JURNAL INFOTEL
VOL. 18, NO. 1, FEBRUARY 2026, PP. 114–127. DOI:10.20895/INFOTEL.V18I1.1429

RESEARCH ARTICLE

Socket-Based File Transfer System using AES-256 and OTP Authentication

Feni Widianti[1], Fauza Khair[2], Agung Mulyo Widodo[3], and Eko Fajar Cahyadi[4,*]

[1]Telecommunication Engineering Study Program, Institut Teknologi Telkom Purwokerto, Purwokerto 53147, Indonesia
[2,4]Telecommunication Engineering Study Program, Telkom University, Purwokerto 53147, Indonesia
[3]Smart Technology and Sustainability Center, National Kaohsiung University of Science and Technology, Kaohsiung 81157, Taiwan

*Corresponding email: ekofajarcahyadi@telkomuniversity.ac.id

Received: August 13, 2025; Revised: December 13, 2025; Accepted: January 10, 2026.

Abstract: The increasing risk of data interception during file transmission over open networks requires the development of secure communication systems. This study proposes a secure file transfer scheme that integrates the Advanced Encryption Standard (AES) with a 256-bit key and One-Time Password (OTP)-based authentication over socket programming. The system ensures that only verified users can transmit files by requiring password log-in and OTP verification via email. Upon successful authentication, files are encrypted using AES-256 before being transmitted over a TCP/IP socket connection. The implementation is carried out in Python using Visual Studio Code, with performance evaluated based on encryption time, transfer speed, and resistance to brute-force attacks. Various file types and sizes, including text, documents, images, audio, video, and compressed files, were tested to validate the robustness and efficiency of the system. The results show that the proposed system maintains high data integrity, enforces strong access control, and effectively resists unauthorized access, making it suitable for applications requiring secure file exchange.

Keywords: AES-256, one-time password (OTP), socket programming, user authentication, secure file transfer, brute-force resistance

1 Introduction

In the era of digital communication, the risk of data interception during transmission has increased significantly, especially in open network environments. Socket programming

 ISSN: 2085-3688; e-ISSN: 2460-0997

enables devices to exchange data efficiently over TCP-IP, but it also exposes communication to potential threats such as sniffing, phishing, and brute-force attacks [1–3]. If exploited, these vulnerabilities can lead to serious consequences, including identity theft and financial losses [4].

To address these challenges, cryptography plays a critical role in ensuring the confidentiality and integrity of data during transmission. Among various encryption algorithms, the Advanced Encryption Standard (AES) stands out due to its balance of performance and security. In particular, AES-256, with its longer key length and 14 processing rounds, provides greater protection than AES-128 and AES-192 [5]. It is well-suited for implementation even in systems with limited memory, making it practical for a wide range of applications [6].

Previous studies have explored encryption performance and file transfer over sockets. Meko [7] compared the performance of DES, AES, IDEA, and Blowfish in terms of encryption time and output size, but did not consider authentication mechanisms. Anshori [8] successfully implemented file transfer via socket programming but did not incorporate any encryption, thereby exposing the data to potential interception. Other research by Endrayanto *et al.* [9] and Kheshaifaty & Gutub [10] addressed encryption in IoT and secure log-in systems, respectively, but did not integrate one-time password (OTP)-based authentication with encrypted file transmission over sockets. Meanwhile, Dian *et al.* [11] evaluated AES with varying key lengths (128, 192, and 256 bits) in the context of login systems for mobile applications. The study revealed that longer keys increased security, but also required more processing time. Although informative on key size effects, the study did not implement or evaluate file transmission systems or OTP mechanisms.

To fill this gap, this study proposes a secure file transfer system that uniquely integrates AES-256 encryption with OTP-based authentication within a socket programming framework. Unlike existing socket-based file transfer approaches that focus solely on encryption, the proposed system enforces a two-step authentication process: password verification followed by an email-based one-time password (OTP), ensuring that only legitimate, actively verified users can initiate file transmission. After successful authentication, files are encrypted with AES-256 and transmitted securely over a TCP/IP socket, thereby strengthening access control and data confidentiality while maintaining efficient file transfer performance.

The system's performance is evaluated based on encryption time, file transfer speed, and resistance to brute-force attacks, using tools such as CrypTool. Various file formats are tested—including text, images, audio, and video—to assess the system's robustness and versatility. The results demonstrate that this integrated approach significantly improves the security of file transmission in network environments.

2 Preliminaries

This section introduces the fundamental concepts underlying the proposed secure file transmission system, including the AES-256 encryption algorithm, socket programming, and brute-force attacks.

2.1 AES-256 Encryption

The AES is a symmetric block cipher standardized by the National Institute of Standards and Technology (NIST) in 2001. It supports three key lengths: 128, 192 and 256 bits, corresponding to AES-128, AES-192, and AES-256, respectively. Among these, AES-256 offers the highest level of security due to its 256-bit key size, which provides 2^{256} possible keys, making brute force attacks computationally infeasible with existing technology [12, 13].

AES encrypts data in fixed-size 128-bit blocks, and AES-256 performs 14 rounds of transformation operations. Each round consists of four stages:

1. `SubBytes`: Byte substitution using an S-box.
2. `ShiftRows`: Row-wise permutation.
3. `MixColumns`: Column-wise mixing using matrix multiplication.
4. `AddRoundKey`: XOR with the round key.

The AES-256 key schedule (key expansion algorithm) generates 15 round keys from the initial key, contributing to its resistance to differential and linear cryptanalysis [14].

AES-256 is widely adopted in high-security applications such as VPNs (IPsec), TLS, BitLocker, PGP, and Secure File Transmission, due to its compliance with international standards (*e.g.,* ISO / IEC 18033-3) and its proven resistance to known cryptanalytic attacks [15, 16]. In this research, AES-256 is utilized to encrypt the file contents before transmission, ensuring data confidentiality even if the transmission channel is intercepted.

2.2 Socket Programming

Socket programming is essential for network communication, providing a standardized interface for connecting client and server programs. A socket is an abstraction denoting endpoints in a network connection, characterized by a unique combination of an IP address and a port number, which facilitates data transfer using network protocols such as TCP/IP and UDP [17]. Socket communication involves creating a socket, binding it to an address, listening for incoming connections, accepting those connections, transmitting and receiving data, and finally terminating the connection, as illustrated in Fig. 1 [18]. Fig. 1 serves as the basis for our socket programming. Typically, the client initiates contact, whilst the server passively awaits a response to the client's request [19].

Typically, the client initiates the dialogue while the server remains silent, awaiting the client's request [18]. Socket programming facilitates the development of client-server applications, consisting of a client program and a server program, within a distributed computer environment [19]. A bidirectional connection between client and server applications operating within a network environment is called a socket [20]. Provides a reliable communication system for two computers. Python's socket library is used alongside auxiliary libraries like tqdm for transfer monitoring and pycryptodome for AES implementation.

2.3 Brute-force Attacks

Brute force is a common attack technique in which an adversary systematically tries all possible keys or passwords until the correct one is found. Although AES-256 is theoretically resistant due to its vast key space (2^{256} combinations), this study includes brute-force testing using CrypTool to simulate attack scenarios and verify the encryption's robustness against such attempts.

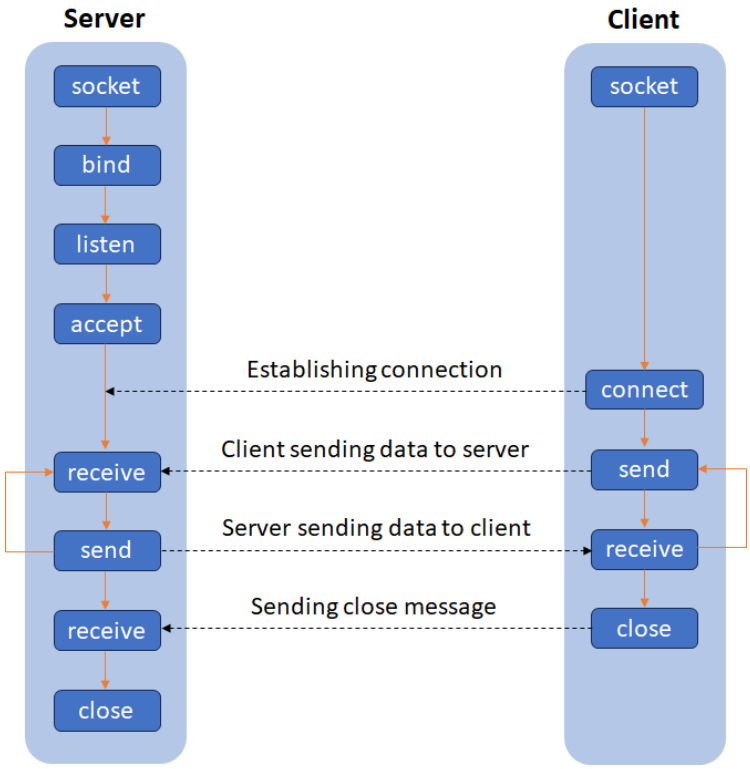

Figure 1: TCP client-server socket programming.

3 Methodology

This section presents the system design, implementation details, and evaluation approach for the proposed secure file transfer system. The methodology encompasses the hardware and software setup, system workflow, authentication and encryption mechanisms, and test scenarios used to assess performance and security.

Table 1: Tools and environment

Tools	**Environment**
Client device	Intel Core i7 (16 GB RAM, Windows 11)
Server device	Intel Core i5 (4 GB RAM, Windows 10)
Programming language	Python 3
Editor	Visual Studio Code
Libraries	pycryptodome for AES, smtplib and email.mime.text for OTP email, socket for communication, tqdm for transfer monitoring
Testing tools	CrypTool for brute force simulation

```
def random_otp(n):
    range_start = 10**(n-1)
    range_end = (10**n)-1
    return randint(range_start, range_end)
```

Figure 2: OTP code transfer.

3.1 System Overview

The proposed system integrates two main components: the authentication layer, which verifies user identity using one-time passwords (OTPs) and email-based passwords, and the encryption and transmission layer, which encrypts user-selected files using AES-256 and transfers them securely via socket programming.

Users must pass OTP-based authentication before initiating any file transfer. Encrypted files are sent from a client to a server over a local TCP/IP socket connection, where they are decrypted and stored securely. Meanwhile, the tools and implementation environment of this study are provided in Table 1.

```
def checkUser(user):
    db = TinyDB('db.json', storage=JSONStorage)

    User = Query()
    search = db.search(User.username == user)
    if search == []:
        return True
    else:
        return False
    return True
```

Figure 3: User verification check on database.

3.2 System Workflow

The test was conducted to implement and evaluate the authentication system, and the file sending process is running safely in the execution of encryption. The program was created with Python, interpreted through Visual Studio Code, and run with Windows Powershell. The system is implemented in four primary Python modules, including

```
def encrypt(key, filename, path):
    chunksize = 64*1024
    outputFile = "enc"+filename
    filesize = str(os.path.getsize(path)).zfill(16)
    IV = Random.new().read(16)

    encryptor = AES.new(key, AES.MODE_CBC, IV)

    with open(path, 'rb') as infile:#rb means read in binar
        with open("file/" + outputFile, 'wb') as outfile:#
            outfile.write(filesize.encode('utf-8'))
            outfile.write(IV)

            while True:
                chunk = infile.read(chunksize)

                if len(chunk) == 0:
                    break
                elif len(chunk)%16 != 0:
                    chunk += b' '*(16-(len(chunk)%16))

                outfile.write(encryptor.encrypt(chunk))
```

Figure 4: User verification check on database.

`TinyDB.py` to handle user data, registration, and login authentication Figure 2, and Figure 3), `Send_mail.py` to send OTP via email using Gmail SMTP, `Client.py` to handle file selection, encryption, and transmission, and `Server.py` to receive encrypted files and perform decryption. The system operates through the following steps:

Table 2: OTP Authentication Result

Scenario	OTP Attempts	Result	System Response
1	Correct on 1st try	Success	Access granted
2	Incorrect on 1st, correct on 2nd	Success	Access granted
3	Incorrect on all three attempts	Failed	Access denied, forced re-login

3.2.1 User registration/login

Users enter their username, password (MD5 encrypted), and email address. For login, credentials are verified against a local JSON database using TinyDB.

```
def encrypt(key, filename, path):
    chunksize = 64*1024
    outputFile = "enc"+filename
    filesize = str(os.path.getsize(path)).zfill(16)
    IV = Random.new().read(16)

    encryptor = AES.new(key, AES.MODE_CBC, IV)

    with open(path, 'rb') as infile:#rb means read in binar
        with open("file/" + outputFile, 'wb') as outfile:#
            outfile.write(filesize.encode('utf-8'))
            outfile.write(IV)

            while True:
                chunk = infile.read(chunksize)

                if len(chunk) == 0:
                    break
                elif len(chunk)%16 != 0:
                    chunk += b' '*(16-(len(chunk)%16))

                outfile.write(encryptor.encrypt(chunk))
```

Figure 5: User verification check on database.

3.2.2 OTP verification

A 6-digit OTP is randomly generated and sent to the user's email. The user is given three attempts to enter the correct OTP. Failure results in system lockout.

3.2.3 File selection and encryption

Upon successful login and OTP verification, users can select a file for transmission. The file is encrypted using AES-256 in CBC mode. A private key is required to initialize encryption.

3.2.4 File transmission

Encrypted files are transmitted over TCP/IP sockets from the client to the server. The file metadata (name, size, and encryption key) is sent ahead of the file chunks.

Table 3: File Encryption and Transmission Performance in Same File Size Dataset (3,143 KB each)

File Type	Encryption Time (s)	Client Speed (KB/s)	Server Speed (KB/s)	Client Time (s)	Server Time (s)
.txt	0.02	771	719	4	4
.rar	0.02	586	569	5	5
.pdf	0.00	621	427	5	7
.docx	0.00	935	877	3	3
.jpg	0.00	936	900	3	3
.png	0.00	544	523	5	6
.mp3	0.00	646	625	4	5
.mp4	0.02	854	797	3	4

Table 4: File Encryption and Transmission Performance in Different File Size Dataset

File Type	Size (KB)	Encryption Time (s)	Client Speed (KB/s)	Server Speed (KB/s)	Client Time (s)	Server Time (s)
.txt	1,101	0.00	3.62	2.77	0	0
.rar	27,718	0.08	3.11	3.03	9	9
.pdf	29,891	0.11	2.17	2.03	14	15
.docx	9,035	0.02	2.12	1.84	3	5
.jpg	20,043	0.05	2.12	2.00	9	10
.png	12,367	0.02	3.12	2.84	4	4
.mp3	6,189	0.02	3.12	2.82	2	2
.mp4	2,117,963	2.03	1.81	1.71	1,197	1,268

3.2.5 Decryption and storage

On the server side, the file is decrypted using the received key and stored in its original format. The encrypted file is deleted after successful decryption.

3.3 Test Scenarios

Three sets of test cases are performed:

3.3.1 OTP authentication testing

The objective is to validate OTP correctness and ensure the system enforces lockout after multiple failed attempts. The test measures how the system responds to varying OTP input attempts, including success at the 1st, 2nd, or 3rd try.

3.3.2 File encryption and transmission testing

The objective is to assess how encryption and transmission perform with different file types and sizes. Eight file types (.txt, .docx, .rar, .pdf, .jpg, .png, .mp3, .mp4) are tested. Two size scenarios are included: identical size (3,143 KB) and varied size (up to 2 GB). Collected

metrics include encryption time, file size before/after encryption, transfer speed (KB/s), and completion time.

3.3.3 Brute force testing

The objective is to determine how resistant AES-256 encryption is to brute-force attacks. We use CrypTool to simulate brute-force attacks on encrypted AES-256 files, evaluating the time required for decryption attempts with incorrect keys.

4 Results and Discussion

This section presents system testing results, including OTP authentication reliability, encryption and transmission performance, and brute-force resistance. The objective is to validate the effectiveness, efficiency, and security of the secure file transfer system.

4.1 OTP Authentication Results

To assess the robustness of the OTP-based authentication mechanism, three test scenarios were conducted, as shown in Table 2. In all cases, the system behaved as expected, including accepting correct OTPs, locking access after three failed attempts, mitigating brute-force attacks, and delivering OTPs via email via SMTP with SSL. This confirms that the OTP mechanism effectively enforces access control and prevents unauthorized access.

4.2 File Encryption and Transmission Performance

The evaluation of file encryption and transmission performance focused on measuring the efficiency of the system when handling various types and sizes of files. This included evaluating the encryption time, transmission speed, and overall file transfer duration on both the client and the server side. The analysis was carried out under two controlled scenarios: (1) transmission of files with uniform sizes and (2) transmission of files with varying sizes and formats. In both scenarios, files were encrypted using AES-256 in CBC mode before transmission through TCP sockets.

4.2.1 Same file size scenario

In this scenario, eight different file types were standardized at 3,143 KB each. The results are summarized in Table 3, which presents the encryption time, the client and server transfer speeds, and the total time taken on both ends. The encryption process was introduced with minimal delay, demonstrating the efficiency of AES-256. The transfer speeds ranged from 500 to 950 KB/s, and all file types were successfully transmitted in less than 7 seconds. Media files such as .jpg and .mp4 exhibited higher throughput than structured or compressed formats. A visual comparison of the transfer speeds between the client and the server is provided in Fig. 6.

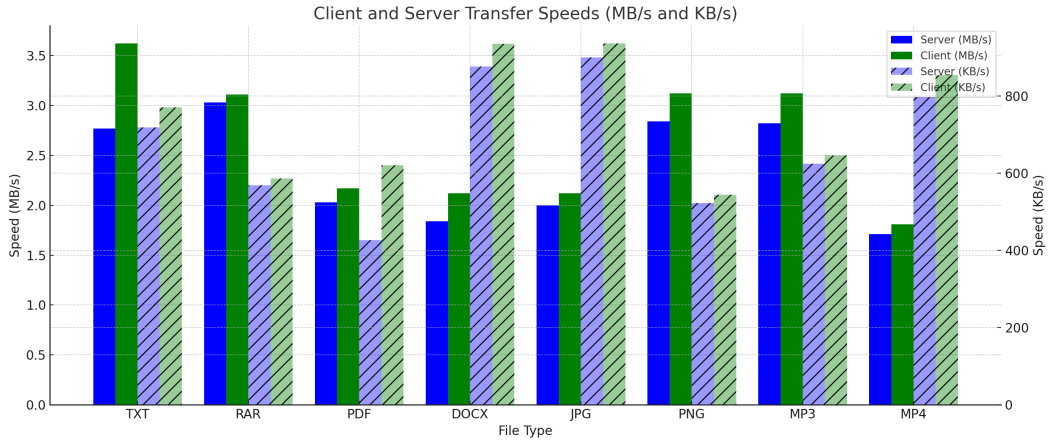

Figure 6: Comparison of transfer speed between client and server in the same (KB/s) and different (MB/s) file size group.

4.2.2 Different file size scenario

This test involved file sizes ranging from 1 MB to more than 2 GB, allowing evaluation of scalability and robustness. Table 4 presents the encryption time and transmission performance for each type of file. In particular, a 2.1 GB *.mp4 file was encrypted in just 2.03 seconds and transmitted without data loss, indicating the system's ability to handle large-scale transfers reliably. Smaller files showed encryption times below 0.1 seconds and transfer speeds between 2–3 MB/s. Comparisons of transmission time for files of the same size and different sizes are depicted in Figure 7, showing consistent performance on both the client and the server sides, despite variations in file type and size.

4.3 Brute Force Resistance

A cryptographic robustness test was conducted using CrypTool to simulate a brute-force attack on an encrypted phits.txt file. The AES-256 algorithm, a symmetric block cipher with a 256-bit key, is mathematically resistant to exhaustive searches. Despite CrypTool's attempt to recover the key, the attack failed due to the large keyspace involved. This aligns with the theoretical cryptographic security assumptions that AES-256 is secure against brute-force attacks, provided the key is generated randomly and properly managed. During simulation, no memory leakage, key inference, or pattern recognition was observed, confirming the correct implementation and strength of the AES-256 encryption.

The use of CBC mode adds complexity by chaining blocks, thereby preventing known-plaintext and ciphertext-prediction attacks. This test validates the cryptographic integrity of the file transfer mechanism and confirms that even if an attacker intercepts encrypted files during transmission, they cannot recover meaningful content without the corresponding decryption key. When combined with an OTP-based authentication system, the system achieves a high level of security assurance. The brute-force resistance test effectively demonstrates that the AES-256 encryption layer provides strong confidentiality guaranties

and is suitable for securing sensitive files in real-world network environments. The brute-force testing process is shown in Figure 8. Meanwhile, the result of brute force testing of encrypted file.txt is shown in Figure 9. It can be seen that to be able to guess using the brute-force method, the estimated time required is very long, which is 5.5×10^{63} years.

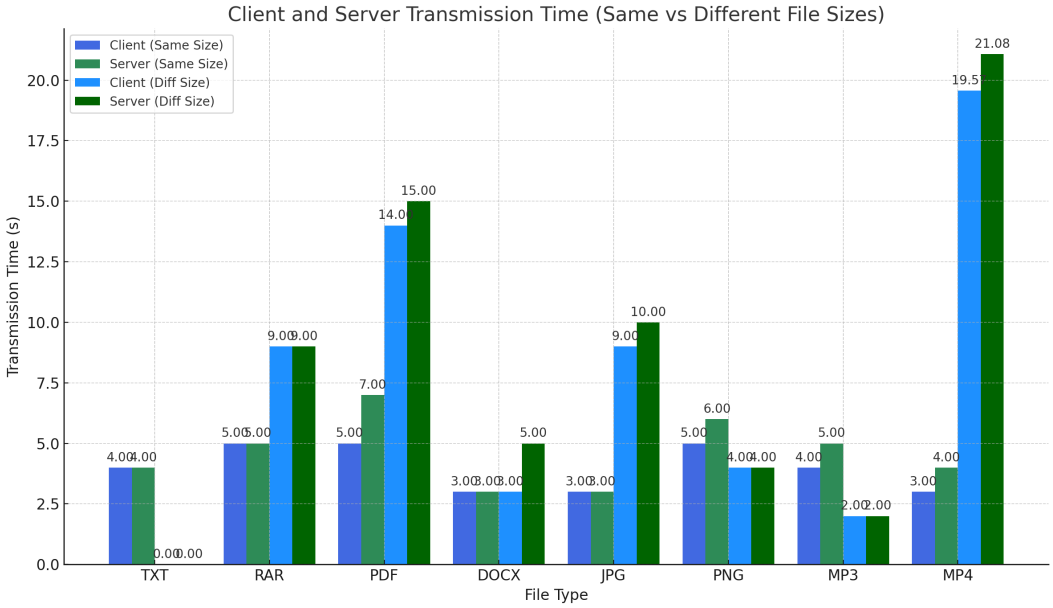

Figure 7: Comparison of transmission time (s) between client and server in same and different file sizes.

5 Discussion

The result confirms that integrating AES-256 encryption with email-based OTP authentication provides a secure and efficient solution for file transfer. The OTP mechanism improves access control by limiting login attempts and requiring time-sensitive email codes, effectively mitigating brute-force and dictionary attacks. AES-256 in CBC mode provides strong encryption across various file types with minimal performance impact. Transmission tests showed stable speeds and scalability, even for large files, with minor variations attributed to hardware and file characteristics. Brute-force simulations using CrypTool validated the cryptographic robustness of the system, making it suitable for secure data exchange within internal networks or private cloud environments.

6 Conclusion

This study presents a secure file transfer system that integrates AES-256 encryption with OTP-based authentication using socket programming, ensuring that only verified users can

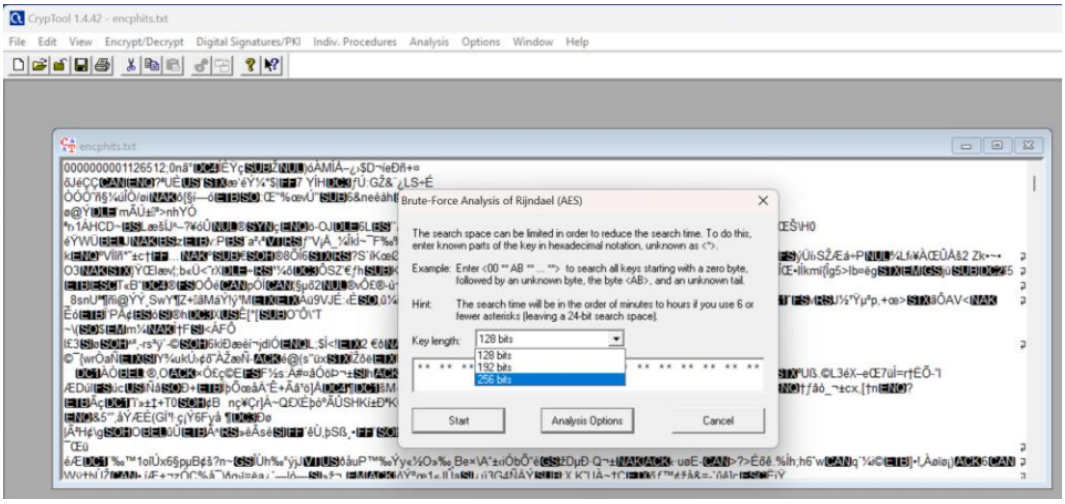

Figure 8: AES key length selection for brute force test.

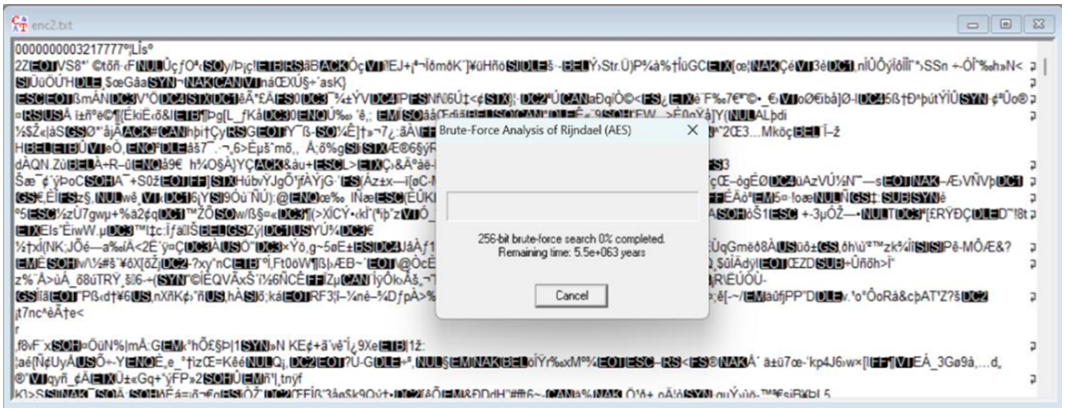

Figure 9: Brute force test results file.txt.

transmit data securely. Testing in various types and sizes of files confirmed that the system provides strong encryption with minimal processing time, stable transmission speeds, and robust resistance to brute-force attacks. The OTP mechanism effectively prevents unauthorized access, while AES-256 ensures data confidentiality. The system successfully handles large file transfers without data loss, demonstrating its scalability and reliability. In general, the proposed approach offers a practical and secure solution for file transmission over networks, with potential future enhancements including digital signature support and a graphical user interface for improved usability.

Acknowledgment

This research is fully supported by the Flagship Internal Research Grant (Hibah Penelitian Internal Unggulan) of the Institute of Research and Community Services LPPM, Institut Teknologi Telkom Purwokerto, under contract No. ITTel3000-b/LPPM-000/Ka.LPPM/III/2023.

References

[1] T. Murkomen, "Performance, privacy, and security issues of tcp/ip at the application layer: A comprehensive survey," *GSC Advanced Research and Reviews*, vol. 18, no. 3, pp. 234–264, 2024.

[2] O. O. Felix, "Tcp/ip stack transport layer performance, privacy, and security issues," *World Journal of Advanced Engineering Technology and Sciences*, vol. 11, no. 2, pp. 175–200, 2024.

[3] A. Abubakar, N. M. Najmuddin, R. A. M. Alwi, and N. A. I. M. Faizal, "Examining potential threats of eavesdropping in tcp stream of personal interactive transmission session," *International Journal on Perceptive and Cognitive Computing*, vol. 10, no. 1, pp. 98–104, 2024.

[4] K. Bala and B. Priya, "Securing file transferring system by implementing aes algorithm," *World Applied Sciences Journal*, vol. 35, no. 9, pp. 1808–1812, 2017.

[5] M. Alawadi, Z. Shukur, and R. Mahmod, "A secure and efficient multi-factor authentication algorithm for mobile money applications," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 21, no. 3, pp. 1624–1632, 2021.

[6] R. Shruthi and S. Ramesh, "Design of secure file transfer over internet using aes and sockets," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 5, no. 11, pp. 101–104, 2016.

[7] D. A. Meko, "Perbandingan algoritma des, aes, idea, dan blowfish dalam enkripsi dan dekripsi data," *Jurnal Teknologi Terpadu*, vol. 4, no. 1, pp. 8–15, 2018.

[8] I. F. Anshori, "Implementasi socket tcp/ip untuk mengirim dan memasukan file text ke dalam database," *Responsif*, vol. 1, no. 1, pp. 1–5, 2019.

[9] R. K. Endrayanto, A. Muttaqin, and R. A. Setyawan, "Advanced encryption standard (aes) pada modul internet of things (iot)," *TELKA*, vol. 5, no. 2, pp. 103–113, 2019.

[10] N. Kheshaifaty and A. Gutub, "Engineering graphical captcha and aes crypto hash functions for secure online authentication," *Journal of Engineering Research*, vol. 11, no. 3, pp. 69–80, 2021.

[11] H. Dian, R. Arifudin, and Alamsyah, "Security login system on mobile application with implementation of advanced encryption standard (aes) using 3 keys variation 128-bit, 192-bit, and 256-bit," *Scientific Journal of Informatics*, vol. 6, no. 1, pp. 34–44, 2019.

[12] National Institute of Standards and Technology, "Fips pub 197: Advanced encryption standard (aes)," tech. rep., U.S. Department of Commerce, 2001.

[13] J. Daemen and V. Rijmen, *The Design of Rijndael: AES – The Advanced Encryption Standard*. Springer, 2002.

[14] N. Ferguson, B. Schneier, and T. Kohno, *Cryptography Engineering: Design Principles and Practical Applications*. Wiley, 2010.

[15] W. Stallings, *Cryptography and Network Security: Principles and Practice*. Pearson, 7 ed., 2017.

[16] ISO/IEC, "Iso/iec 18033-3:2010 information technology – security techniques – encryption algorithms – part 3: Block ciphers." International Standard, 2010.

[17] W. R. Stevens, B. Fenner, and A. M. Rudoff, *Unix Network Programming: The Sockets Networking API*. Addison-Wesley Professional, 2014.

[18] M. R. Kabir, B. B. Y. Ravi, and S. Ray, "A virtual prototyping platform for exploration of vehicular electronics," *IEEE Internet of Things Journal*, vol. 10, no. 18, pp. 16144–16155, 2023.

[19] R. L. R. Maata, R. Cordova, B. Sudramurthy, and A. Halibas, "A virtual prototyping platform for exploration of vehicular electronics," in *2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, (Coimbatore, India), 2017.

[20] T. Sayah, "Conception of client/server system with encrypted data exchanging using sockets," Master's thesis, Mohamed Khider University of Biskra, 2018.