Network Security in Cloud-based File Hosting Service

By Alex Xiao Cai - axiaocai@uoregon.edu & Tien Dinh - tvd@uoregon.edu

Abstract

It is noticeable that the Internet continues to grow at a rapid pace. As one of the most important outcomes from this progress, Cloud Computing emerged to solve many challenges such as hardware limitations, storage limitations, cost limitation, etc. However, as the Internet keeps growing, attacks from cyber-criminals are becoming more sophisticated. Thus, in order to protect valuable data, cloud-base technologies need to consider many security aspects. Then, They need to apply countermeasures or defensive mechanisms against these attacks. Thus, in order to understand the security aspects of cloud-based technologies, we studied Google Drive, Microsoft OneDrive, and DropBox due to their popularity, free-to-use, easy-to-use, and accessibility. Then, we developed our own cloud-based file hosting application which features include, but not limited to, password authentication, file encryption, and access control.

Keyword: File-based cloud hosting, network security, password encryption, file encryption, hashing, SHA, AES, RSA, access control.

1. Introduction

With the introduction of the Internet, the development of several useful applications came along. One of these is the ability to share resources across several platforms in a real-time manner. This will be later known as Cloud Computing. Nowadays, cloud-based technologies are on a rapid rise as they provide several advantages. One of these advantages is cost savings as the user or client does not have to buy expensive equipment to host their applications. Another advantage is that there is virtually an unlimited storage capacity as long as the user is willing to pay for that storage. Additionally, cloud-based technology offers availability meaning that the user can access to the application anywhere at any time, and accessibility meaning that the user can control the access and privileges of their files. However, because sensitive and critical information is being stored at one place, security must be strong at all times to prevent malicious agents or cyber-criminals from gaining this valuable information.

As a countermeasure against attacks from cyber-criminals, several cloud services deploys several security mechanisms. One of these mechanisms is authentication. Authentication is the process of proving or showing that the user claims to be by providing information that only the user knows such as security questions, or providing information that the user has such as certificates, or providing information that the user is such as biometrics. Another defense mechanism is file encryption. File encryption refers to the method of converting the file's information into a secret code which only authorized entities can decipher. In addition to the previous, the cloud services can also provide access control to ensure the proper management of the data. Although there are more defensive implementations, the previous implementation serves as a base for cloud-based services to improve and become more secure.

In order to understand more the security aspects of cloud-based services, we proposed to develop our own file hosting service through a web application. Additionally, we will explain the security aspects in this application in detail that will be implemented. For this application, these security aspects are password authentication, file encryption, and access control.

2. Related Work

As mentioned before, our application is not new, but an attempt to study the security aspects of other cloud-base services. As there are many cloud-based file hosting services in existence, we selected the most popular such as Google Drive, Microsoft OneDrive, and DropBox. Additionally, these platforms were selected because of its reputation, reliability, and design.

Google Drive was launched on April 24, 2012. Google Drive allows users to store files in the cloud, synchronize files across devices, and share files. The access of the shared files are managed by the owner. Additionally, the files are encrypted with an AES256 or AES128 encryption. On top of this security feature, it requires an authenticated Google account in order to access or modify these files.

Microsoft OneDrive was launched in August, 2007. Same as Google Drive, Microsoft OneDrive allows users to store files in the cloud, synchronize, and share files. In addition, Microsoft OneDrive provides access control to the user. However, the most noticeable difference between these two platforms is that Microsoft OneDrive has a personal vault where encryption can be done locally. This is done by syncing the Personal Vault files to a BitLocker-encrypted area of the local hard drive.

DropBox founded in 2007 is another cloud-based file hosting service. Same as the previous two, DropBox offers storage, synchronization, access control, and file sharing. Dropbox files are encrypted using 256-bit Advanced Encryption Standard (AES) [1]. For the data transmission, it uses Secure Sockets Layer (SSL)/Transport Layer Security (TLS) to protect data in transit between Dropbox apps and their servers. Additionally, the SSL/TLS creates a secure tunnel protected by 128-bit or higher Advanced Encryption Standard (AES) encryption.

3. Password Authentication

Password authentication is the process of a user authenticating through an unique identification and key. This unique identification and key is then checked against the database stored credentials. For our application, we utilize Bcrypt encryption for password authentication. Bcrypt [2] is a password-hashing function that incorporates a salt to protect against rainbow table attacks. Rainbow table attacks consist of an attacker stealing the password hashes and, using a rainbow table. Then, the attacker is able to decrypt the passwords of every user of the application. Additionally, it is adaptive meaning that over time, the iteration count can be increased to make it slower and making it resistant to brute-force search attacks which consist of an attacker submitting many passwords or passphrases with the hope of eventually guessing correctly. The components of a Bcrypt encryption consist of an hash algorithm identifier, input cost, input salt, and the hash as seen below.

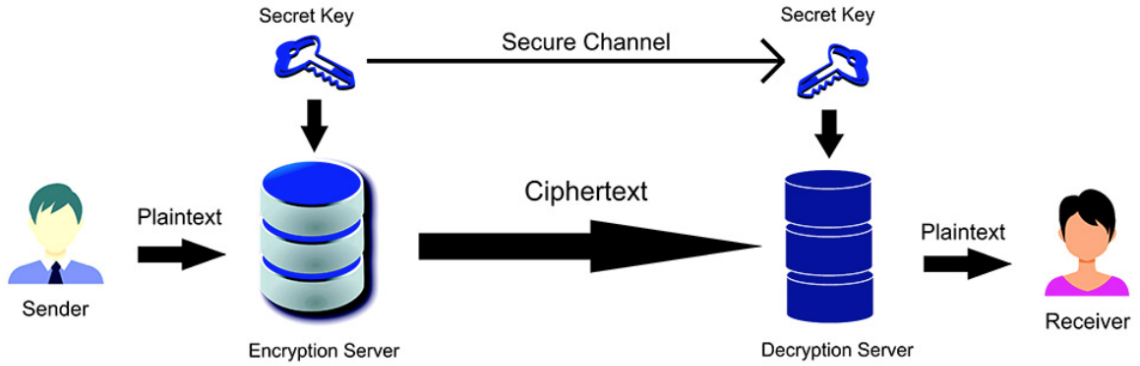
```
$2a$12$R9h/cIPz0gi.URNNX3kh2OPST9/PgBkqquzi.Ss7KIUgO2t0jWMUW
\__/\/ \____________________/\_____________________________/
Alg Cost      Salt                        Hash
```

For the Bcrypt encryption the hash algorithm identifier is 2a which refers to Blowfish-based crypt ('bcrypt'). The cost input refers to the number of rounds of encryption done by 2 to the power of that cost input. In the above case, the cost is 12 meaning that 2^{12}, 4096, rounds of encryption are being done. This ensures brute-force protection. The salt is a random generated data for the application of one-way functions. This is the countermeasure for rainbow table attacks. The hash is a unique string generated by the encryption process. The pseudocode for bcrypt encryption algorithm can be seen below.

```
Function bcrypt
   Input:
      cost:     Number (4..31)
log2(Iterations). e.g. 12 ==> 2^12 = 4,096 iterations
      salt:     array of Bytes (16 bytes)           random salt
      password: array of Bytes (1..72 bytes)        UTF-8 encoded
password
   Output:
      hash:     array of Bytes (24 bytes)

   //Initialize Blowfish state with expensive key setup algorithm
   //P: array of 18 subkeys (UInt32[18])
   //S: Four substitution boxes (S-boxes), S0...S3. Each S-box is
1,024 bytes (UInt32[256])
   P, S ← EksBlowfishSetup(cost, salt, password)

   //Repeatedly encrypt the text "OrpheanBeholderScryDoubt" 64 times
   ctext ← "OrpheanBeholderScryDoubt"  //24 bytes ==> three 64-bit
blocks
   repeat (64)
      ctext ← EncryptECB(P, S, ctext) //encrypt using standard
Blowfish in ECB mode

   //24-byte ctext is resulting password hash
   return Concatenate(cost, salt, ctext)
```

4. File Encryption

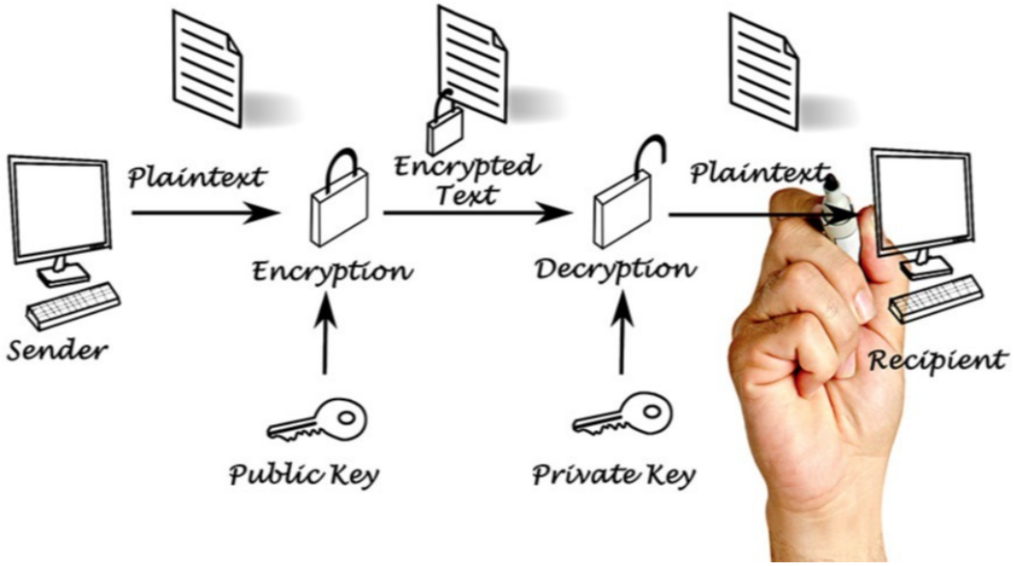
A file encryption is a set of security algorithms used to turn raw data into cipher data using encryption key(s). The encryption transforms the data into a completely different format from the raw data, with different length, random characters and unreadable for humans. This cipher data is delivered safely to the receiver. On the receiving side, the receiver will take the data, using one of the keys (private key or shared key depending on the algorithm) that was agreed with the sender to decrypt the cipher data into understandable material. The encryption process usually takes longer than just uploading raw data, but it ensures the confidentiality of the data during transmission. File encryption is different from password encryption because files are usually and are in many forms, not only string text. We use Node.js crypto [3] package to implement our own encryption algorithm. There are two main encryption algorithms for file encryption that are categorized as symmetric encryption and asymmetric encryption.



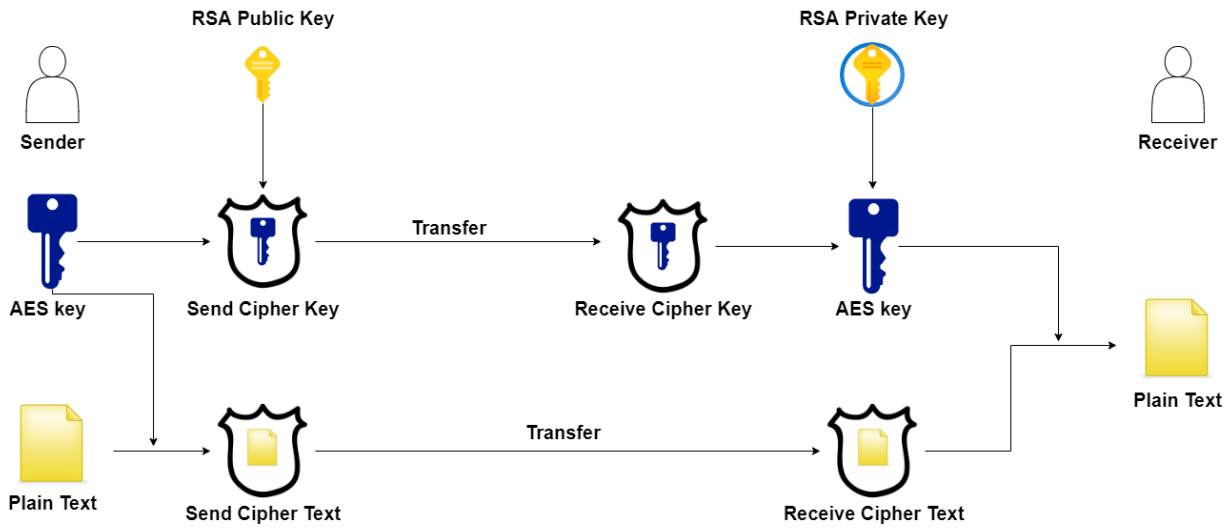
Symmetric Cryptography

AES (Advanced Encryption Standard) [1] was initially referred to as Rijndael, is a symmetric encryption algorithm. It was developed originally from the National Institute of Standards and Technology for government usage only. Ever since the internet exploded and became commercial, AES is known worldwide. This is a symmetric encryption because the sender and the receiver side would share a secret key and use it to encrypt and decrypt the file at the same time. The AES algorithm has three cipher blocks: AES-128, AES-192 and AES-256. The number following AES is the number of bits of the key used for encryption. Therefore, the longer the key bit, the more resistance to attack the algorithm is. However, even AES-128 is resistant to attack by standard, offer speed, thus is a prefered choice in many occasions. AES-256 is the more difficult algorithm to crack, very resistant to brute-force attack and more often used than AES-192. During the process, several steps are performed in one encryption round. The first step is initialize round key addition, where each byte of the state is combined with a byte of round key using bitwise XOR. During the next rounds, the algorithm performs SubBytes, ShiftRows and MixColumns steps. SubBytes is a non-linear substitution step where each byte is replaced with another byte according to a predefined lookup table. ShiftRows is a transition step where the last three rows of the

current state are shifted periodically by a number of steps. MixColumns is a linear mixing operation on the columns of the current state combining with the four bytes in each column. The process is repeated several times. Each repeat is called a round and encrypted with the round keys.



RSA [4] was named after its inventors Rivest, Shamir, and Adleman. RSA is the modern asymmetric encryption used worldwide due to the secure nature of public and private key pairs. RSA uses two keys, the first one being the public key that is for encryption and the second one is the private key used for decryption, hence making the algorithm asymmetric. The public key is made known to everyone while the private key is only known to the owner. The RSA algorithm is derived based on factorial function in math. RSA first generates encryption and decryption key pairs by picking two large prime numbers x and y in secret. The algorithm then chooses any integer e in which $1 < e < (x-1)(y-1)$ (e is the public key exponent). A private exponent k is choose by $k*e = 1 \pmod{(x-1)*(y-1)}$. So we have the public key is made of modulus z and public exponent e, while the private key is made of x, y and the private exponent k. To encrypt the message, the sender first sends the public key (z, e) to the receiver. The sender turns the message into a number g, g is smaller than z. The sender then transforms the original text to cipher text using this expression: $c = g^e \bmod z$ (c represents the cipher text). On the receiving side, the message can be decrypted by using the reverse expression: $g = c^d \bmod z$. The reversible nature of this modulus function is proven by Euler's theorem [5]. Because the private key is never shared with anyone, the RSA algorithm becomes the most secure, unlikely to be cracked algorithm. Reverse engineering on RSA with a large prime key number RSA can take around 300 trillion years, thus making RSA a state-of-the-art encryption algorithm.



Hybrid data encryption

Both AES and RSA have their own weaknesses and they can be combined together to strengthen the encryption algorithm. AES is a standard encryption employed by many applications in the industry. Google Drive used 128-bit AES encryption from the beginning until 2015, where they shifted some user portions to AES-256 encryption. Dropbox used 256-bit AES encryption for every user. AES became a common practice because of the simpleness, fast runtime nature of the algorithm. It can encrypt/decrypt the file within milliseconds, thus ensuring the quality of service for the users. The AES algorithm however is vulnerable in exchanging the secret key. If the AES key is discovered during exchange or is exposed on the client side, the data will become vulnerable. On the other hand, the RSA algorithm is more secure with private keys that are never shared. However, RSA is a more complex algorithm that requires more encryption time. Moreover, RSA can only encrypt the data length that is smaller than the key length. This feature makes RSA useless to encrypting big file data, large messages, thus more fit to encrypting password only. Therefore, our solution is to combine both of the algorithms to make sure that we have RSA protection for our file. The senders first encrypt their file to cipher text by using the AES shared key. The senders then encrypt the AES shared key by using the receiver's RSA public key. The senders will send the cipher text and the cipher key to the other side. On the receiving side, the receivers will use their RSA private key to decrypt the cipher key. The receivers then use the decrypted RSA key to decrypt the ciphertext, thus enabling them to read the text. The file is encrypted during transfer and the RSA private key is always kept private to ensure maximum protection of the data at all times.

5. Access Control

Access control is another important aspect of security. Access control ensures a person has the access to the right content, and only the content that is available to them. There are two methods implemented in our project to ensure that: authentication and file hiearchy.

On the first entry to the webpage, a user must register to be able to access the content. The server takes user email, name and password to allow the entry point to the site. Password is hashed and stored

using Node.js bcrypt package as discussed in section 3. On every page, we make sure to have a checkAuthenticated function to verify that a user is already logged in. This is to protect the website from random entry points an attacker might use to exploit. Node.js passport [6] package is used to support this feature. Once a user is logged out, he/she cannot access the content anymore.

For a standard cloud-based file hosting, we offer public and private share of files. A user upload is defaulted to private upload unless specified otherwise. The file upload is secured by a combined AES and RSA encryption method as described in section 4. The complexity comes with public file sharing. When the user decides to share the file publicly, everyone will be able to read the file, including the server. But how is this achievable when the file transfer is encrypted? Or should we allow the server to read the file? If the server is not allowed to read the file, the problem nature turns into a messenger, not file hosting anymore. In the case a receiving client wants to read a shared file, the sending client needs to do all the encryption every single time. It is better to let the server handle this task, thus enable the server to read the file you upload. Google Drive and Dropbox both can read user's files if they want to. This raises another question: Should we trust the server? One of the security aspects is to trust no one, which is why Google Drive, Dropbox and other platforms still remain the first choice for file hosting service. This comes with a trade-off. There are risks involving the service provider leaking data to a third-party, the government and encryption key vulnerability. However, Google, Dropbox, Microsoft and other well-known companies have built up their credibility from a long history of industrial work, and they offer needed, quality service in this information technology time where storing data on paper or manually is impossible. In the end, when the users sign up for a service, they already agree to the terms and entrust the privacy of data to the service provider.

6. Conclusion

Cloud-based file hosting service is an important service during the information technology and data science era. Security, especially network security is highly needed to ensure the confidentiality, integrity and availability (CIA triad) of the data. The entry to the content needs to be password authenticated with a strong hash algorithm. The file upload needs to be encrypted using standard encryption methods or combined multiple methods to ensure the integrity of the file during file transfer over the internet. Our project employed a hybrid encryption method from both AES and RSA algorithms to further protect the file from attackers. Last but not least, the file needs to be available to the right person and only the right person. This raises the question of whether we should entrust the service provider to be able to read the file we upload. The security aspect is said to trust no one but industry practice proves otherwise, where service providers offer the users desired, fast, quality services with the trade-off of trusting them. This remains true and cloud-based file hosting service providers continue to be an attractive option in this time of exploding information.

Appendix

- Software: Node.js, ejs, npm.
- Important package: bcrypt, crypto, passport

Reference

[1] Daemen, Joan; Rijmen, Vincent. *AES Proposal: Rijndael* (PDF). National Institute of Standards and Technology. p. 1. Archived (PDF). 5 March 2013.

[2] Antonio S. Cardozo, Ben Glow, Van Nguyen, David Trejo, Vincent Côté-Roy, Lloyd Hilaiel, Roman Shtylman, Vadim Graboys, Ben Noordhuis, Nate Rajlich. Sean McArthur, Amitosh S. Mahapatra. *Node.bcrypt.js*. v5.0.1. 2021. https://www.npmjs.com/package/bcrypt

[3] crypto-js. v4.1.1. 2021. https://www.npmjs.com/package/crypto-js

[4] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. *A method for obtaining digital signatures and public key signatures, Communications of the ACM*. Vol. 21, No. 2, Feb. 1978; or reprinted in Secure Communications and Asymmetric Cryptosystems, AAAS Selected Symposium 69, Westview Press, Boulder, CO, 1982.

[5] *Carmichael function*. https://en.wikipedia.org/wiki/Carmichael_function#Carmichael's_theorem. 15 January 2022, at 20:16 (UTC).

[6] Jared Hanson. *Passport*.v0.5.2. https://www.npmjs.com/package/passport