt or access critical data, the combination of access tokens and AES encryption offers an extra degree of protection. A digital identification that is issued upon user authentication and authorizes specific operations, like file encryption, is called an access token. Sensitive files can only be encrypted by authorized users or systems when AES encryption and access tokens are combined. By providing defence against unwanted access and any data breaches, this approach guarantees that the data will always be private and safe.

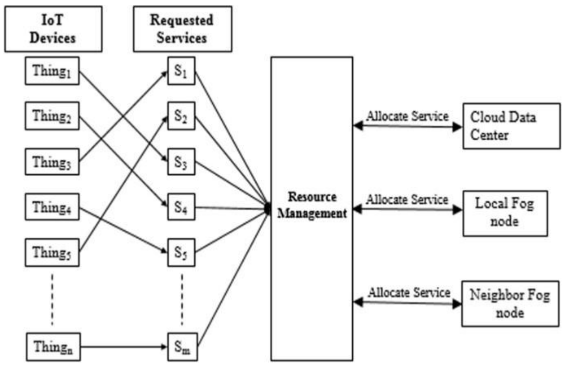


Fig. 2

Organizations can improve their data protection policies by combining access tokens with AES encryption, particularly in settings like healthcare IoT applications that demand strong security and real-time data processing. This combination helps to create a more secure, dependable, and responsive infrastructure for contemporary digital ecosystems by guaranteeing that private data is kept safe and only available to those who are authorized.

II. LITERATURE SURVEY

Because cloud computing provides centralized storage with strong encryption, access management, and compliance with laws like GDPR and HIPAA, it is essential for improving data security for Internet of Things applications. By using firewalls, multi-factor authentication, and frequent upgrades, it protects IoT-generated data from cyber threats and unauthorized access, allowing for scalable administration [1][4]. However, typical cloud infrastructures are frequently challenged by the latency and real-time processing requirements of IoT devices, specifically for time-dependent applications [10].

By processing and encrypting IoT data closer to the network's edge and reducing dependency on remote cloud servers, fog computing overcomes these issues [5][9]. By minimizing data exposure during transmission, this decentralized method improves data privacy, guarantees faster reaction times, and lowers latency. For IoT systems, cloud and fog computing work together to provide a safe and effective framework that facilitates scalable data management and real-time decision- making in vital applications such as industrial automation, smart cities, and healthcare [2].

For data security and privacy, several kinds of cryptographic techniques are employed. Asymmetric and symmetric encryption are the two main categories of encryption. Asymmetric encryption process invloves encrypting with a public key and decrypting with a private key. The private key, which is used for both encryption and decryption, is the only key used in symmetric encryption. Symmetric encryption [6] requires less processing resources because it requires same secure key for encryption and decryption, making it incredibly secure, straightforward, and 1000 times faster. The Advance Encryption Standard (AES), a symmetric encryption method, is the main topic of this essay.

AES is not the only older algorithm; 3-DES and DES are also older. Data Encryption Standard [7] (DES) is a rather poor encryption technique that creates a 64-bit ciphertext using a 56-bit key size and 16-bit block size plaintext [16]. It can be readily cracked in a day by applying the brute force method. Therefore, it cannot be regarded as a safe method of encrypting private medical information. Similar to DES, Triple Data Encryption Standard (3DES) has a 16-bit block size with a key size of 56, 112, or 168 bits. Although it is superior to DES, it is still readily cracked.

Despite being popular and safe for key exchanges, the RSA method has shortcomings [14][15]. Because of its high processing cost, it is ineffective for encrypting huge amounts of data. Compared to symmetric algorithms like AES, RSA requires bigger key sizes [17], which raises the processing and storage requirements. It is also susceptible to specific attacks, like side-channel and selected ciphertext assaults, as well as potential threats from quantum computing in the future. RSA is less appropriate for high-performance or bulk encryption workloads because of these drawbacks.

The AES algorithm is introduced to address the aforementioned problems. The US government has embraced the symmetric block cypher standard AES [12] [14]. It is regarded as a more sophisticated, safe, and efficient standard.

III. METHODOLOGY

A. *Existing security system in fog*

Because IoT devices lack memory and processing power, data from these devices is directly stored on the cloud in the present IoT-cloud architecture. But this brings up issues with data security, including issues like denial-of-service (DoS) assaults, data loss, and insecure access points. For further protection, fog computing employs a decoy system that gives attackers trying to gain unauthorized access fictitious data. Users are asked security questions during login, and if they answer them incorrectly, erroneous files are created. Although somewhat successful, attackers may correctly guess replies, rendering the decoy system inadequate for strong data security.

B. *Proposed System*

Implementing an AES algorithm in the fog node is the solution to the aforementioned issue. The AES standard encrypts and decrypts data using a symmetric block cypher method. This method's implementation in the middle layer offers two levels of data protection: one at the fog layer and another at the cloud's conventional security system. AES uses a 16-byte plaintext block with a key lengths of 128, 192, or

256 bits. When they are combined, the result will be a ciphertext. AES operates on a major order matrix with 4x4 columns. Depending on the key size, it goes through several rounds of transformation to change the plaintext to ciphertext. 10 cycles for 128-bits (16 bytes). 12 cycles for 192-bits (24 bytes). Regarding 256-bit.

The AES (Advanced Encryption Standard) algorithm is a symmetric block cipher designed for encrypting and decrypting data securely (Fig. 4).

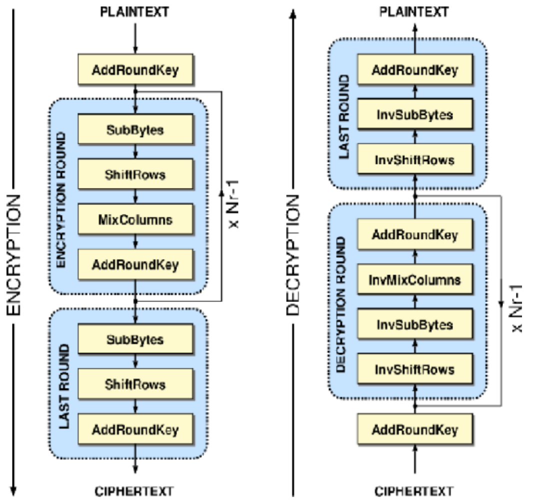


Fig. 3

It works on predefined block sizes (128 bits) and supports key sizes of 128, 192, or 256 bits. AES operates through a series of processing rounds, influenced by the key size.

C. *AES Encryption Algorithm Overview*

The AES encryption mechanism consists of the following main steps:

i) Key Expansion: The original encryption key is expanded into an array of key schedule words. These words will be used in subsequent rounds of the encryption process.

1) *Initial Round (Pre-Round):*

AddRoundKey: The state (data) is combined using the XORed operation with the round key generated through the key expansion process.

2) *Main Rounds (Performed for 9, 11, or 13 rounds, depending on key size):*

In each round, the following steps are applied to the data block:

(i) SubBytes: Each byte of the state is substituted using an S- box (substitution table), which provides non-constant.

(ii) ShiftRows: The rows of the state are rotated cyclically. The first row remains unchanged, while the second, third and fourth rows are shifted by 1, 2 and 3bytes respectively.

(iii) MixColumns: Each column of the state is mixed to provide diffusion (combining the bytes from different rows).

(iv) AddRoundKey: The state (data) is combined using the XORed operation with the round key generated through the key expansion process.

3) *Final Round (Last Round):*

The final round does not include the MixColumns step. It consists of the following operations:

(i) SubBytes

(ii) ShiftRows

(iii) AddRoundKey

4) *Output:*

Once the final round is complete, the outcome is the encrypted ciphertext.

D. *AES Decryption Algorithm*

Decryption is performed by reversing the steps of encryption. The steps are essentially the same but performed in the reverse order and using the inverse operations:

(i) Inverse SubBytes (using the inverse of the S-box)

(ii) Inverse ShiftRows (rows are shifted in the opposite direction)

(iii) Inverse MixColumns (inverse transformation of the MixColumns step)

(iv) AddRoundKey (same as encryption, but using the round keys in reverse order)

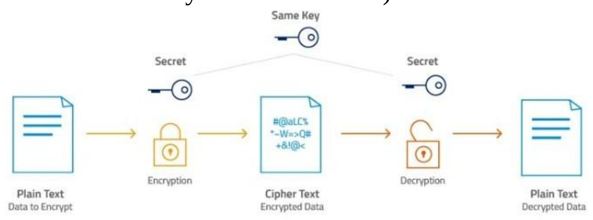


Fig. 4

Key Points

(i) Block Size: AES works with 128-bit blocks.

(ii) Key Sizes: AES supports three key sizes: 128, 192, and 256 bits.

(iii) Rounds: The number of rounds is determined by the size of the key.

- 10 rounds for 128-bit key
- 12 rounds for 192-bit key
- 14 rounds for 256-bit key

This algorithm is extensively used to ensure data encryption for various purposess, like communication, files, and other confidential data.

E. *Pseudocode*

```
void substituteBytes(byte[][] block) {
for (int i = 0; i < 4; i++) {
for (int j = 0; j < bloc[0].length; j++) {
block[i][j] = sBox[block[i][j]];
}
}
}

void shiftRows(byte[][] block) {
byt[] tempRow = new byte[4];
for (int rowIndex = 1; rowIndex
< 4; rowIndex++) {
for (int colIndex = 0; colIndex
< bloc[0].length; colIndex++) {
tempRow[colIndex]
= block[rowIndex][(colIndex
+ rowIndex) % bloc[0].length];
}
for (int colIndex = 0; colIndex
< bloc[0].length; colIndex++) {
block[rowIndex][colIndex]
= $tempRow[colIndex]$;
}
}
}
        $void\ mixColumns(byte[][]\ block)$ {
        $byt[]\ columnResult\ =\ new\ byte[4]$;
        $for\ (int\ col\ =\ 0;\ col\ <\ 4;\ col+$
        $+)$ {
      $columnResul[0]$
    $=\ (byte)\ ((0x02\ *\ bloc[0][col])\ ^\wedge\ (0x03$
    $*\ bloc[1][col])\ ^\wedge\ block[2][col]\ ^\wedge$
    $block[3][col])$;
      $columnResul[1]$
              $=\ (byte)\ (bloc[0][col]\ ^\wedge$
              $(0x02$
              $*\ bloc[1][col])\ ^\wedge\ (0x03$
              $*\ bloc[2][col])\ ^\wedge$
              $block[3][col])$;
      $columnResul[2]$
    $=\ (byte)\ (bloc[0][col]\ ^\wedge\ block[1][col]\ ^\wedge$
    $(0x02$
    $*\ bloc[2][col])\ ^\wedge\ (0x03\ *\ block[3][col]))$;
    $columnResul[3]$
  $=\ (byte)\ ((0x03$
  $*\ bloc[0][col])\ ^\wedge\ block[1][col]\ ^\wedge\ block[2][col]$
  $^\wedge\ (0x02$
  $*\ bloc[3][col]))$;
$for\ (int\ row\ =\ 0;\ row\ <\ 4;\ row+$
$+)$ {
$block[row][col]\ =\ columnResult[row]$;
              }

          }

          }

     $void\ addRoundKey(byte[][]\ block,\ byte[]$
            $roundKey)$ {
          $int\ keyIndex\ =\ 0$;
  $for\ (int\ col\ =\ 0;\ col\ <\ bloc[0].length;\ col+$
              $+)$ {
     $for\ (int\ row\ =\ 0;\ row\ <\ 4;\ row++)$ {
  $block[row][col]\ ^\wedge=\ roundKey[keyIndex++]$;
              }
              }

              }
```

The explanation below uses a 4x4 matrix (which represents the state matrix in AES encryption) to walk through each of the four steps.

Step 1: *Substitution of Bytes*

In this stage, the lookup table known as the S-box provided in the design is used to replace the 16-byte input. Sbox's box measures 16 by 16.

Input Matrix (State):

[$0x63$ $0x7C$ $0x77$ $0x7B$ $0xF2$ $0x6B$ $0x6F$ $0xC5$
$0x30$ $0x01$ $0x67$ $0x2B$
$0xFE$ $0xD7$ $0xAB$ $0x76$]

Each byte in the matrix is updated using the S-box lookup table. For simplicity, assume $Sbo[0x63]\ =\ 0x09$, $Sbo[0x7C]\ =\ 0xEF$, and so on. Output Matrix (After Substitution):

[$0x09$ $0xEF$ $0xFC$ $0x4D$ $0x10$ $0x6A$ $0xDD$ $0x2E$
$0x87$ $0xC2$ $0xB3$ $0x5F$
$0x12$ $0xA4$ $0x5B$ $0x6C$]

Step 2: *Shift Rows (RowShift)*

In this step, every matrix row is shifted to the left cell. The shifting pattern is as follows:

The first row of the matrix stays the same. The second row shifts left by one cell.
The third row shifts left by two cells. The fourth row shifts left by three cells.
As a result, a new 4x4 matrix containing the same element in the cell at various locations is created during each cycle. Input Matrix:

[$0x09$ $0xEF$ $0xFC$ $0x4D$ $0x10$ $0x6A$ $0xDD$ $0x2E$
$0x87$ $0xC2$ $0xB3$ $0x5F$
$0x12$ $0xA4$ $0x5B$ $0x6C$]

Output Matrix (After Shifting):

[$0x09$ $0xEF$ $0xFC$ $0x4D$ $0x6A$ $0xDD$ $0x2E$ $0x10$
$0xB3$ $0x5F$ $0x87$ $0xC2$
$0x6C$ $0x12$ $0xA4$ $0x5B$]

Step 3: *Mix Columns (MixCol)*

In this stage, a mathematical method is used to convert each column, which consists of four bytes, into an entirely new set of bytes. As a result, a new matrix with all 16 new bytes is created at each round.
The transformation applies a fixed matrix multiplication: Fixed Matrix: [2 3 1 1
1 2 3 1
1 1 2 3
3 1 1 2]

Example for column: [$0x09, 0x6A, 0xB3, 0x6C]^\wedge T$
Each value in the new column is computed as:

New Byte = $2 \cdot byte1 \oplus 3 \cdot byte2 \oplus 1 \cdot byte3 \oplus 1 \cdot byte4$
For simplicity, assume calculations yield:
New Column: $[0x8E, 0x5C, 0x36, 0x2F]$^T
Apply this to all columns. Output Matrix (After Mixing):

$[0x8E\ \ 0x3D\ \ 0x7A\ \ 0x4F\ 0x5C\ \ 0xF4\ \ \ 0x1C\ \ 0x29$
$0x36\ \ 0x12\ \ 0x98\ \ 0x5B$
$0x2F\ \ 0xA4\ \ 0xC3\ \ 0x09]$

Step 4: *Add Round Key (AddRoundKey)*

The 256 bits of the round key are XORed with the new 16- byte (128-bit) matrix in the AddRoundKey step. In the final phase, the cypher text will be the final XORed value; if not, the 16-byte matrix will advance to the next round.
Round Key Matrix:

$[0xA0\ \ 0x88\ \ 0x23\ \ 0x2A\ 0xFA\ \ 0x54\ \ 0xA3\ \ 0x6C$
$0xFE\ \ 0x5A\ \ 0xE5\ \ 0x30$
$0x33\ \ 0x5C\ \ 0xD3\ \ 0x88]$
Input Matrix (State):

$[0x8E\ \ 0x3D\ \ 0x7A\ \ 0x4F\ 0x5C\ \ 0xF4\ \ \ 0x1C\ \ 0x29$
$0x36\ \ 0x12\ \ 0x98\ \ 0x5B$
$0x2F\ \ 0xA4\ \ 0xC3\ \ 0x09]$

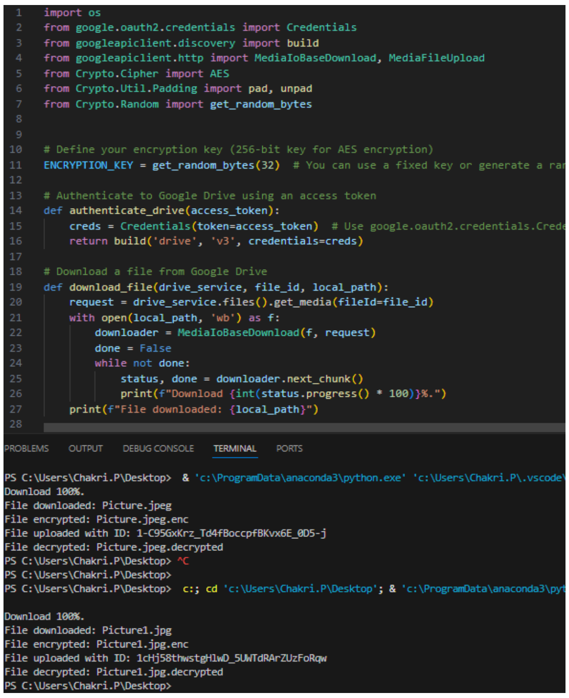
Perform XOR operation for each byte:

Example: $0x8E\ \oplus\ 0xA0\ =\ 0x2E$
Output Matrix (After XOR):

$[0x2E\ \ 0xB5\ \ 0x59\ \ 0x65\ 0xA6\ \ 0xA0\ \ 0xBF\ \ 0x45$
$0xC8\ \ 0x48\ \ 0x7D\ \ 0x6B$
$0x1C\ \ 0xF8\ \ 0x10\ \ \ 0x81]$

Final Result:

The output matrix after these steps represents the encrypted state for that round. This process repeats for a set number of rounds depending on the AES key size (128, 192, or 256 bits).
Each step in the AES rounds is reversible and rely on a structure of substitution and permutation . The encryption begins with its first operation, shown in Fig. 4, is
(i) Key Expansion.
(ii) The first round in which the AddRoundKey step is executed
(iii) Rounds in which the aforementioned cycle is carried out based on the key size employed, and
(iv) Final Round, which incorporates AddRoundKey, ShiftRows, and Byte replacement. It is simply the opposite of encryption for decryption as shown in Fig. 3.



```
import os
from google.oauth2.credentials import Credentials
from googleapiclient.discovery import build
from googleapiclient.http import MediaIoBaseDownload, MediaFileUpload
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes


# Define your encryption key (256-bit key for AES encryption)
ENCRYPTION_KEY = get_random_bytes(32)  # You can use a fixed key or generate a ran

# Authenticate to Google Drive using an access token
def authenticate_drive(access_token):
    creds = Credentials(token=access_token)  # Use google.oauth2.credentials.Crede
    return build('drive', 'v3', credentials=creds)

# Download a file from Google Drive
def download_file(drive_service, file_id, local_path):
    request = drive_service.files().get_media(fileId=file_id)
    with open(local_path, 'wb') as f:
        downloader = MediaIoBaseDownload(f, request)
        done = False
        while not done:
            status, done = downloader.next_chunk()
            print(f"Download {int(status.progress() * 100)}%.")
    print(f"File downloaded: {local_path}")

```

```
PS C:\Users\Chakri.P\Desktop>  & 'c:\ProgramData\anaconda3\python.exe' 'c:\Users\Chakri.P\.vscode\
Download 100%.
File downloaded: Picture.jpeg
File encrypted: Picture.jpeg.enc
File uploaded with ID: 1-C95GxKrz_Td4fBoccpfBKvx6E_0D5-j
File decrypted: Picture.jpeg.decrypted
PS C:\Users\Chakri.P\Desktop> ^C
PS C:\Users\Chakri.P\Desktop>
PS C:\Users\Chakri.P\Desktop>  c:; cd 'c:\Users\Chakri.P\Desktop'; & 'c:\ProgramData\anaconda3\pyt

Download 100%.
File downloaded: Picture1.jpg
File encrypted: Picture1.jpg.enc
File uploaded with ID: 1cHj58thwstgHlwD_5UWTdRArZUzFoRqw
File decrypted: Picture1.jpg.decrypted
PS C:\Users\Chakri.P\Desktop>
```

Fig. 5

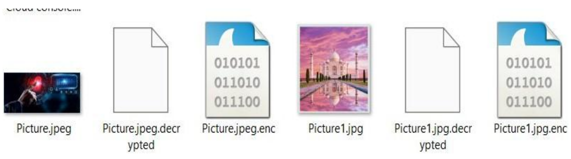


Fig. 6

IV. RESULT ANALYSIS

Strong encryption is ensured by utilizing Python to implement the AES algorithm with a 256-bit key size to protect sensitive data. A 256-bit key offers a remarkable degree of security, making it nearly hard for attackers to crack using brute force methods because of the enormous amount of calculation time needed. This strong encryption prevents unwanted access to data. After encryption, the data can be safely saved on the cloud for dependable and effective access. GCP, a well-known cloud service provider, provides a scalable and easily accessible cloud storage option (Fig. 5). GCP offers on-demand cloud computing solutions to consumers and businesses, guaranteeing smooth data management, backup, and retrieval.
It's strong infrastructure and cutting-edge security measures preserve the availability and integrity of the encrypted data. The end-to-end secure

environment for data processing and storage is ensured by the combination of AES encryption and GCP storage, meeting contemporary security and accessibility requirements.
AES encryption secures data by using a key to make it unreadable, preventing unauthorized access. Confidentiality is ensured since attackers cannot decrypt the data without the right key. Because of this robust encryption system, it is computationally impossible for unauthorized parties to recover the original data (Fig. 6).

V. CONCLUSION

By processing data closer to the source, at the network edge (Fig. 1), fog computing improves on conventional IoT-cloud systems by lowering latency, increasing accuracy, and guaranteeing higher consistency [3]. Fog computing reduces reaction times and improves overall Quality of Service (QoS) by serving as an intermediary layer to reduce the dangers associated with sending sensitive data to the cloud over long distances. The IoT-fog-cloud architecture is essential as IoT devices and real-time compute demands rise [8]. Performance is further optimized by real-time data monitoring and fog's instantaneous processing capabilities. By converting data into an unintelligible format that is computationally hard for unauthorized parties to decrypt, AES encryption strengthens security in intelligent applications and guarantees data confidentiality.

VI. REFERENCES

[1]. Mayayise, T.O., Osunmakinde, I.O.: A compliant assurance model for assessing the trustworthiness of cloud- based e-commerce systems. In: IEEE (2013)

[2]. Majid Hajibaba and Saeid Gorgin. A Review on Modern Distributed Computing Paradigm: Cloud Computing, Jungle Computing and Fog Computing. Journal of Computing and Information Technology 2014, Volume 2, pp. 69-84.

[3]. Dr. Jordan Shropshire. Extending the Cloud with Fog: Security Challenges and Opportunities. In 20th Americas Conference on Information System, Savannah, 2014.

[4]. Nasiri, R., Hosseini, S.: A case study for a novel framework for cloud testing. In: IEEE (2014)

[5]. Alrawais, Arwa, Abdulrahman Alhothaily, Chunqiang Hu, and Xiuzhen Cheng. "Fog computing for the internet of things: Security and privacy issues." IEEE Internet Computing, 21, no. 2 (2017): 34-42.

[6]. Elminaam, Diaa Salama Abdul, Hatem Mohamed Abdul Kader, and Mohie Mohamed Hadhoud. "Performance evaluation of symmetric encryption algorithms." IJCSNS International Journal of Computer Science and Network Security, 8, no. 12 (2008): 280-286.

[7]. P. C. van Oorschot and M. J. Wiener, "A KnownPlaintext Attack on Two-Key Triple Encryption," Lecture Notes in Computer Science 473, Advances in Cryptology— Eurocrypt '90 Proceedings, Springer-Verlag, Berlin, 1991, pp. 318-325.

[8]. F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in Big Data and Internet of Things: A Roadmap for Smart Environments. Springer, 2014, pp. 169–186.

[9]. J. Burt, "Fog computing aims to reduce processing burden of cloud systems," http://www.eweek.com/networking/ fog-computing-aims-to-reduce-processing-burden-of-cloud- systems. html, 2010.

[10]. Y. Chen, W. Shen, H. Huo, and Y. Xu, "A smart gateway for health care system using wireless sensor network," in 2010 Fourth International Conference on Sensor Technologies and Applications (SENSORCOMM), July 2010, pp. 545–550.

[11]. Wang, Qixu, et al. "PCP: A Privacy-Preserving Content- Based Publish–Subscribe Scheme With Differential Privacy in Fog Computing." IEEE Access, 5 (2017): 17962-17974.

[12]. Joan Daemen and Vincent Rijmen, "The Design of Rijndael: AES The Advanced Encryption Standard," Springer-Verlag, 2002.

[13]. Vishwanath, Akhilesh, et al. Security in fog computing through encryption. DigitalCommons@ Kennesaw State University, 2016.

[14]. Abdullah Al Hasib, Abul Ahsan Md. Mahmudul Haque, "A Comparative Study of the Performance and Security Issues of AES and RSA Cryptography", Third International Conference on Convergence and Hybrid Information Technology, 2008.

[15]. Sadikin, Mohamad Ali, and Rini Wisnu Wardhani. "Implementation of RSA 2048-bit

and AES 256-bit with digital signature for secure electronic health record application." IEEE International Seminar on Intelligent Technology and Its Applications (ISITIA), 2016, pp. 387- 392.

[16]. F. Hendessi and M. R. Aref, "A Successful Attack Against the DES," Information Theory and Application, Third Canadian Workshop Proceedings, 1994, pp. 78-90.

[17]. Li Dongjiang, Wang Yandan, Chen Hong, “The research on key generation in RSA public- key cryptosystem”, Department of Computer Science, North China Electric Power University, Beijing, China, Fourth International Conference on Computational and Information Sciences 2012.