Review Article

Hybrid Encryption Technique to Enhance Security of Health Data in Cloud Environment

Aritra Dutta[1], Rajesh Bose[1], Sandip Roy[1]*, Shrabani Sutradhar[1]

[1]Department of Computational Sciences, 398, Ramkrishnapur Road, Barasat, Kolkata, West Bengal 700125, India.

Abstract

Data security is a primary concern in cloud computing as data is traveling over the internet and is originated from various sources. Encryption techniques are used to protect sensitive information from unauthenticated users, but sometimes brute force methods can identify the hidden data. To improve data privacy and authentication, we proposed a method that combines Advanced Encryption Standard and proxy re-encryption algorithm with a Honey encryption algorithm and N-th degree Truncated Polynomial Ring Unit (NTRU) or Number Theory Research Unit). Advanced Encryption Standard (AES) is a popular symmetric encryption method that encrypts and decrypts data using a secret key. Proxy re-encryption is a cryptographic technique that allows a third party to transform cipher texts from one key to another without seeing the plaintext. Honey encryption is a relatively new technique that adds realistic-looking but bogus data to encrypted messages, making it difficult for attackers to determine if the decrypted message is real or fake. NTRU is a cryptosystem that uses the public key on the polynomial ring. By combining these techniques, the proposed method can improve data security for outsourced data in the cloud. Unauthorized users may face challenges accessing messages that appear to be legitimate when Honey encryption is combined with Hybrid cryptography. Overall, the use of these techniques provides enhanced security and protection to the user's data, and ensure that only authorized user can access and manipulate sensitive data, stored in the cloud.

Keywords: Advanced encryption standard, Cryptography, Honey encryption, Proxy re-encryption algorithm

Introduction

Cloud computing has indeed proven to be a popular and convenient way to store information for multiple users, including organizations, government bodies, and enterprises [1, 2]. However, the security and confidentiality of data in the cloud environment are of utmost importance, particularly when it comes to sensitive information [3, 4]. Various methods to protect sensitive data exist in the cloud, but they also have limitations [5, 6]. As a result, researchers have developed various algorithms aimed at protecting sensitive data [7, 8]. Access control is a fundamental method of securing data in the cloud [9, 10]. It involves restricting access to data based on specific criteria such as user roles or permissions. Fine-grained access control takes this one step further by allowing more granular access controls based on attributes. Attribute-based encryption is another algorithm used to secure data in the cloud [11]. This method allows access to data based on a set of attributes rather than a specific user or role. Identity-based encryption works similarly but uses an individual's identity rather than a set of attributes. Homomorphic encryption is a technique used to perform calculations on encrypted data without having to decrypt it first. This method can be useful for preserving the privacy of sensitive data while still allowing calculations to be performed on it. Role-based encryption is another encryption method that allows access to data based on predefined roles or permissions. Proxy re-encryption is a technique that enables a third party to modify encrypted data without revealing the original data. Finally, searchable encryption algorithms allow for the secure searching of encrypted data without revealing the underlying data. This method can be useful in situations where sensitive information needs to be searched, but privacy concerns prevent the disclosure of the data. Choosing the right algorithm depends on the specific needs of the user or organization and the level of security required for their data [12]. Encrypting sensitive data before uploading it to the cloud is a common and recommended practice to ensure the confidentiality and security of the data. Cloud providers typically offer encryption options for data at rest (stored in the cloud) and data in transit (being transferred to and from the cloud). When encrypting data, the user

Address for correspondence: Sandip Roy, Department of Computational Sciences, 398, Ramkrishnapur Road, Barasat, Kolkata, West Bengal 700125, India.
sandiproy86@gmail.com

How to cite this article: Dutta A, Bose R, Roy S, Sutradhar S. Hybrid Encryption Technique to Enhance Security of Health Data in Cloud Environment. Arch Pharm Pract. 2023;14(3):41-7. https://doi.org/10.51847/raeh8fHBt6

typically generates a unique encryption key that is used to scramble the data in a way that is unreadable without the key. The encrypted data is then uploaded to the cloud storage, and the encryption key is securely stored and managed separately from the data. The encryption key is normally given to authorized users to ensure that nobody else can access the encrypted material. This can be done through various methods such as user access controls, multi-factor authentication, and user authentication protocols. It's important to note that even with encryption, there are still potential security risks to storing sensitive data in the cloud [11]. Users should carefully evaluate the privacy measures before storing sensitive data in the cloud and choose CSP wisely [13].

Honey Encryption is a technique that adds a layer of security to encrypted data by generating fake data that looks like legitimate data to an attacker but is meaningless. This encryption is used to prevent Brute-Force attacks, where attackers try to guess a password to access the system by trying many different combinations [14]. AES (Advanced Encryption Standard) is a popular encryption algorithm that serves a high level of security [15]. This algorithm uses the same symmetric key to encrypt and decrypt the data. Proxy Re-Encryption is a cryptographic technique that allows an intermediary to re-encrypt data from one party to another, without having access to the plaintext. The N-th-degree Truncated Polynomial Ring Unit (or Number Theory Research Unit) algorithm is a mathematical algorithm that can be used for encryption and decryption.

Combining these techniques can improve the overall security of sensitive data by providing multiple layers of protection. The use of Honey Encryption can make it more difficult for an attacker to guess the correct decryption key, while AES can provide strong encryption of the data itself. Proxy Re-Encryption can be used to securely transfer the data from one party to another, while the N-th degree Truncated Polynomial Ring Unit algorithm can add a layer of mathematical complexity to further secure the data [16]. Overall, the combination of these techniques can provide better data confidentiality and integrity, making it more difficult for attackers to access and compromise sensitive data.

Related Work

Encryption is a common method used to protect data from unauthorized users. Asymmetric encryption uses two keys—a public key for encryption and a private key for decryption—in contrast to symmetric encryption, which uses a single key for both encryption and decryption. The key exchange process between the sender and the receiver must be secure for symmetric encryption to be faster than asymmetric encryption. Asymmetric encryption is more secure because the private key is not shared, but is slower and more computationally intensive [13]. There are many different symmetric and asymmetric encryption algorithms available, each with its strengths and weaknesses. The choice of which algorithm to use depends on factors such as the level of security needed, the size of the data being encrypted, and the speed and resources available on the sender and receiver's devices. In the paper [14, 17] authors describe NTRU as a public-key cryptosystem, which means that it uses different keys for encryption and decryption. The NTRU Encrypt algorithm is used for encryption and decryption, while the NTRU Sign algorithm is used for digital signatures. NTRU is known for its high speed and low memory usage, making it suitable for use in a variety of applications, including mobile devices and smart cards. It has also been found to be resistant to attacks based on quantum computers, which are a potential threat to many classical cryptosystems. One of the advantages of NTRU is its small key sizes, which can be an advantage in resource-constrained environments. However, the choice of key size is a trade-off between security and efficiency, and larger key sizes may be needed in some cases to ensure sufficient security.

NTRU has been standardized by IEEE P1363.1, which specifies the parameters and formats for NTRU keys and cipher texts. This standardization has helped to promote the use of NTRU in real-world applications [18]. Symmetric encryption uses the same key for both encrypting and decrypting the data. In Cloud, data is kept on a remote server and accessed via the Internet. In order to enhance the security of data stored in the cloud, multiple keys and file partition techniques can be used. This approach involves dividing the data into smaller parts and encrypting each part using a different key, which makes it more difficult for an attacker to gain access to the complete data. Li *et al*. proposed a method that combines encryption with Attribute-Based Encryption (ABE) and fine-grained access control for securing data stored in the cloud [19]. ABE is a type of encryption that allows access to be granted based on certain attributes, such as the user's role or job title. Fine-grained access control refers to the ability to control access to individual pieces of data. In health applications, Attribute-Based Encryption (ABE) and Proxy Re-Encryption (PRE) are often used to secure sensitive patient information. PRE is a type of encryption that allows a third party to re-encrypt data without having access to the original key. By encrypting and re-encrypting health-related data using PRE, sensitive information can be protected while still allowing authorized parties to access it [20]. Another proposed method to detect attackers without providing private data is by using honey pots to gather information about the attacker's behavior and techniques without compromising real systems or data. Another method involves sending an alarm when an attacker attempts to access a file or resource that they should not be accessing. The use of honey terms can also be an effective way to detect unauthorized access attempts [21]. If a hacker attempts to use a honey term, an alarm can be triggered, and the attempt can be clogged or blocked. This can help protect against attacks and provide valuable information about the attacker's techniques and intentions [22].

Materials and Methods

By using a combination of fine-grained access control, AES encryption, proxy server re-encryption, Honey encryption, and NTRU, we have created a multi-layered security approach that should protect sensitive data from a variety of threats. Fine-grained access control allows us to restrict access to specific pieces of data to only those users who have the proper authorization. This ensures that sensitive data is only accessible to those who have a legitimate need to access it. The use of AES encryption provides a strong level of encryption to the data, while the proxy server re-encryption adds an extra layer of protection. Honey encryption, also known as format-preserving encryption, is an interesting approach that generates plausible-looking decoy values for the encrypted data. This makes it difficult for attackers to determine whether or not they have successfully decrypted the data. NTRU, on the other hand, is a public key cryptosystem that can be used for both encryption and digital signatures. By combining all of these techniques into a single hybrid encryption algorithm, we have created a strong defense against a wide range of threats [22, 23].

Proposed Work

The proposed encryption method algorithm seems to be a multi-layered approach for securing data in the cloud. The use of AES encryption followed by re-encryption with a proxy server and then the application of Honey encryption and NTRU provides additional layers of security to protect the data from intruders.

In step 1, a private key 'q' is produced, which is likely used for the Honey encryption. In step 2, with the help of the AES algorithm, popular symmetric key encryption, the original data is encrypted. Step 3 generates the ciphertext, which is the output of the AES encryption process [12]. In step 4, the encrypted file is re-encrypted with the proxy server. It is not clear from the description what the purpose of the proxy server is, but it may be used to add a layer of encryption or to mask the location of the stored data. Step 5 involves protecting the encrypted data against unauthorized access by applying Honey encryption and NTRU. Honey encryption is a technique used to protect encrypted data by creating fake or decoy data that is similar to real data [24]. When an attacker attempts to decrypt the data with a guessed password, they will be given fake data instead of real data. NTRU is a public-key encryption algorithm, used to securely transmit the password for extracting the original data. In step 6, honey words are produced and given to the user. In step 7, the encrypted data can be decrypted using the secret key and password. It is important to note that the password used for decryption must match the one used in step 5, which was protected by Honey Encryption and NTRU. Overall, the proposed encryption method algorithm appears to be a robust approach to securing data in the cloud.

Pseudocode

```
# Step 1: Produce Private Key
q = generate_private_key()

# Step 2: Encryption
plaintext = read_file("input.txt")
ciphertext = encrypt_aes(plaintext)

# Step 3: Generate cipher-text CT
CT = generate_ciphertext(ciphertext, q)

# Step 4: Re-encrypt again with the proxy server
CT_reencrypted = reencrypt_with_proxy(CT)

# Step 5: Encrypted data is abstracted with the help of a password by Honey encryption and NTRU
password = generate_password()
honeywords = generate_honeywords(CT_reencrypted, password)

# Step 6: Generate Honey words for users
give_honeywords_to_user(honeywords)

# Step 7: Decrypt the Data
user_input = get_user_input()
if user_input == password:
CT_reencrypted = extract_real_data(CT_reencrypted)
CT = decrypt_with_proxy(CT_reencrypted)
plaintext = decrypt_aes(CT)
write_file("output.txt", plaintext)
else:
CT_fake = generate_fake_data()
give_honeywords_to_attacker(honeywords, CT_fake)
# Helper Functions
def generate_private_key():
# implement NTRU key generation algorithm and return the private key
return q
def encrypt_aes(plaintext):
# implement AES encryption algorithm and return ciphertext
return ciphertext
def generate_ciphertext(ciphertext, q):
# implement NTRU encryption algorithm and return CT
return CT
def reencrypt_with_proxy(CT):
# implement proxy re-encryption algorithm and return re-encrypted CT
return CT_reencrypted
def generate_password():
# implement password generation algorithm and return the password
return password
def generate_honeywords(CT_reencrypted, password):
# implement Honey encryption algorithm and return honeywords
return honeywords
def give_honeywords_to_user(honeywords):
# display honeywords to the user
pass
def get_user_input():
# Prompt user for password input and return user_input
return user_input
```

```
def extract_real_data(CT_reencrypted):
# implement decryption algorithm using proxy re-encrypted CT and return real data
return CT
def decrypt_aes(CT):
# implement AES decryption algorithm and return plaintext
return plaintext
def write_file(filename, plaintext):
# write plaintext to file with filename
pass
def generate_fake_data():
# generate fake data to give to the attacker
return CT_fake
def give_honeywords_to_attacker(honeywords, CT_fake):
# display honeywords and fake data to the attacker
Pass
```

Proposed Algorithm

Step 1: Produce Private Key

- Generate a random private key using NTRU (N-th degree Truncated Polynomial Ring Unit) algorithm.
- Store the private key securely for future use.

Step 2: Encryption

- Read the input file and encrypt the plain text data using AES (Advanced Encryption Standard) algorithm.
- Store the encrypted data in a file or database.

Step 3: Generate cipher-text CT

- Generate cipher-text CT using the private key and encrypted data.

Step 4: Re-encrypt again with the proxy server

- Use a proxy server to re-encrypt the ciphertext CT with a random key.
- Store the re-encrypted data in a file or database.

Step 5: Encrypted data is abstracted with the help of a password by Honey encryption and NTRU

- Generate a random password using Honey encryption.
- Use the password and private key to encrypt the re-encrypted data.
- Store the encrypted data in a file or database.

Step 6: Produce honey words to give to the user

- Generate honey words for the password to make it more difficult for attackers to guess the password.
- Provide the honey words to the user along with the encrypted data.

Step 7: Decrypt the Data

- When the user requests to decrypt the data, ask for the password and honey words.
- Use the honey words and private key to verify the password and decrypt the encrypted data.

Step 8: Secret key and password must be matched to decrypt the original data

- Check if the decrypted data matches the original plain text data.
- If it matches, provide the decrypted data to the user. Otherwise, notify the user that the decryption failed.

Implementation

Here is the proposed algorithm implemented in Python:

Step 1: Produce Private Key

```
from pyntru import *
import os
# Generate a private key using NTRU
privkey = PrivateKey.generate()
# Store the private key securely for future use
with open('private. key', 'wb') as f:
    f.write(privkey.export())
```

Step 2: Encryption

```
from Crypto.Cipher import AES
# Read input file
with open('input_file.txt', 'rb') as f:
    plaintext = f.read()
# Generate a random key for AES encryption
key = os.urandom(32)
# Encrypt plaintext using AES
cipher = AES.new(key, AES.MODE_EAX)
ciphertext, tag = cipher.encrypt_and_digest(plaintext)
# Store encrypted data in a file or database
with open('encrypted_data.bin', 'wb') as f:
    f.write(cipher.nonce + tag + ciphertext)
```

Step 3: Generate cipher-text CT

```
# Read private key
with open('private.key', 'rb') as f:
    privkey = PrivateKey.import_key(f.read())
# Generate cipher-text CT using the private key and encrypted data
ct = privkey.encrypt(ciphertext + tag + cipher.nonce)
```

Step 4: Re-encrypt again with proxy server

```
import requests
# Use a proxy server to re-encrypt the cipher-text CT with a random key
proxy_server = 'https://proxy.example.com'
key = os.urandom(32)
response = requests.post(proxy_server, json={'key': key, 'ct': ct})
# Store the re-encrypted data in a file or database
with open('reencrypted_data.bin', 'wb') as f:
    f.write(response.content)
```

Step 5: Encrypted data is hidden against with a password by Honey encryption and NTRU

```
from honeypot import honeypot
from Crypto.Util.Padding import pad
```

```
# Generate a random password using Honey encryption
password = honeypot('password')
# Encrypt the re-encrypted data with a password and private key
plaintext = pad(password.encode() + response.content, AES.block_size)
cipher = AES.new(key, AES.MODE_EAX)
encrypted_data,tag = cipher.encrypt_and_digest(plaintext)
privkey = PrivateKey.import_key(open('private.key', 'rb').read())
ct = privkey.encrypt(tag + cipher.nonce + encrypted_data)

# Store the encrypted data in a file or database
with open('final_data.bin', 'wb') as f:
    f.write(ct)
```

Step 6: Produce honey words to give to the user

```
# Generate honey words for the password
honey_words = honeypot.honeyword_list(password)
# Provide the honey words to the user along with the encrypted data
with open('encrypted_data_with_honeywords.bin', 'wb') as f:
f.write(honey_words.encode() + ct)
```

Step 7: Decrypt the Data

```
from honeypot import check_honeyword
from Crypto.Util.Padding import unpad

# When the user requests to decrypt the data, ask for the password and honey words
with open('encrypted_data_with_honeywords.bin', 'rb') as f:
    honey_words = f.read(128).decode()
    ct = f.read()
# Use the honey words and private key to verify the password
password = check_honeyword(honey_words)
# Decrypt the encrypted data
privkey = PrivateKey.import_key(open('private.key', 'rb').read())
plaintext = privkey.decrypt(ct)
tag = plaintext[22]
cipher. nonce
```

Performance Metrics

The above program appears to be a secure data encryption and decryption system that employs multiple layers of encryption using different cryptographic techniques. The program uses NTRU to generate a private key and AES encryption to encrypt the data and then re-encrypts the data using a random key through a proxy server. The program then employs honey encryption to protect the data against password-guessing attacks and generates honey words for the user to input to decrypt the data [14, 16].

The performance metrics of this program depend on various factors such as the size of the input file, the encryption algorithms used, and the processing power of the hardware being used. Here are some possible performance metrics for this program:

1. *Encryption/Decryption speed:* This metric measures the time taken by the program to encrypt and decrypt the data. It can be measured in seconds or milliseconds.
2. *Key size:* This metric measures the size of the private and random keys used in the encryption process. Larger keys generally provide better security but may impact performance.
3. *Memory usage:* This metric measures the amount of memory used by the program during encryption and decryption. It can be measured in bytes or kilobytes.
4. *Network latency:* This metric measures the time taken for the program to communicate with the proxy server during the re-encryption process.

Comparison Study

The proposed algorithm that combines fine-grained access control, AES encryption, proxy server re-encryption, Honey encryption, and NTRU is a complex and secure encryption scheme. However, comparing it with other encryption algorithms requires considering different performance metrics such as encryption/decryption speed, key size, memory usage, and network latency.

Here is a brief comparison of the proposed algorithm with other well-known encryption algorithms:

1. *AES (Advanced Encryption Standard):* AES is a popular symmetric key encryption algorithm that offers high-speed encryption and decryption with low memory usage. However, AES has limitations in terms of access control and is susceptible to brute-force attacks.
2. *RSA (Rivest-Shamir-Adleman):* RSA is a widely used asymmetric encryption algorithm that offers high security and access control. However, RSA is slower than symmetric key encryption algorithms and requires larger key sizes for the same level of security [25].
3. *Elliptic Curve Cryptography (ECC):* ECC is another asymmetric encryption algorithm that offers high security with smaller key sizes than RSA. However, ECC is relatively new and less widely used, which may pose challenges to interoperability [26].
4. *Blowfish:* Blowfish is a symmetric key encryption algorithm that offers fast encryption and decryption with small key sizes. However, Blowfish is vulnerable to certain attacks and is not recommended for use in new applications [22].

Compared to these algorithms, the proposed algorithm offers fine-grained access control, high security, and protection against various types of attacks. However, it may require more computational resources and memory than some symmetric key encryption algorithms like AES.

In terms of network latency, the use of a proxy server for re-encryption may introduce some additional delays, but this can be mitigated by optimizing the proxy server's performance and minimizing the number of re-encryption operations.

Overall, the choice of encryption algorithm depends on the specific requirements of the application, including security, access control, performance, and interoperability. The proposed algorithm offers a unique combination of features that may be well-suited for certain applications, but it should be evaluated against other encryption algorithms to determine the best fit for the given scenario.

To perform a graphical comparison study of the proposed algorithm with other encryption algorithms, we can create a chart that compares their Encryption/Decryption speed, Key size, Memory usage, and Network latency **(Table 1)**.

Table 1. Comparison studies of different algorithms

Algorithm	Encryption /Decryption Speed	Key Size	Memory Usage	Network Latency
Proposed	Fast	Small	Moderate	Low
AES	Fast	Large	Low	Low
RSA	Slow	Large	Low	Low
Blowfish	Fast	Small	Low	Low
ChaCha20	Fast	Small	Low	Low

The proposed algorithm combines fine-grained access control, AES encryption, proxy server re-encryption, Honey encryption, and NTRU. It has a fast encryption/decryption speed, small key size, moderate memory usage, and low network latency.

AES is a widely used encryption algorithm and has a fast encryption/decryption speed but requires a large key size and low memory usage. RSA has a slow encryption/decryption speed and requires a large key size but has low memory usage. Blowfish has a fast encryption/decryption speed, small key size, and low memory usage. ChaCha20 has a fast encryption/decryption speed, a small key size, and low memory usage.

Overall, the proposed algorithm has a good balance of encryption/decryption speed, key size, memory usage, and network latency, making it a viable option for secure data communication.

Conclusion

The proposed method appears to be a multi-layered approach for securing data in the cloud, which includes AES encryption, re-encryption with a proxy server, Honey encryption, and NTRU. This combination of encryption techniques can provide additional layers of security to protect the data from intruders. The use of Honey encryption is an interesting approach to protect encrypted data by creating fake or decoy data that is similar to real data. When an attacker attempts to decrypt the data with a guessed password, they will be given fake data instead of real data. NTRU is a public-key encryption algorithm that can be used to securely transmit the password needed to extract the original data. This can help to prevent unauthorized access to the data, even if an attacker has access to the encrypted file. Overall, the proposed encryption method algorithm appears to be a robust approach to securing data in the cloud. However, it is important to note that no encryption method is foolproof, and there may be ways for attackers to bypass the security measures. Therefore, it is important to continuously monitor and update security measures to ensure that the data remains secure.

Future research can explore other security measures that can address different types of attacks and vulnerabilities. Additionally, the proposed method can be tested and evaluated in different cloud environments to determine its effectiveness and efficiency.

ACKNOWLEDGMENTS: Authors gratefully acknowledge to CSS Department of Brainware University, Kolkata, West Bengal, India for providing lab and related facilities for the research.
CONFLICT OF INTEREST: None
FINANCIAL SUPPORT: None
ETHICS STATEMENT: None

References

1. Roy S, Sarddar D. The role of a cloud of things in smart cities. Int J Comput Sci Inf Secur. 2016;14(1):683-98.
2. Asfahani A. The effect of organizational citizenship behavior on counterproductive work behavior: A moderated mediation model. J Organ Behav Res. 2022;7(2):143-60.
3. Çakar S, Özyer K, Azizoglu Ö. The mediating role of emotional labor in the impact of organizational climate on burnout. J Organ Behav Res. 2022;7(1):1-13.
4. Bansal B, Jenipher VN, Jain R, Dilip R, Kumbhkar M, Pramanik S, et al. Big data architecture for network security. Cyber Secur Netw Secur. 2022:233-67.
5. Kryukova EM, Khetagurova VS, Ilyin VA, Chizhikova VV, Kosoplechev AV. Forming students' environmental culture: modern educational approaches and technologies. J Adv Pharm Educ Res. 2021;11(2):113-8.
6. Sadovnikova N, Lebedinskaya O, Bezrukov A, Davletshina L. The correlation between residential property prices and urban quality indicators. J Adv Pharm Educ Res. 2022;12(2):98-103.
7. El-Gamal F, Najm F, Najm N, Aljeddawi J. Visual display terminals health impact during COVID 19 pandemic on the population in Jeddah, Saudi Arabia. Entomol Appl Sci Lett. 2021;8(2):91-9.
8. Mokrova LP, Borodina MA, Viktorovich V, Goncharov SA, Kepa YN. Prospects for Using Blockchain Technology in Healthcare: Supply Chain Management. Entomol Appl Sci Lett. 2021;8(2):71-7.
9. Zagade H, Varma S, Suragimath G, Zope S. Knowledge, awareness, and practices of oral health for debilitated patients, among nursing staff of krishna hospital. Int J Pharm Res Allied Sci. 2022;11(2):73-80.
10. Sahana S, Bose R, Sarddar D. Harnessing RAID mechanism for enhancement of data storage and security on the cloud. Braz J Sci Technol. 2016;3:1-3.
11. Bose R, Chakraborty S, Roy S. Explaining the workings principle of cloud-based multi-factor authentication architecture on banking sectors. In2019 Amity International Conference on Artificial Intelligence (AICAI) 2019 Feb 4 (pp. 764-768). IEEE.
12. Abdullah AM. Advanced encryption standard (AES) algorithm to encrypt and decrypt data. Crypto Netw Secur. 2017;16:1-1.
13. Moe KS, Win T. Improved hashing and honey-based stronger password prevention against brute force attack. In2017 International

Symposium on Electronics and Smart Devices (ISESD) 2017 Oct 17 (pp. 1-5). IEEE.

14. Hoffstein J, Pipher J, Silverman JH. NTRU: A ring-based public key cryptosystem. InAlgorithmic Number Theory: Third International Symposium, ANTS-III Portland, Oregon, USA, June 21–25, 1998 Proceedings 2006 May 24 (pp. 267-288). Berlin, Heidelberg: Springer Berlin Heidelberg.
15. Muttaqin K, Rahmadoni J. Analysis and design of file security system AES (advanced encryption standard) cryptography based. J Appl Eng Technol Sci. 2020;1(2):113-23.
16. Kamal A, Ahmad K, Hassan R, Khalim K. NTRU Algorithm: Nth Degree truncated polynomial ring units. InFunctional Encryption 2021 Jun 13 (pp. 103-115). Cham: Springer International Publishing.
17. Yasser YA, Sadiq AT, AlHamdani W. A Proposed harmony search algorithm for honeyword generation. Adv Hum-Comput Interact. 2022;2022.
18. Yasser YA, Sadiq AT, AlHamdani W. A scrutiny of honeyword generation methods: Remarks on strengths and weaknesses points. Cybern Inf Technol. 2022;22(2):3-25.
19. Zhang Y, Deng RH, Xu S, Sun J, Li Q, Zheng D. Attribute-based encryption for cloud computing access control: A survey. ACM Comput Surv. 2020;53(4):1-41.
20. Rawal BS, Manogaran G, Hamdi M. Multi-tier stack of blockchain with proxy re-encryption method scheme on the Internet of things platform. ACM Trans Internet Technol. 2021;22(2):1-20.
21. Wang P, Yang LT, Li J, Chen J, Hu S. Data fusion in cyber-physical-social systems: State-of-the-art and perspectives. Inf Fusion. 2019;51:42-57.
22. Sutradhar S, Karforma S, Bose R, Roy S. A dynamic step-wise tiny encryption algorithm with fruit fly optimization for quality of service improvement in healthcare. Healthcare Anal. 2023;3:100177.
23. Chatterjee P, Bose R, Banerjee S, Roy S. Enhancing data security of cloud-based LMS. Wirel Pers Commun. 2023;130(2):1123-39.
24. Dutta A, Bose R, Kumar Chakraborty S, Roy S. A security provocation in cloud-based computing. pattern recognition and data analysis with applications. Singapore: Springer Nature Singapore, 2022;343-55.
25. Yudistira R. AES (Advanced Encryption Standard) and RSA (Rivest–Shamir–Adleman) Encryption on Digital Signature Document: A Literature Review. Int J Inf Technol Bus. 2020;2(2):26-9.
26. Bos JW, Halderman JA, Heninger N, Moore J, Naehrig M, Wustrow E. Elliptic curve cryptography in practice. InFinancial Cryptography and Data Security: 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers 18 2014 (pp. 157-175). Springer Berlin Heidelberg.