Data Encryption Tool with dynamic and secure key management (AES, RSA) using graph

Somchart Fugkeaw, Arnon Sennual, Rathsiri Chintaworn, Weerawat Puengpipattrakul
School of ICT, Sirindhorn International Institute of Technology, Thammasat University, Pathum Thani, Thailand
somchart@siit.tu.ac.th, asennual.ss@gmail.com, rathsiri.chi@gmail.com, willy.pueng@gmail.com

***Abstract*— Cloud-based data encryption is a growing field of research, as more and more data are being stored and processed in the cloud. However, cloud-based data encryption also introduces new security challenges, such as key management. In this paper, we present a cloud-based data encryption tool with dynamic and secure key management. The system uses both symmetric and asymmetric keys to encrypt data and manage the ownership of the keys. With the dynamic approach implemented into the system, users will have a dynamic and secure data encryption and key management tool. The graph model is proposed to manage the data and key within the system. Finally, the performance of the system is evaluated.**

Keywords*— *Cloud-based data encryption, Dynamic key management, Secure key management, Graph model

I. Introduction

Data security has become a paramount concern in today's digital age. With the rise of cyber threats, it has become more crucial than ever. Cybercriminals use a variety of tactics to gain access to sensitive information, such as social engineering, malware, and phishing attacks. It is essential to employ robust data encryption tools that can protect against unauthorized access and ensure the confidentiality and integrity of the data.

A cloud-based data encryption application that utilizes advanced encryption standards, such as AES and RSA, can provide a highly secure solution for data protection. Such an application can be accessed from anywhere, making it a convenient option for remote teams or users who need to access sensitive data on the go. AES (Advanced Encryption Standard) is a symmetric encryption algorithm that is widely used for data protection. It uses a single key to encrypt and decrypt data, making it an efficient and secure method for protecting sensitive information. AES is a block cipher, which encrypts data in fixed-size blocks. It is regarded as one of the most secure encryption algorithms available today. RSA is an asymmetric encryption algorithm that uses a pair of keys, one public and one private, to encrypt and decrypt data. The public key is used to encrypt data, while the private key is used to decrypt it. RSA is extensively used for secure data communication, digital signatures, and secure key exchange. RSA is based on the difficulty of factoring large prime numbers, making it a secure method for protecting sensitive information.

In addition to encryption, key management is a crucial component of data security by using a graph database system like Neo4j to manage keys. Neo4j is a graph database system that is designed to manage and store graph data efficiently. It is ideal for applications that require complex data modeling and relationship management, such as social networks, recommendation engines, and fraud detection systems. Neo4j enables users to model their data as nodes and relationships, which can be queried and analyzed using a query language called Cypher. For key management for data encryption, Neo4j can provide a powerful solution for tracking and managing encryption keys and their relationships. The application can provide a more efficient and secure way to track and manage encryption keys. The graph database also enables more sophisticated key management techniques, such as hierarchical key management, that can be used to enforce fine-grained access control policies. Hosting the application and database on Google Cloud offers numerous benefits, including automatic scaling, high availability, and disaster recovery capabilities. The cloud platform's robust security features, such as data encryption, access controls, and compliance certifications, also provide an additional layer of protection for sensitive data.

In this paper, we present a cloud-based data encryption tool that employs AES and RSA encryption algorithms with dynamic and secure key management using Neo4j on Google Cloud can provide a highly secured and efficient solution for data protection. This technology stack has broad applications across various industries and can be used to enhance the security of everything from personal data to corporate secrets. With the ever-increasing threat of cyber-attacks, such a tool can help individuals and organizations protect their sensitive data from unauthorized access and ensure its confidentiality and integrity.

II. Propose Schema

A. System model

Figure 1 illustrate the system model which consists of four entities.

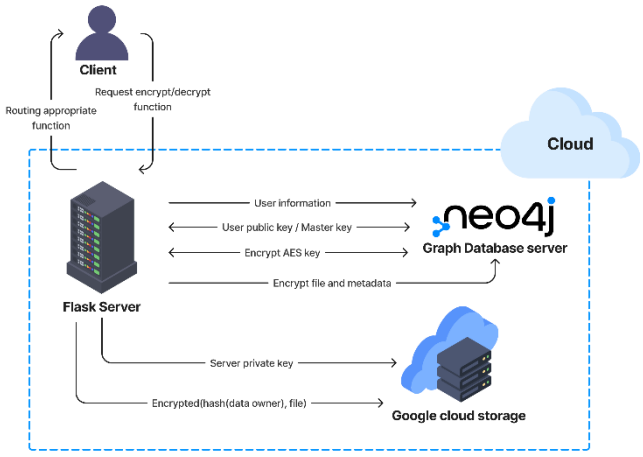

Fig. 1. System model

Our system consists of four entities as follows.

1. Clients are users who encrypt data by using either symmetric or asymmetric encryption and decryption. Only clients having a match session key can access the encrypted data. Clients also can generate RSA key pairs.
2. Flask server is a core server that works by receiving client requests, routing them to the appropriate handler function, and returning the response to the Neo4j and GCS.
3. Google Cloud Storage (GCS) stores encrypted files which are both the folder of hashing function of data owner username and file, also stores private key of server.
4. Graph database storage (Neo4j) stores public keys of users and public keys of server, metadata of encrypted files, encrypted AES key for key distributed, and users' information details.

B. Data encryption process

Our system process consists of three major phases including System setup, Symmetric or Asymmetric encryption selection, and Data transmission to cloud.

Phase 1: System Setup
This phase consists of one algorithm: User Registration

(1) User Registration

In this phase, users are registered to the data encryption system using their valid credentials. Then, the system issues the username and login credentials to the users.

Phase 2: Symmetric or Asymmetric encryption selection
In our model, there are two encryption types for selecting to encrypt data.

a. Symmetric encryption process

This process is done by clients. It consists of two steps including key and IV generation by PRNG and Symmetric data encryption by AES.

(1) Key and IV generation
CreateKeyAndIV PRNG → (Key, IV)

The algorithm generates both key and IV as cryptographically strong random values 32 bytes and 16 bytes respectively by using the concept of pseudo-random number generator algorithm (PRNG).

(2) Symmetric Encryption
AESencrypt(Data, Key, IV) → CT

This process performs symmetric encryption by AES-CBC on the data with key and IV by using a Frog Cryptographic tool. It produces ciphertext (CT).

b. Asymmetric encryption process

This process is happened after the AES data encryption process and done by clients. It sends to phase 3.

Phase 3: Data transmission to cloud

In this phase, the system prepares keys and IV for key distribution and future access. It consists of five algorithms: Server key encryption, Encrypt kay and IV, Encryption data, Append the data, IV, and key into the form, and Upload to GCS and Neo4j.

(1) Server key encryption

The public server key is requested from the server by the function *requested_public_key*. Then, the server responds from the route the client requested in JSON format which is then stored as a variable. From the server side, the server responds from the route that the client requests.

(2) Encrypt kay and IV
RSAencryptData((iv, pbkey), (key, pbkey)) → (Encrypt_iv, Encrypt_key)

The function *RSAencryptData()* is called to encrypt the iv and the AES session key using a server key. It uses the public server key that is requested from the server to encrypt.

(3) Encryption data
RSAencryptData(data, pbkey) → CT

The *RSAencryptData()* utilizes RSA-OAEP algorithm to encrypt the data. Optimal Asymmetric Encryption Padding (OAEP) is a widely used padding scheme that is often paired with RSA encryption. By introducing an element of randomness, it can transform a deterministic encryption scheme (such as traditional RSA) into a probabilistic one. In addition, OAEP provides protection against partial decryption of ciphertext and other types of information leakage. It accomplishes this by ensuring that any attempts to recover even a portion of the plaintext without being able to invert the trapdoor one-way permutation f will be unsuccessful.

(4) Append the data, IV, and key into the form

The process of uploading AES-encrypted data, server-key-encrypted IV, and AES key involves concatenating these elements together into a single package, which is then uploaded to its destination via the */upload_AESencrypted_file* route. This concatenation process ensures that the three components are treated as a single entity, rather than as separate and potentially disjointed pieces of information. This approach to data transfer provides an additional layer of security by keeping the different components of the encrypted data together and ensuring their integrity during transmission.

(5) Upload to GCS and Neo4j

The process entails uploading the appended data to Google Cloud Storage (GCS) for secure storage. Subsequently, the metadata is uploaded to Neo4j for the purpose of establishing a relational database. During this process, the server is capable of identifying the username associated with the active session. Upon receiving a file request, the server proceeds to hash the username to create a folder name and subsequently uploads the file to GCS. The server then executes an integrity check by hashing the file content before creating the metadata node in Neo4j. The final step involves the creation of a session key node in Neo4j.

Phase 4: Share encrypted file to others

This phase consists of one algorithm: Key distribution.

(1) Key distribution

The system encrypts AES key, IV of data with other user public key, then store the encrypted key, IV to Neo4j node with SHARED_TO relation. The system creates metadata from the encrypted file and stores its on Neo4j database node.

Algorithm: Key distribution

Input: Username
Output: Session key node on Neo4j
query = (

"MATCH (owner:User{username:$username}) " #Owmer of session key

"MATCH (shared:User{username:$shared_to}) " #User that shared to

"MATCH (d:data{file_name:$filename}) " #Data that shared to

#in case that not shared to anyone shared_to = Owner

"MERGE (s:SessionKey {owner:$username,issuance:timestamp(),value:$session_key, iv:$iv,shared_to:$shared_to}) " #create session key node

"MERGE (owner)-[:OWNED]->(s) " #create relation between owner and session key

"MERGE (s)-[:SHARED_TO]->(shared) " #create relation between session key and shared_to

"MERGE (s)-[:ENCRYPT]->(d) " #create relation between session key and data

"RETURN s")

C. Key management

(1) Key Regeneration

The client is responsible for generating the encryption key. Once the key is generated, the user can download the corresponding private key in PEM (Privacy Enhanced Mail) file format. The public key, on the other hand, is transmitted to the server to create a Neo4j public key node for storing the public key information. Notably, each user is assigned a unique public key for ownership purposes.

Privacy-Enhanced Mail (PEM) is a file format for storing and transmitting encrypted email messages and other encrypted data securely over the internet. PEM is designed to protect the confidentiality of sensitive information by encrypting data in a way that makes it difficult for unauthorized parties to access or read the information. It uses a combination of public-key encryption and symmetric-key encryption to provide security for email messages and other types of data. The data is first encrypted using a symmetric encryption algorithm, such as AES, and a random symmetric key is generated for each message.

Then, the symmetric key is encrypted using the recipient's public key and sent along with the encrypted message. The recipient can then use their private key to decrypt the symmetric key, which can then be used to decrypt the message. By using a combination of symmetric and asymmetric encryption, PEM provides a high level of security while minimizing the computational overhead associated with asymmetric encryption alone. PEM is widely used for secure email communication and is also used in other applications where secure transmission of data is required, such as digital certificates and secure web browsing.

(2) Key Rotation

Key rotation is the process of generating a new set of keys to replace the compromised, lost or stolen key with a new set of keys. It is worth noting that in this particular system, this process does not affect the data that is stored on our cloud, since the users' session keys are encrypted using the server's public key so the users can still retrieve the data stored in the cloud without using the old set of keys to access.

This code below shows the algorithmic details of our key management process.

Algorithm: Count and update user public key

Input: Username, User public key
Output: User public key node on Neo4j

```
def rsapbkeysubmit():
    data = request.get_json()
    username = data['username']
    keySize = data['keySize']
    key_value = data['k_value']
    user_detail = neo4japp.user_detail(username)['u']
    id = user_detail['id']
        if (id != None and neo4japp.multiple_key_check
(username) < 1):
            neo4japp.submit_pb_rsa(username,id,keySize,key
_value)
            return render_template('profile.html')
        else:
            neo4japp.update_pb_key(username,keySize,key_
value)
            return render_template('profile.html')
```

D. Graph database storage model

Graph database storage model is a Neo4j graph structure that structurally correlates Master key (MK) node, User (U) node, User public key (PK) node, Session key (SK) node, and Data (D) node for supporting dynamic key management and query performance. In our system, we used a graph database to store the nodes data.

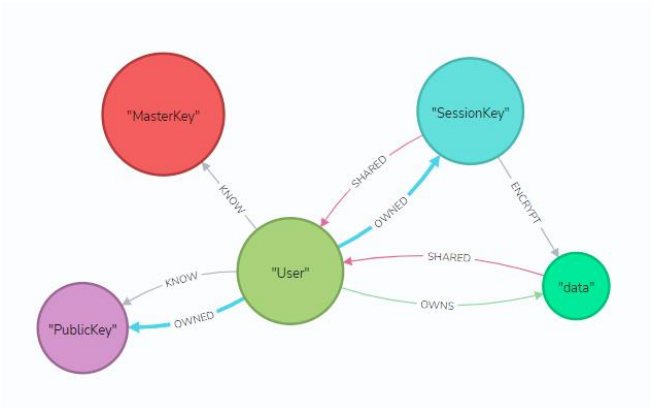

Fig. 2. Neo4j model

As shown in Fig. 2, the relationship between nodes is expressed through the edge that represents the data necessary for dynamic key management and improved query performance. The description of each node and edge is described as follows.

User node represents the user related attributes information. The process of checking the authenticity of accessing encrypted data. Only users who have a matched key with the corresponding encrypted data are able to decrypt the ciphertext.

User public key node contains information of the RSA key related to the RSA encryption. User public keys can be rotated by regenerating a new user RSA key pair without affecting encrypted data from user to server.

Master key node contains information about the RSA public key of the server. Master keys can be rotated by regenerating a new user RSA key pair without affecting encrypted data from server to user.

Session key node contains the information of the AES key that is encrypted by RSA related to the encrypted data located in the GCS.

Data node contains the information of the encrypted file including metadata to define the location of data on GCS and its hash value for checking integrity.

Table 1 presents the details about the relationship among the nodes presented in the Neo4j model.

TABLE 1. RELATIONSHIP BETWEEN NODES

Edge (Node : Node)	Relationship	Meaning
User node and Master key node	Known	Every User knows the server Master key.
User node and Session key node	Owned	Every user has their own session key from AES encryption encrypted by using the user public key that they want to share to other users.
User node and Data node	Owned	Data has a user as an owner and also shares to other data users.
User node and User public key node	Owned	Every User has their own key and knows the other user's key.
Session key node and Data node	Encrypted	Each encrypted data has its own Session key.

III. EXPERIMENT DETAILS

In this paper, we conducted the experiment to measure the query performance of graph database model structure (Neo4j) to demonstrate the efficiency of key management of cloud-based data encryption tool.

A. Graph database structure (Neo4j)

In Neo4j, "db hit" refers to the number of times the database is accessed to fulfill a particular query. Each time the database is accessed to retrieve or modify data, it counts as a db hit. For example, the first function is used to reveal data and session keys that the user has authorized to access.

```
PROFILE
MATCH (u:User)-[o:OWNED]-(sk:SessionKey)-[r:ENCRYPT]-(d:data)
WHERE d.owner = "Test1225"
return sk,r,d,u,o
```

This function produces results at Neo4j by 105 db hits which means that the database had to be accessed and traversed 105 times in order to fulfill the query to retrieve metadata and session key.

The second function is used to retrieve all public key node that user can distribute key to others.

```
PROFILE
MATCH (u:User)-[r:OWNED]-(k:PublicKey)
RETURN u,
```

The Query produces all public key that user known with 56 db hits which mean that to retrieve all public key 56 time traversal between node

Furthermore, all of the functions in our Neo4j graph database design produce under 130 db hits. The performance of the query of our Neo4j graph database design is efficient for our project because it uses less communication cost to query in data side database.

B. Key management

a. Dynamic Key Management

The client is tasked with generating the encryption key, which, once generated, can be used to download the corresponding private key in PEM (Privacy Enhanced Mail) file format. Additionally, the public key is sent to the server, where it is used to create a Neo4j public key node that stores public key information. It's important to note that each user is assigned a unique public key to ensure ownership of the data.

In the event of a compromised, lost, or stolen key, the system uses a key rotation process that generates a new set of keys to replace the affected key. It is worth noting that this process does not have any effect on the data stored on our cloud, as the users' session keys are encrypted using the server's public key. This allows users to retrieve their data from the cloud without using the old set of keys to access it. By employing these security measures, our system provides a robust defense against potential cyber threats and ensures the confidentiality, integrity, and availability of sensitive information.

b. Security

By leveraging the robust security features of GCS and Neo4j, our system provides a high level of protection against potential attackers. Our encryption protocols, combined with the advanced security measures implemented by the cloud service providers, create a multi-layered approach to data protection that ensures its confidentiality, integrity, and availability. Together, these measures provide a strong defense against unauthorized access, data breaches, and other

types of cyber threats that could compromise the security of our system.

IV. Conclusion

We have proposed this cloud-based data encryption and dynamic key management application to system deliver a secure user's key and data. The system uses both symmetric and asymmetric keys to encrypt data and manage the ownership of the keys. With the dynamic approach implemented into the system, users will have a dynamic and secure data encryption and key management tool. The graph model is proposed to manage the data and key within the system. Finally, the performance of the system is evaluated.

References

[1] Neo4j. (2023, April 24). Cypher Manual. Retrieved from https://neo4j.com/docs/cypher-manual/current/introduction/

[2] Flask-Session. (2023, April 24). Flask-Session. Retrieved from https://flask-session.readthedocs.io/en/latest/#

[3] PyCryptodome. (2023, April 24). PyCryptodome. Retrieved from https://www.pycryptodome.org/

[4] Digital Bazaar Forge. (2023, April 24). Digital Bazaar Forge. Retrieved from https://digitalbazaar.github.io/forge/

[5] Li, J., & Zhang, Y. (2021). Hybrid encryption algorithm based on AES and RSA in file encryption. In 2021 2nd International Conference on Artificial Intelligence and Information Processing (ICAIP) (pp. 1-5). Atlantis Press.

[6] M. Bellare, P. Rogaway. Optimal Asymmetric Encryption -- How to encrypt with RSA. Extended abstract in Advances in Cryptology - Eurocrypt '94 Proceedings, Lecture Notes in Computer Science Vol. 950, A. De Santis ed, Springer-Verlag, 1995.

[7] Mozilla Developer Network. (2023, April 24). Web Crypto API. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API

[8] Mozilla Developer Network. (2023, April 24). XMLHttpRequest. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

[9] Johnson, Mike (1995-10-01). "Cryptology in Cyberspace". Cryptologia. 19 (4): 392–396. doi:10.1080/0161-119591884042. ISSN 0161-1194. S2CID 41770450

[10] Kent, S. (1993). "Internet Privacy Enhanced Mail". Communications of the ACM. 36 (8): 48–60. doi:10.1145/163381.163390. S2CID 15759913