Computers, Materials & Continua
DOI: 10.32604/cmc.2022.026949
Research Article

Tech Science Press

Vulnerability Analysis of MEGA Encryption Mechanism

Qingbing Ji[1,2,*], Zhihong Rao[1,2], Lvlin Ni[2], Wei Zhao[2] and Jing Fu[3]

[1]School of Cybersecurity, Northwestern Polytechnical University, Xi'an, 710072, China
[2]No.30 Institute of CETC, Chengdu, 610041, China
[3]Eberly College of Science, Pennsylvania State University-University Park, PA, 16802, USA
*Corresponding Author: Qingbing Ji. Email: jqbdxy@163.com
Received: 07 January 2022; Accepted: 23 March 2022

Abstract: MEGA is an end-to-end encrypted cloud storage platform controlled by users. Moreover, the communication between MEGA client and server is carried out under the protection of Transport Layer Security (TLS) encryption, it is difficult to intercept the key data packets in the process of MEGA registration, login, file data upload, and download. These characteristics of MEGA have brought great difficulties to its forensics. This paper presents a method to attack MEGA to provide an effective method for MEGA's forensics. By debugging the open-source code of MEGA and analyzing the security white paper published, this paper first clarifies the encryption mechanism of MEGA, including the detailed process of registration, login, and file encryption, studies the encryption mechanism of MEGA from the perspective of protocol analysis, and finds out the vulnerability of MEGA encryption mechanism. On this basis, a method to attack MEGA is proposed, and the secret data stored in the MEGA server can be accessed or downloaded; Finally, the efficiency of the attack method is analyzed, and some suggestions to resist this attack method are put forward.

Keywords: TLS; advanced encryption standard; forensics; protocol analysis; vulnerability

1 Introduction

MEGA is a cloud storage service launched by MEGA limited company. The content data stored on MEGA, such as files, messages, audio and video, are encrypted on the user's client. After encryption, the user uploads the encrypted data to the MEGA platform, but the encryption key of the data will not be directly saved on the platform. Therefore, the access to the data stored on Mega is controlled by the user, not the platform. Even the platform cannot access the data. If other users want to access the data, the user must transmit the encryption key encrypted with the recipient's public key to the recipient.

All encryption related to the security of the user's data is performed only on the user's device. MEGA has released the source code of all client applications [1,2]. Interested third parties can independently verify whether MEGA has the security advertised in the white paper and has no backdoors or accidental vulnerabilities. MEGA supports browser access, which not only lowers barriers to entry but also facilitates the use of other encryption technologies. As of August 20, 2021, the site had 236 million registered users and uploaded more than 107 billion files.

Encryption is a general double-edged sword, on the one hand, it protects the privacy of users, on the other hand, it is used by a small number of users as a tool to engage in illegal activities. As mentioned, MEGA has very good security and its security mechanism, and all communication between the client and the server is protected by TLS encryption from the time it is installed. In 2019, TLS1.2 was revealed that there were some vulnerabilities. Related attacks can be carried out based on man-in-the-middle attacks. With TLS1.3, these attack methods have failed. So far, no fatal defects have been found in the Advanced Encryption Standard (AES) and other algorithms used by MEGA [3,4]. As you can imagine, MEGA's forensics were very difficult. There are very few articles or information about MEGA's encryption protocol analysis, almost none. Here, we analyze MEGA's encryption mechanism, find its vulnerabilities, that is, its public links are encrypted only by password. Because people's brain memory is limited and can only remember 5–7 passwords, most of the passwords set by the same person are similar [5–7]. Based on this, we present a method to attack MEGA that can access or download encrypted data stored on MEGA's server, thus bolstering MEGA's forensics. According to the principle and efficiency of this attack method, we also put forward some protection suggestions, which can resist the attack to a certain extent or reduce the success rate of this attack.

The paper is arranged in 6 sections as follows: In Section 2, by debugging the open-source code of MEGA and analyzing the security white paper published [8], we clarify MEGA's encryption mechanism, including MEGA's registration, login, and file and folder encryption details. In Section 3, we analyze MEGA's security, point out that MEGA is not invulnerable, and propose a method or idea of attack. Section 4 analyzes the security mechanism of file or folder's secure public links in detail, and proposes an attack method against the public links protected by password. Section 5 analyzes the efficiency of the attack and gives some suggestions to resist this attack. Finally, the paper is summarized in Section 6.

2 The Encryption Mechanism of MEGA

We clarify MEGA's encryption mechanism as follows by debugging the open-source code of MEGA and analyzing the security white paper published.

2.1 Registration and Login

2.1.1 The Process to Register MEGA

The process to register MEGA is as follows:

Step1. Enter the password *pwd*.

Step2. The local client uses the client's native CSPRNG to generate a random 128-bits *Master Key(MK)*.

Step3. The Client generates a 128-bits *Client Random Value*.

Step4. The Client computes the *Salt* (=SHA-256 ("mega.nz" || "Padding" || "PP . . . P" || *Client Random Value*)). The string "PP . . . P" is composed of the capital letter "P", and its length is equal to 200 minus the character length of the string "mega. NZ" and the string "Padding".

Step5. Process the Password and generate several keys.

- The client will generate a 256-bits $Derived\ Key(DK)$ using PBKDF2:

  $DK = \text{PBkDF2-HMAC-SHA-512}\,(pwd, Salt,\ 100000,\ 256)\,.$

- The client will take the first half of the DK from left to right as the $Derived\ Encryption\ Key(DEK)$, and use it to encrypt MK by AES:

  $Encrypted\ MK = \text{AES-ECB}\ (DEK, MK)\,.$

  The client will take the remaining half of the DK as the $Derived\ Authentication\ Key(DAK)$ and use it to calculate $Hash\ Authentication\ Key(HAK)$:$HAK = \text{SHA-256}\ (DAK)$.
  Note, the HAK consists of the first 128 bits of the output of SHA-256, and it will be used to authenticate the user's login with the API.

Step6. The client will send the information described in Tab. 1 to register an account.

Table 1: User registration information

Serial number	Information content
1	Full Name (including first name and last name)
2	Email address
3	Client random value
4	Encrypted MK
5	HAK

Step7. The API will use the native CSPRNG to generates a 128-bits random *Email Confirmation Token* (*ECT*) and send a link to confirm:

ConfirmInformationLink=“https://mega.nz/#confirm”||Base64UrlEncode(“ConfirmCode V2”||ECT || Email Address)

Step8. The user clicks the confirmation link in the email, and the client sends the confirmation information back to the API. If the API request is successful, the user will be asked to enter login information such as email address and password again, where email address can be filled in advance.

Step9. If the user logs in successfully for the first time, a set of 2048-bits Rivest-Shamir-Adleman(RSA) key pairs, a set of 256-bits Ed25519 key pairs, and a set of 256-bits Curve25519 key pairs will be generated.

2.1.2 The Process to Register MEGA

The process to register MEGA is as follows:

Step1. The user enters the local client login interface and inputs their *Email Address* and *Password*.

Step2. The API handles the Email Address in the following two cases.

- If the Email Address is valid, the API will send the *Salt* back to the client:

  Salt = SHA-256 (“mega.nz" || “Padding"|| “PP . . . P"||*Client Random Value*)

- If the Email Address is invalid, the API will send the *Salt* back to the client:

  Salt = SHA-256 (Email Address ||“mega.nz"|| “Padding"|| “PP . . . P"||*Server Random Value*)

Step3. The client will generate the *DK* as follows:

$$DK = \text{PBKDF2-HMAC-SHA-512}\,(Password, Salt,\ 100000,\ 256)$$

Step4. The client will take the first half of the *DK* from left to right as the *DEK*, and take the remaining half of the *DK* as the *DAK*.

Step5. The client will send the string *Email Address* || *DAK* to the API.

Step6. The API will generate the *HAK* as follows:

$$HAK = \text{SHA} - 256\,(DAK)\,.$$

Note, the *HAK* consists of the first 128 bits of the output of SHA-256. The API will verify the user's identity by comparing the *HAK* stored in the database. If the user successfully passes authentication, the API will return their *Encrypted Private RSA Key*, *Encrypted MK* etc.

2.2 The Upload Encryption of File and Folder

The keys of each file or folder are different. Since the folder does not contain data, the folder is not encrypted, and only the folder attribute (that is, the folder name) is encrypted.

To encrypt the file, the *File Key(FK)* consists of 128 random bits and 64 random bits *none*. The file is split into chunks, and each chunk is encrypted using Advanced Encryption Standard-Counter with Cipher lock chaining Message Authentication Code mode (AES-CCM). The *nonce* in each encrypted block is incremented.

After all chunks are encrypted, a *Condensed Message Authentication Code (MAC)* will be calculated according to the following steps: Firstly, A 128-bits array is initialized to zero. Secondly, the array is XORed with a block MAC, and the result is encrypted with Advanced Encryption Standard-Electronic Codebook Mode (AES-ECB). Again, each subsequent MAC block is processed according to this method. The final encryption result is the final MAC.

The *FK* is uploaded to the API after processed as follows:

- An *Obfuscated File Key(OFK)* is created by computing:

  TMAC[0] = Condensed MAC[0] xor Condensed MAC[1], TMAC[1] = Condensed MAC[2] xor Condensed MAC[3]

  OFK = [

  FK[0] xor IV[0], FK[1] xor IV[1],

  FK[2] xor TMAC[0],

  FK[3] xor TMAC[1],

  IV[0], IV[1], TMAC[0], TMAC[1]

  ];

- The *OFK* is encrypted with the *MK* as follows:

  $Encrypted\ File\ Key\ = \text{AES-ECB}\,(MK,\ OFK)$

- The *Encrypted File Key* is uploaded to the API.

3 MEGA Security Analysis

All communication between MEGA client and server is protected by TLS encryption from the time it is installed, and intercepting key packets during registration and login from traffic is not feasible unless the TLS encryption mechanism can be broken. Is MEGA unbreakable? Not necessarily. While MEGA offers end-to-end encryption, it does not use two-factor authentication for logins, so an attacker can log into each account using only login credentials and grab the name of the file in the account. Many users use Email Address as a user name and use the same user name and password for multiple sites. According to Troy Hunt [9], administrator of the website "Have I Been Pwned", a massive file leak on MEGA in 2019 contained over 12,000 individual files and 87GB of data. It contained nearly 773 million email addresses and 22 million passwords.

Meanwhile, while communication between MEGA client and server is protected by TLS encryption, anyone other than MEGA's uploader who wants to access or download the uploader's material needs the uploader to give him a public link to share the file or folder. When the downloader is an unregistered MEGA user, the uploader can only send it through insecure channels. In this case, if an attacker obtains a public link to a file or folder, he may access and download encrypted file data stored on the MEGA server to which the link points, as detailed in the next section for analysis and attack implementation.

4 Cracking the Password Protected Public Links

Anyone other than the MEGA uploader who wants to access or download the uploader's profile needs the uploader to give him a public link to share the file or folder. Public links are classified into plaintext public links and password protected public links.

4.1 Analysis of the Plaintext Public Link

The plaintext public file links are as follows:

https://mega.nz/file/Base64(Handle)#Base64(Key).

The plaintext public folder links are as follows:

https://mega.nz/folder/Base64(Handle)#Base64(Key).

In the above links, "Handle" is the Handle of a file or folder, similar to ID or index. "Key" is the *OFK* for public file links and the *Share Key* for public folder links.

The generic format of the plaintext public link is shown in Tab. 2.

Table 2: Generic format of plaintext public link

Generic head	Type	Generic operator	Data 1	Generic operator	Data 2
https://mega.nz/	file/floder	/	Base64(Handle)	#	Base64(Key)

4.2 Analysis of the Password Protected Public Links

The password protected public file or folder links are as follows:

https://mega.nz/#P!Base64(data).

The generic format of password protected public link is shown in Tab. 3.

The difference in the format of MEGA File and Folder's password protected public links is the length of the data section. The length of the data in password protected public folder links is equal to 118, and the length of the data in password protected public file links is equal to 139.

Table 3: Generic format of password protected public link

Generic head	Generic operator	Data
https://mega.nz/	#P!	Base64(data)

The procedure for constructing a password protected link is as follows:

Step1. Key generation.

- ■ A *DK* of 512 bits was calculated by computing PBKDF2-HMAC-SHA512(*Password*, *Salt*, 100000, 256).
- ■ The key lengths of folder and file links are 128 bits and 256 bits, respectively. The first 128 bits of the *DK* will be used to encrypt the folder key by XOR. The first 256 bits of the *DK* will be used to encrypt the file key by XOR
- ■ The *MAC Key* consists of the last 256 bits of the *Derived Key*.

Step2. Generating the data.

The format of the generated data is

Algorithm ||Type|| Public Handle ||Salt|| Encrypted Key || MAC Key.

In the above format, the meaning of each field identification is shown in Tab. 4.

Table 4: The meaning of each field identification

Field Name	Meaning	Byte length
Algorithm	An identifier is used to indicate which algorithm is used.	1
Type	An identifier is used to indicate the type of link. Type = 1, the link is a file. Type = 0, the link is a folder.	1
Public handle	The handle of the public folder or file	6
Salt	A string of random number bytes	32
Encrypted key	The ciphertext of folder or file key encrypted with *Encryption Key*	16/32
MAC key	the last 256 bits of *DK*	32

Step3. Constructing protected links.

- ■ A MAC Tag of 32 bytes is computed by

$$\text{MAC Tag} = \text{HMAC-SHA-256}\,(\text{MAC Key},\ (\text{Algorithm}||\text{Type}||\text{Public Handle}||\text{Salt}||\text{Encrypted Key}))\,.$$

- ■ The format of protected link data is constructed by Algorithm || Type || Public Handle || Salt || Encrypted Key || MAC Tag.

According to Tab. 3, we firstly Base64 encode the link data, then substitute incompatible characters, and finally get a password protected link, for example, *https://mega.nz/#P!WWWT5WcTsZ7Z_ghxV0FTJXKOQZs_3a*...

4.3 *Cracking Algorithm of the Password Protected Public Links*

When the downloader is an unregistered MEGA user, the uploader can only send it through insecure channels. In this case, the attacker has a chance to obtain a public link to a file or folder. If the public link is not password protected, the attacker can use the link to access and download encrypted file data stored on the MEGA server to which the link points. If the link is password-protected, the attacker needs to crack it first.

As you can see from the construction process of the password protected public links, its security depends on the password entered by the user. Although MEGA excludes passwords that it considers weak by forcing users to input passwords with a length greater than 8 and using different types of characters, to facilitate memory, users are usually far from meeting the requirements of random construction when constructing passwords [10–14]. Generally, people choose passwords that are easy to remember for themselves, resulting in the centralized distribution of a large number of passwords in the whole password range, which greatly improves the success rate of the attacker to crack passwords [15–19]. Next, we give the cracking algorithm of the password protected links based on password guessing.

The cracking process of the password protected links is as follows:

Step1. Replace characters.

- Keep the characters after # P! for cracking, and keep the preceding characters for assembling the plaintext URL.
- Replace ‘-’ by ‘+’.
- Replace ‘_’ by ‘/’.
- Replace ‘,’ by ‘ ’.

Step2. Base64 decoding.

- Call the Base64 decoder function to get the data in the form of “Algorithm || Type || Public Handle || Salt || Encrypted Key || MAC Tag”.

Step3. Parse plaintext.

- Algorithm: 1 byte, not used in the cracking.
- Type: 1 byte, used to concatenate plaintext URL, reserved.
- Public Handle: 6 bytes, corresponds to plaintext link data 1, used to recover the plaintext URL, reserved.
- Salt: 32 bytes, used to calculate the DK, reserved.
- Encrypted Key: 16 or 32 bytes, corresponds to plaintext link data 2, used to recover the plaintext URL, reserved.
- MAC Tag: 32 bytes, used to verify the accuracy of the password *pwd*, reserved.

Step4. Cracking the password.

- Guess the password *pwd*.
- Calculate DK: DK = PBkDF2_HAMC_SHA512 (100000, Salt, *pwd*).
- Calculate Mac Tag*: Mac Tag* = HMAC-SHA-256 (DK[256:], (Algorithm || Type || Handle || Salt || Encrypted Key)). DK[256:] represents the last 256 bits of the DK.
- Mac verification: if Mac Tag* = Mac Tag, *pwd* is the correct password; Otherwise, continue to guess the password.

Step5. Constructing plaintext links.

- Base64(Public Handle) calculates and replaces characters: Base64 encodes Public Handle in step2 and replaces ‘ + ’ by ‘ - ’, ‘/’ by ‘_’, and ‘\n’ by ‘ ’ in the encoded string. Next, remove “=” if there is “=”.
- Recovery of the original Encryption Key: Select the first 128 or 256 bits (from left) of the *DK* according to the type in Step3, and XOR of the Encrypted Key in step3 to generate the original Encryption Key of the corresponding length.
- Base64(original Encryption Key): Base64 encoding the original Encryption Key in the previous step, and replace ‘+’ by ‘-’, ‘/’ by ‘_’, and ‘\n’ by ‘ ’ in the encoded string. Next, remove “=” if there is “=”.
- Concatenate plaintext URL: The form of the plaintext URL concatenation is as follows:

https://mega.nz/Type/Base64(PublicHandle)#Base64(originalEncryptionKey)

Here, Type is taken from step3, Folder is concatenated in the link if Type is 0, and File is concatenated in the link if Type is 1.

4.4 Examples of Cracking

4.4.1 Cracking the Password Protected Public Folder Links

Here, we give a password protected public folder link:

https://mega.nz/#P!AgDJUADUNBAfILW6rGEVD0Po68-q27s0jbEYyvSDQ1fbS9EeUvFv06geiV6jv4hTl1aXNNyE--SVwYKRfvJgU_VFkmiIltBY0Z5JtmvhHJ69uyZOxSIGUA.

The link is protected by the password “*123.admin.30S*”.

Next, we begin to crack according to the cracking algorithm given in Section 4.3. The cracking process is as follows:

Step1. Intercept and replace characters.

We intercept the following data:

AgDJUADUNBAfILW6rGEVD0Po68-q27s0jbEYyvSDQ1fbS9EeUvFv06geiV6jv4hTl1aXNNyE–SVwYKRfvJgU_VFkmiIltBY0Z5JtmvhHJ69uyZOxSIGUA.

The length of the intercepted data is 118. The data after replacing the intercepted data is as follows:

AgDJUADUNBAfILW6rGEVD0Po68+q27s0jbEYyvSDQ1fbS9EeUvFv06geiV6jv4hTl1aXNNyE++SVwYKRfvJgU/VFkmiIltBY0Z5JtmvhHJ69uyZOxSIGUA.

Two “=” are concatenated in the back of the above data, that is,

AgDJUADUNBAfILW6rGEVD0Po68+q27s0jbEYyvSDQ1fbS9EeUvFv06geiV6jv4hTl1aXNNyE++SVwYKRfvJgU/VFkmiIltBY0Z5JtmvhHJ69uyZOxSIGUA==,

which is to be decoded by Base64, with a length of 120.

Step2. Base64 decoding.

The decoded data are as follows:

\x02\x00\xc9P\x00\xd44\x10\x1f\xb5\xba\xaca\x15\x0fC\xe8\xeb\xcf\xaa\xdb\xbb4\x8d\xb1\x18\xca\xf4\x83CW\xdbK\xd1\x1eR\xf1o\xd3\xa8\x1e\x89#####\xa3\xbf\x88S\x97V\x974\xdc\x84\xfb\xe4\x95\xc1\x82\x91~\xf2`S\xf5E\x92h\x88\x96\xd0X\xd1\x9eI\xb6k\xe1\ x1c\x9e\xbd\xbb&N\xc5”\x06P

Step3. Parse plaintext.

- Algorithm: 0x02.
- Type: 0x00, Folder.
- Public Handle: 6 bytes, \xc9P\x00\xd44\x10.
- Salt: 32 bytes, \x1f\xb5\xba\xaca\x15\x0fC\xe8\xeb\xcf\xaa\xdb\xbb4\x8d\xb1\x18\xca\xf4\x83CW\xdbK\xd1\x1eR\xf1o\xd3.
- Encrypted Key: 16 bytes, \xa8\x1e\x89######\xa3\xbf\x88S\x97V\x974\xdc\x84\xfb\xe4.
- MAC Tag: 32 bytes, \x95\xc1\x82\x91~\xf2`S\xf5E\x92h\x88\x96\xd0X\xd1\x9eI\xb6k\xe1\x1c\x9e\xbd\xbb&N\xc5"\x06P.

Step4. DK calculation and HMAC calculation.

- Traverse each password *pwd* in the password set and calculate the DK by DK = PBkDF2 _HAMC_SHA512 (100000, Salt, *pwd*).
- Calculate HMAC by Mac Tag* = HMAC-SHA-256 (DK[256:], (Algorithm || Type || Public Handle || Salt || Encrypted Key)).
- Judge MAC Tag < > MAC Tag*.

Step5. Constructing plaintext links.

- Recovery of the original Encryption Key: \\Z:z3J\xee\xf0\xb5U\x9c]I/\xa5y
- Base64 encode the handle and replace the characters: yVAA1DQQ
- Base64 encode the original Encryption Key and replace the characters: XFo6ejNK7vC1VZxdSS-leQ
- Constructing plaintext URL: *https://mega.nz/floder/yVAA1DQQ#XFo6ejNK7vC1VZxdSS-leQ*

4.4.2 Cracking the Password Protected Public File Links

Here, we give a password protected public file link:

https://mega.nz/#P!AgFsUmcdCoPHOobt_DN5op-rhFtI6AF0mxDyzh7OAC_frDSVebejj8xIDciZUIb19Sg-xb-0YkpeqvrKjOKivzyGp1W8Plf3QAmgELeneVg_xmOxpck8diLiM8UnbOuYCHb4JnpyeHQ.

The link is protected by the password "*admin.30S*".

Next, we begin to crack according to the cracking algorithm given in Section 4.3. The cracking process is as follows:

Step1. Intercept and replace characters.

We intercept the following data:

AgFsUmcdCoPHOobt_DN5op-rhFtI6AF0mxDyzh7OAC_frDSVebejj8xIDciZU Ib19Sg-xb-0YkpeqvrKjOKivzyGp1W8Plf3QAmgELeneVg_xmOxpck8diLiM8UnbOuYCHb4JnpyeHQ.

The data after replacing the intercepted data is as follows:

b'AgFsUmcdCoPHOobt/DN5op+rhFtI6AF0mxDyzh7OAC/frDSVebejj8xIDciZUIb19Sg+xb+0YkpeqvrKjOKivzyGp1W8Plf3QAmgELeneVg/xmOxpck8diLiM8UnbOuYCHb4JnpyeHQ='

Step2. Base64 decoding.

The decoded data are as follows:

b'\x02\x01lRg\x1d\n\x83\xc7:\x86\xed\xfc3y\xa2\x9f\xab\x84[H\xe8\x01t\x9b\x10\xf2\xce\x1e\xce\x00/\xdf\xac4\x95y\xb7\xa3\x8f\xccH\r\xc8\x99P\x86\xf5\xf5(>\xc5\xbf\xb4bJ

#####\xaa\xfa\xca\x8c\xe2\xa2\xbf<\x86\xa7U\xbc>W\xf7@\t\xa0\x10\xb7\xa7yX?\xc6c\xb1\xa5\xc9<v"\xe23\xc5\'l\xeb\x98\x08v\xf8&zrxt'

Step3. Parse plaintext.

- Algorithm: 0x02.
- Type: 0x01, File.
- Public Handle: 6 bytes, b'lRg\x1d\n\x83'.
- Salt: 32 bytes, b'\xc7:\x86\xed\xfc3y\xa2\x9f\xab\x84[H\xe8\x01t\x9b\x10\xf2\xce\x1e\xce\x00/\xdf\xac4\x95y\xb7\xa3\x8f'.
- Encrypted Key: 16 bytes, b'\xccH\r\xc8\x99P\x86\xf5\xf5(>\xc5\xbf\xb4bJ#####\xaa\xfa\xca\x8c\xe2\xa2\xbf<\x86\xa7U\xbc>W\xf7'.
- MAC Tag: 32 bytes, b'@\t\xa0\x10\xb7\xa7yX?\xc6c\xb1\xa5\xc9<v"\xe23\xc5\'l\xeb\x98\x08v\xf8&zrxt'.

Step4. DK calculation and HMAC calculation.

- Traverse each password *pwd* in the password set and calculate the DK by DK = PBkDF2_ HAMC_SHA512 (100000, Salt, *pwd*).
- Calculate HMAC by Mac Tag* = HMAC-SHA-256 (DK[256:], (Algorithm || Type || Public Handle || Salt || Encrypted Key)).
- Judge MAC Tag < > MAC Tag*.

Step5. Constructing plaintext links.

- Recovery of the original Encryption Key: b'\xf8\xa6\xa2e\x8f\xafvHz\x0b\xed\x16]w\x1a\xaa)G\xb1?\x9a\x04\xf3e\x1f\xab\x11\xa5/\x19\x1a\x98'
- Base64 encode the handle and replace the characters: bFJnHQqD
- Base64 encode the original Encryption Key and replace the characters: -KaiZY-vdkh6C-0WXXcaqilHsT-aBPNlH6sRpS8ZGpg

Constructing plaintext URL: *https://mega.nz/file/bFJnHQqD#-KaiZY-vdkh6C-0WXXcaqilHsT-aBPNlH6sRpS8ZGpg*.

5 Efficiency Analysis and Suggestions

Our attack method mainly depends on password guessing, so the efficiency of our attack method is equal to that of password guessing. The efficiency of password guessing generally depends on the rate of password guessing, password guessing algorithm, the complexity of the guessed password, and the size of the password training set. Since it is difficult to collect the data of MEGA's password protected public links, we will use the public data to analyze the efficiency of the password guessing method, as shown in Tabs. 5 and 6 [20].

Table 5: Intra-site password cracking

	10% training data				30% training data				50% training data			
	PCFG	OMEN	J215	H4	PCFG	OMEN	J215	H4	PCFG	OMEN	J215	H4
17173.com	.4826	.5711	.5940	.1491	.5776	.5705	.5934	.1104	.6525	.5718	.5940	.0829
178.com	.5270	.6097	.6168	.1839	.5675	.6096	.6165	.1359	.5828	.6091	.6161	.1020

(Continued)

Table 5: Continued

	10% training data				30% training data				50% training data			
	PCFG	OMEN	J215	H4	PCFG	OMEN	J215	H4	PCFG	OMEN	J215	H4
7k7k	.4550	.6024	.6376	.1642	.5849	.6026	.6379	.1220	.6186	.6027	.6385	.0914
CSDN	.3312	.3860	.3941	.1875	.3602	.3874	.3927	.1386	.3768	.3866	.3932	.1045
Duduniu	.3731	.4198	.4571	.0645	.4293	.4198	.4582	.0478	.4481	.4209	.4573	.0359
Hotmail	.1728	.1112	.4359	.0060	.1936	.2967	.4662	.0054	.2006	.3240	.4758	.0058
LinkedIn	.1616	.1333	.1724	.0007	.1636	.1367	.1721	.0006	.1656	.1337	.1718	.0004
MySpace	.5150	.3504	.4401	.0075	.5332	.4238	.4482	.0060	.5399	.4407	.4465	.0047
phpBB	.2758	.3754	.4473	.0032	.2877	.4176	.4511	.0025	.2921	.4214	.4523	.0021
Renren	.4090	.5178	.6116	.1647	.4565	.5187	.6118	.1219	.4754	.5177	.6120	.0916
Rockyou	.4623	.5059	.5445	.0067	.4777	.5058	.5440	.0050	.4844	.5055	.5441	.0037
Tianya	.4820	.5814	.6501	.1654	.5417	.5815	.6502	.1207	.5633	.5824	.6500	.0921
Yahoo!	.4050	.3700	.3797	.0039	.4161	.3765	.3780	.0032	.4184	.3797	.3784	.0022

Notes: *Each value in this table represents the fraction of passwords been cracked in a dataset (*e.g., *.4826 indicates that 48.26 percent passwords of a dataset have been cracked). Default number of guesses; ~10^9 for PCFG and Ordered Markov ENumerator (OMEN); ~1.4×10^9 for JtR-J/JtR-B Markov mode with levels 215 (J215) and Hashcat Markov mode with threshold 4 (H4).*

Table 6: Intra-site password cracking

	Training data:Tianya				Training data:Rockyou				Training data:Tianya + Rockyou			
	PCFG	Sem+	J215	H4	PCFG	Sem+	J215	H4	PCFG	Sem+	J215	H4
17173.com	.4693	.5615	.5677	.0808	.3664	.3696	.4781	.0004	.4732	.5380	.5390	.0811
178.com	.4959	.5768	.5674	.0686	.4092	.3988	.4698	.0000	.4974	.5484	.5384	.0707
7k7k	.5486	.6248	.6275	.0905	.3807	.3914	.5477	.0006	.5531	.6090	.6076	.0897
CSDN	.3790	.4752	.3915	.1047	.2918	.3084	.3163	.0001	.3760	.4414	.3694	.1107
Duduniu	.3885	.4920	.4066	.0369	.3214	.3295	.3400	.0007	.3983	.4779	.3890	.0352
Hotmail	.3071	.2936	.3105	.0035	.3279	.5259	.4512	.0030	.3399	.4988	.4293	.0037
LinkedIn	.1519	.1426	.0971	.0003	.1776	.2873	.1757	.0004	.1769	.2662	.1589	.0003
MySpace	.3648	.4182	.2163	.0015	.5679	.7495	.4146	.0021	.5465	.7026	.3713	.0017
phpBB	.3239	.3653	.3007	.0021	.3551	.5602	.4438	.0017	.3609	.5458	.4201	.0022
Renren	.5176	.5722	.5823	.0897	.4409	.4968	.5744	.0019	.5328	.6110	.6051	.0869
Rockyou	.4190	.4539	.3588	.0186	.5407	.7909	.5441	.0037	.5323	.7434	.5108	.0197
Tianya	.7221	.8037	.6499	.0920	.4241	.4322	.5696	.0008	.7193	.7812	.6379	.0937
Yahoo!	.3906	.3802	.2203	.0074	.4433	.6138	.3779	.0020	.4416	.5734	.3449	.0081

Notes: *Each value in this table represents the fraction of passwords been cracked in a dataset (*e.g., *.4693 indicates 46.93 percent passwords of a dataset have been cracked). Number of guesses: ~10^9 for PCFG; ~1.4×10^9 for Sem+, J215, and H4*

It can be seen from Tabs. 1 and 2 that when the password space is 1.4×10^9, the success rate of probabilistic context-free grammars (PCFG) in intra-site password cracking is more than 40%, and the success rate is greater with the increase of the password training set. If it is cross-site password cracking and the training set is homologous, the average success rate of PCFG is more than 30%. As far as the guessing algorithm is concerned, the success rate of semantic based password cracking algorithm (Sem+) [21] is about 7% higher than that of PCFG.

According to the above analysis of password guessing efficiency, to improve the security of MEGA's password protected public links, our suggestions are as follows:

1. Increase the complexity of password setting, such as requiring at least 12 bits, including upper case letters, lower case letters, numbers, special characters, etc.
2. The number of iterations of generating *DK* in Section 4.2 is increased to more than 400000.

The combination of the above two measures will greatly reduce the efficiency of password guessing, effectively resist password attacks and enhance the security of MEGA's password protected public links.

6 Conclusion

MEGA is an end-to-end encrypted cloud storage platform controlled by users. The content data stored on MEGA are encrypted on the user's client before the user uploads them to the MEGA platform. The encryption key of the data will not be directly saved on the platform. Therefore, the access to the data stored on Mega is controlled by the user, not the platform. Even the platform cannot access the data. MEGA not only has a good security mechanism of its own but all communication between the client and the server is protected by TLS encryption. So, MEGA's forensics were very difficult. This paper clarifies the encryption mechanism of MEGA and finds out the vulnerability of the MEGA encryption mechanism. A method to attack MEGA is presented, and the secret data stored in the MEGA server can be accessed or downloaded. Finally, two examples are given to verify the correctness of the method. Therefore, the result of this paper provides an effective method for MEGA's forensics.

Funding Statement: This work was supported by the Key Laboratory of confidential communication Fund Project (No.6142103190308).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] "Source Code Transparency." (Accessed Oct. 25, 2021) [Online] https://mega.nz/sourcecode.
[2] Mega Limited. (Accessed Oct. 25, 2021) [Online] https://github.com/meganz/.
[3] H. Mestiri, I. Barraj, A. A. Mohamed and M. Machhout, "An efficient aes 32-bit architecture resistant to fault attacks," *Computers, Materials & Continua*, vol. 70, no. 2, pp. 3667–3683, 2022.
[4] J. Kh-Madhloom, M. Khanapi and M. R. Baharon, "Ecg encryption enhancement technique with multiple layers of aes and dna computing," *Intelligent Automation & Soft Computing*, vol. 28, no. 2, pp. 493–512, 2021.
[5] M. Weir, S. Aggarwal, B. D. Medeiros and B. Glodek, "Password cracking using probabilistic context-free grammars," in *Proc. IEEE Symp. Security & Privacy(S&P'09)*, California, USA, pp. 391–405, 2009.
[6] C. Herley, "An administrator's guide to unternet password research," in *Proc. USENIXLISA2014*, San Diego, USA, pp. 44–61, 2014.
[7] J. Bonneau, C. Herley, P. C. V. Oorschot and F. Stajan, "Passwords and the evolution of imperfect authentication," *Communications of the ACM*, vol. 58, no. 7, pp. 78–87, 2015.

[8] Mega Limited. MEGA security whitepaper, Second Edition-January 2020. (Accessed Oct. 25, 2021) [Online]. https://mega.nz/SecurityWhitepaper.pdf.

[9] Threat Watch. "MEGA cloud dump exposes 87GB of data," (Accessed Oct. 25, 2021) [Online] https://www.binarydefense.com/threat_watch/mega-cloud-dump-exposes-87gb-of-data/.

[10] J. Bonneau, "The science of guessing: Analyzing an anonymized corpus of 70 million passwords," in *Proc. IEEE Symp. Security & Privacy(S&P'12)*, California, USA, pp. 538–552, 2012.

[11] D. Malone and K. Maher, "Investigating the distribution of password choices," *Computer Ence*, vol. 8, no. 3, pp. 301–310, 2012.

[12] M. D. Leonhard and V. N. Venkatakrishnan, "A comparative study of three random password generators," in *Proc. IEEE Int. Conf. on Electro/Information Technology (EIT 2007)*, Chicago, USA, pp. 227–232, 2007.

[13] R. Veras, J. Thorpe and C. Collins, "Visualizing semantics in passwords: The role of dates," in *Proc. of the 9th Annual Int. Symp. on Visualization for Cyber Security VizSec '12*, Seattle, USA, pp. 88–95, 2012.

[14] J. Yan, A. Blackwell, R. Anderson and A. Grant, "Password memorability and security: Empirical results," *IEEE Security & Privacy Magazine*, vol. 2, no. 5, pp. 25–31, 2004.

[15] S. Houshmand, S. Aggarwal and R. Flood, "Next gen PCFG password cracking," *IEEE TRANSACTIONS on Information Forensics and Security*, vol. 10, no. 8, pp. 1776–1791, 2015.

[16] D. Wang, Z. Zhang, P. Wang, J. Yan and X. Huang, "Targeted online password guessing: An underestimated threat," in *Proc. of the 23nd ACM Conf. on Computer and Communications Security (ACM CCS 2016)*, Vienna, Austria, pp. 1242–1254, 2016.

[17] B. Pal, T. Daniel, R. Chatterjee and T. Ristenpart, "Beyond credential stuffing: Password similarity models using neural networks," in *Proc. of the 40th IEEE Symposium on Security and Privacy (IEEE S&P 2019)*, Washington DC, USA, pp. 417–434, 2019.

[18] M. K. Qabalin, Z. A. Arida, O. A. Saraereh, F. Wu, I. Khan *et al.,* "An improved dictionary cracking scheme based on multiple gpus for wi-fi network," *Computers, Materials & Continua*, vol. 66, no. 3, pp. 2957–2972, 2021.

[19] X. Chu, J. Gao and B. Sheng, "Efficient concurrent 11-minimization solvers on gpus," *Computer Systems Science and Engineering*, vol. 38, no. 3, pp. 305–320, 2021.

[20] S. Ji, S. Yang, X. Hu, W. Han, Z. Li *et al.,* "Zero-sum password cracking game: A large-scale empirical study on the crackability, correlation, and security of passwords," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 5, pp. 550–564, 2017.

[21] R. Veras, C. Collins and J. Thorpe, "On the semantic patterns of passwords and their security impact," in *Proc. Netw. Distrib. Syst. Security Symp.*, San Diego, USA, pp. 1–16, 2014.