A forensic analysis of rclone and rclone's prospects for digital forensic investigations of cloud storage

By:
Frank Breitinger, Xiaolu Zhang and Darren Quick

From the proceedings of
The Digital Forensic Research Conference
DFRWS APAC 2022
Sept 28-30, 2022

DFRWS is dedicated to the sharing of knowledge and ideas about digital forensics research. Ever since it organized the first open workshop devoted to digital forensics in 2001, DFRWS continues to bring academics and practitioners together in an informal environment.
As a non-profit, volunteer organization, DFRWS sponsors technical working groups, annual conferences and challenges to help drive the direction of research and development.
https://dfrws.org

Forensic Science International: Digital Investigation 43 (2022) 301443

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS 2022 APAC - Proceedings of the Second Annual DFRWS APAC

A forensic analysis of `rclone` and `rclone`'s prospects for digital forensic investigations of cloud storage

Frank Breitinger [a, *], Xiaolu Zhang [b], Darren Quick [c, 1]

[a] School of Criminal Justice, University of Lausanne, 1015, Lausanne, Switzerland
[b] Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX, 78249, United States
[c] South Australia Police, 100 Angas St, Adelaide, Australia

ARTICLE INFO

Article history:

Keywords:
Rclone
Cloud storage
Acquisition
Application forensics
Cloud computing forensics

ABSTRACT

Organizations and end users are moving their data into the cloud and trust Cloud Storage Providers (CSP) such as pCloud, Dropbox, or Backblaze. Given their popularity, it is likely that forensic examiners encounter one or more online storage types that they will have to acquire and analyze during an investigation. To access cloud storage, CSPs provide web-interfaces, proprietary software solutions (e.g., Dropbox client for Windows) as well as APIs allowing third-party access. One of these third-party applications is *rclone* which is an open-source tool to access many common CSPs through a command line interface. In this article, we look at rclone from two perspectives: First, we perform a forensic analysis on rclone and discuss aspects such as password recovery of the configuration file, encryption, and JA3 fingerprints. Second, we discuss rclone as a prospect to be a forensic tool which includes its read-only mount feature and sample cases. Under the circumstances tested, rclone is suitable for forensic practitioners as it is open-source, documented, and includes some essential functionality frequently needed but practitioners need to be aware of the caveats.

© 2022 The Author(s). Published by Elsevier Ltd on behalf of DFRWS This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

Rclone is a platform-independent software that offers a documented command line interface (CLI) to access a variety of cloud storage providers (CSPs).[2] In this article, rclone is discussed from two angles:

Forensic analysis of rclone: Instead of installing one client application per CSP, rclone provides one single interface for accessing storage. This feature makes it an interesting tool for criminals. For instance, according to Greetham (2021), rclone has been used by "a large number of ransomware cases […] for data exfiltration". Consequently, it is important for investigators to understand how this tool is configured and what artifacts it leaves on the system. Contribution: We conduct a comprehensive forensic analysis of the rclone application and present artifacts found on the system, in memory, and the network.

Rclone as a forensic tool: From a digital forensic investigation perspective, cloud storage poses several jurisdictional and technical challenges. One of these technical challenges is the acquisition of evidence, i.e., accessing and downloading data, as there is only a limited number of forensic tools available (conventional tools have focused upon having physical access to the media that stores the data (Quick et al., 2013)). This forces examiners to fall back on applications provided by CSPs or utilize a web interface (if available) which brings two problems:

Ease of acquisition: Given the sheer amount of CSPs, many applications are necessary to access the cloud storage making it time-consuming and hampering automation.
Error-proneness: As these applications have not been developed for forensic purposes, there is an unnecessary risk that an investigator modifies/deletes data and therefore may be accused

* Corresponding author.
E-mail addresses: frank.breitinger@unil.ch (F. Breitinger), xiaolu.zhang@utsa.edu (X. Zhang), darren.quick@gmail.com (D. Quick).
URL: https://www.FBreitinger.de

[1] Dr. Quick contributed to this article in his personal capacity. The views expressed are his own and do not necessarily represent the views of the South Australia Police or the Australian Government.

[2] We use the terms cloud storage and cloud storage service as synonyms describing disk space provided by service providers; examples are Google Drive, Microsoft OneDrive, or Dropbox.

https://doi.org/10.1016/j.fsidi.2022.301443
2666-2817/© 2022 The Author(s). Published by Elsevier Ltd on behalf of DFRWS This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

of data modification. Furthermore, it also requires vetting the CSP applications beforehand to ensure they work as expected.

In comparison, for offline storage, practitioners may use write blockers automatically performing a forensic sound acquisition. Contribution: We demonstrate and discuss rclone's viability for forensic investigations based on two sample cases.

Situating this work: As discussed by Chung et al. (2012, Fig. 1), there are several steps involved when investigating CSPs such as considering the Smartphone and PC/MAC, analyze locally found data (with the aim to retrieve user credentials), or obtaining a search and seizure warrant. This article *only* focuses on the two steps 'access to user's cloud storage' followed by 'collect data in cloud storage'. An investigator may encounter other challenges such as 2-Factor-Authentication, jurisdictional obstacles, or missing user credentials. These aspects are not addressed by this work. Furthermore, this article focuses on consumer CSPs and ignores business solutions.

Background: Many CSPs can be accessed through a web interface (Browser) or client application where the latter provides better usability and security, e.g., automatically sync modified files with the cloud or the usage of zero-knowledge encryption. We differentiate between two mechanisms where, in both cases, more than one device may be connected:

Storage: The data is stored in the cloud and no local backup exists. To access the data, it requires an Internet connection. The client application downloads a requested file on-the-fly or the cloud storage is mounted to the local system.
Sync: A copy of the data is stored local but also synchronized to the cloud. The synchronization may be immediately or timed (e.g., once per day for larger files).

Naturally, there are hybrid solutions where some folders are synchronized, and others are not. An example would be Dropbox's smart sync where the user decides which folders/files are only stored online and no local copy exists.

Outline: The remainder of this work is organized as follows. The next section summarizes the Related work. This is followed by the Rclone application forensics presenting artifacts found on the system. In Rclone as a forensic tool we discuss the viability of rclone as a forensic tool by looking into key functionality. The last section concludes this article.

We also provide a short summary of rclone and its features in A which is recommended for readers completely unfamiliar with rclone (the article keeps general aspects/functionality at a minimum). Alternatively, one may read the official documentation: rclone.org/docs/.

2. Related work

In 2010, Gartner estimated cloud computing services produced a profit of $68.3 billion, which was a 16.6% increase from 2009, and was forecast to be $148.8 billion by 2014 (Chung et al., 2012). In 2021, with an increased reliance on cloud infrastructure due to the impact of the COVID-19 pandemic with many organizations switching to work-from-home, enabling a greater use of online collaboration, and a hybrid workforce, Gartner forecast spending on public cloud services to total $332.3 billion in 2021, and nearly $400 billion by 2022, increasing from $270 billion in 2020 (Costello and Rimol, 2021).

With cloud computing and cloud storage gaining popularity, the digital forensics community also started to investigate its impact on investigations and what traces can be found. Given the sheer number of publications, this section primarily discusses literature focusing on cloud storage (services) and forensics but ignores secondary literature, e.g., cloud computing in general, distributed storage technologies (Ricci et al., 2019).

2.1. Cloud storage analysis

Over the years several articles have been published which examine local traces of cloud storage providers, where a local trace is an artifact found on a (seized) device. Predominately these focus on examining traces of client software, using digital forensic tools, traces of web browser artifacts, or use the providers' software to collect data, such as Chung et al. (2012) where cloud remnants from Amazon, Google, Dropbox, and Evernote on Windows and MacOS computers and iOS and Android on mobile devices are examined. Similarly, Quick et al. (2013) examine Dropbox, SkyDrive (OneDrive), Google Drive remnants on Windows and iOS devices, and Hale (2013) examines Amazon Cloud Drive remnants on Windows computers.

Of note is Federici (2014) who examine Dropbox, along with Google Drive and SkyDrive (OneDrive) and develop an imaging solution for these providers, and Roussev et al. (2016) developed a collection tool for four major providers: Dropbox, Box, Google Drive, and OneDrive (also discussed in Ahmed and Roussev (2019)). A specific tool for Google Docs collection was also developed (also discussed in Roussev and McCulley (2016)), and a third tool to provide a filesystem interface and remotely mount a cloud drive (Roussev et al., 2016). Shariati et al. (2016) examined Sugarsync remnants on Windows and MacOS computers, and iOS and Android devices. These indicate a demand for software tools to enable the collection of cloud stored data in a forensically sound manner for a variety of cloud service providers.

2.2. Usage of cloud storage

Chung et al. (2012) stresses the importance to not only analyze the client application and its traces but also cookies and log files of web browsers to identify if cloud storage has been accessed. For client applications, traces may be found in general log files or the registry (Windows). Client application often also comes with a database file (maybe a text file), containing information about (successful) login attempts or synchronized files.

Data volume is increasing, which causes issues for digital forensic examiners in collecting and examining data in a timely

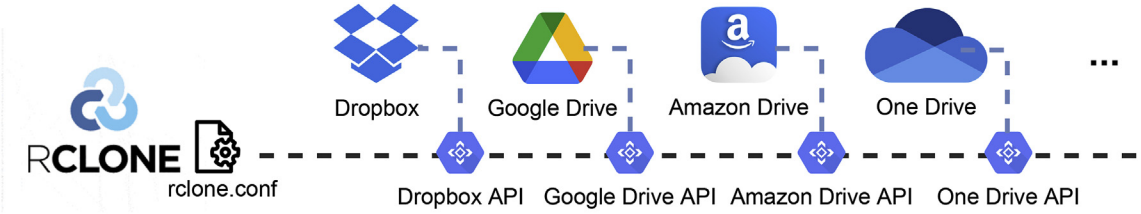

Fig. 1. Rclone authentication.

manner (Quick and Choo, 2014). It is impractical to collect and preserve all data from all devices seized in an investigation due to increasing volumes of data, along with potential collection of co-mingling data from innocent users, and the business impact on cloud providers (Almulla et al., 2014).

Data deletion by users prior to preservation is also an issue, but the preservation policies of cloud storage providers assist law enforcement agencies by retaining deleted data for 30 or 90 days by default, or longer depending on the provider (Quick et al., 2013). In addition, cloud providers often retain data in multiple locations, and hence the impact of a disruption to data from one location is minimized (Almulla et al., 2014).

As Ruan et al. (2011) highlighted, cloud users have no control over the location of their data and can usually only provide examiners an object or container as a location. Identification of a physical location by a user is extremely difficult and relies on a high level of cooperation from the provider. As highlighted by Simou et al. (2014) and Simou et al. (2016) there exists a need to develop tools that enable collection of specific data, which must also abide with forensic principles and standards. This includes ensuring read-only access to remote data (Federici, 2014). Any tool solutions should minimize reliance and involvement of service providers to preclude liability issues and ensure timely collection (Alqahtany et al., 2015).

Electronic crime is defined as computers or computational devices being used as a tool, target, or storage device in the commission of a criminal offence (Police Commissioners' Conference Electronic Crime Working Party, 2000). Cloud-based evidence is not limited to crime or offending in the cloud environment, with much of law enforcement collection and examination of cloud data relating to data stored in the cloud, as opposed to cloud infrastructure being used as a tool or as a target of offending. Cloud storage usage is increasing, as mobile device manufacturers and mobile operating system developers offer cloud storage options to users who regularly upgrade devices and have a need to retain their personal data across devices and across operating systems. With the increase in retained data over many years of use, and limited data storage on devices, this adds to the demand for increasing volume and use of cloud storage infrastructure.

Something that is becoming more and more prolific are requests from investigators and prosecutors for in-depth analysis and specifically, reports around the duplication of key evidence across devices and storage media (digital cross-pollination). These digital cross-pollination analysis requests are time-consuming and even more so when the complexity of cloud stored data is added to the mix. The source of evidential data, oftentimes duplicated across devices, is where an investigation should focus preservation and collection efforts. Discovery and analysis of data located in cloud storage and synchronized across various devices can be crucial to an investigation to tell the complete story and enable facts to be presented to a Court of law to aid decision makers. Digital pollination analysis adds time (and hence cost) to an investigation (Stawski, 2018).

Legislation should encompass the collection of data available to a device at the point of execution of the warrant and subsequent to warrant execution, as oftentimes the use of cloud storage is identified during post-warrant analysis. This is commonplace when a forensic examiner is subsequently analyzing a device extract or computer image and discovers remnants of cloud storage use, such as those outlined in the related work papers. The ability to identify and collect potentially relevant cloud-stored data post-warrant in a forensically sound manner can be crucial to an effective investigation, either confirming known evidence, providing further evidence of offending, or exonerating a suspect and allowing an investigation to move forward and closer to the complete truth.

3. Rclone application forensics

Rclone serves as a universal interface to access various CSPs by implementing their backend APIs and can be used for accessing multiple cloud storages simultaneously (see Fig. 1). This section outlines various artifacts that can be found on the system (Sec. 3.1 to 3.5) and concludes with a procedure for conducting a forensic analysis of a computer that has rclone installed.

3.1. rclone.conf

As shown in Fig. 1, rclone authenticates a user to cloud services through their provided APIs. To maintain access to the authorized accounts, rclone stores the information for authentication (username, password, access tokens, etc.) in a single configuration file named *rclone.conf* (the file name is hardcoded in the source code). As soon as the user is authenticated, the data on the cloud storage can be copied/moved among cloud storages or the local computer. By default, the file is stored under `%AppData%/rclone/rclone.conf` on Windows and under ~/.config/rclone/rclone.conf on other systems (there are some other locations as explained in the documentation[3]).

Configuration files are in the INI format[4]; an example containing three remotes is listed in Fig. 2. The 1st, 9th and 15th line are the user-chosen names of the remote storage followed by one of forty-three the predefined storage `types`, e.g., `ftp` or `dropbox`. The remaining lines differ depending on the remote storage and may include `host`, `user`, `pass`, `key_file` or `token` among others. Remark: While it is possible to manually create the configuration file, it is recommended to use the interactive CLI as many providers use token-based authentication.

Passwords are not stored in clear text but obscured by

```
$ rclone obsure PASSWORD
```

to prevent eyedropping.[5] To obtain the cleartext password, one may use

```
$ rclone reveal OBSCURED_PW
```

Once a configuration file is created or found, it can be used by anyone on any computer. To use a configuration, it must either be placed in the default directory or loaded using the `–config="path/rclone.conf"` option.

3.2. rclone.conf password recovery

Due to its importance, users can encrypt the configuration file with a user-supplied password either during the creation or at a later point in time. When the password is set, its (unsalted) SHA-256 hash value is used for producing the secret key for the encryption algorithms (XSalsa20 and Poly1305).

If an encrypted rclone.conf file is found, which can be easily identified by its header as depicted in Fig. 3, the only possibility to recover the key is through memory forensic. In the following we discuss our findings based on the newest pre-compiled version (Linux Intel/AMD - 64 v1.56.1) installed on a Debian 10 (kernel ver. 4.19.0) PC.

There are two occasions to enter a password: (1) when a user sets the password for the first time or (2) when a user wants to run any command (e.g., `rclone copy`). To observe how the entered password and the decrypted rclone.conf file are stored in memory, we analyzed rclone's and traced the stack/heap changes. Since the

[3] https://rclone.org/docs/#config-config-file (all footnotes were last accessed on 2022-08-16; date is omitted in the remaining footnotes).
[4] https://madmurphy.github.io/libconfini/html/libconfini.html
[5] https://rclone.org/commands/rclone_obscure/

```
[germanFTP]
type = ftp
host = 11111.server.de
user = ftp11111-rclone
pass = QWi3M1kjZ370FQHzqdKO9zTLNvPz4neFX-CRWwVnpNrbDPQk
tls = false
explicit_tls = true

[crypto]
type = crypt
remote = germanFTP:
password = MgyzqDmYgfxbzzNuNGgG5yMUkmQDiPSAAAAAAAA
password2 = MyxFf3EdFJh_Mi30OZ2KXfUSAAAAAAA

[case03_Dropbox]
type = dropbox
token = {
  "access_token":"sl.A4mVX4se7_vZNfBHk27XkDbJf3nGT7o
     7J56bC1hOULqL2_qPCaRI2rtDrR6d4a-yiihj5Zy9GynnLcaiW",
  "token_type":"bearer",
  "refresh_token":"d3I-aU8GiJgAAAAAAA5xUPuEwsu47InG0
     QyW6FauCS9Uva2qbWua",
  "expiry":"2021-09-09T01:29:03.222131+02:00"
}

[pi]
 ...
```

Fig. 2. Rclone configuration file content for Dropbox; entries have been modified/shortened for better readability and not reveal credentials.

```
# Encrypted rclone configuration File

RCLONE_ENCRYPT_V0:
yVUs6uMot760oBntHBH2tNT43KyX27jj ...
```

Fig. 3. Encrypted rclone.conf opened in a regular text editor.

executable is stripped, we list the names and offsets of the relevant functions for analysis in Table 1 where the prompt-column describes what is shown on the screen, the layer-column specifies callers and callees (e.g., 02 function is the callee of the 01 function right above it), followed by the corresponding function and offset.

1. To set a new password, `SetPassword()` is called when user selects 's', and `SetPassword()` calls `changeConfigPassword()` when user selects 'a', which then invokes the `ChangePassword()` and the `SetConfigPassword()`.
2. If a password has been already set, the user must enter the password for conducting any operation. Thus, rclone invokes the `getConfigPassword()` to prompt 'Enter configuration password:' on screen and then calls the `GetPassword()` which itself calls the `readPassword()` to load the input password. When these functions return, `SetConfigPassword()` uses the password to generate the secret key for decrypting rclone.conf.

We found that in both situations `SetConfigPassword()` is called which reformats the password to `[password][rclone-config]` (compare Fig. 4) and then calculates the SHA256 hash of this string which acts as the secret key. During this procedure, the cleartext string and the hash value are both stored in the memory, and according to our test, the `[rclone-config]` string is an effective indicator for locating the password in the memory dump. Depending on how many times rclone was executed/the password was entered before taking the memory dump. There may be multiple copies of the string retained in the memory dump. For instance, searching for `[rclone-config]` in the memory dump where the rclone process terminated, the password `testpwd` was successfully recovered (an example is shown in Fig. 5).

In addition to the `[rclone-config]` string, we found two other indicators that are sometimes near the password which are 1) `e/n/d/r/c/s/q`, the string at the bottom of the main menu for command `rclone config`, and 2) `[THEFIRSTREMOTE]` (`THEFIRSTREMOTE` is the name of the first remote in the configuration file, e.g., `germanFTP` in Fig. 2).

3.3. rclone.conf fragments recovery

If the password recovery fails, e.g., because the memory page was overwritten, it may be still possible to recover (parts of) the unencrypted content of rclone.conf. As mentioned, the rclone.conf is in the INI format utilizing pre-defined keys, fields, and the section names which could be used to locate the cleartext content in memory. For instance, Fig. 2 shows the configuration template defined for Dropbox where strings such as `type =`, `token =`, "`access_token`":, "`token_type`":, "`refresh_token`":, etc. may be found in a memory dump and are also used in templates for other storage types. More templates are provided in the `docs/content/CLOUDNAME.md` in rclone's github repository where `CLOUDNAME` is a placeholder for the different types such as 'onedrive' or 'dropbox'.

According to our test, the complete file, or fragments of it could be found by searching these keywords in the dump. An example is shown in Fig. 6 where we searched for `token =` in the dump and found the `access_token` for a OneDrive remote at address `0x665c5020`. However, the access token was broken at address `0x6959ec40` and we could not find the next field of `access_token`. To find the missing part of the access token and the rest of the rclone.conf file, we continued by searching for `token_type` which revealed the rest of the access token at address `0x26d798fa`. Note, while the fragments matched perfectly in this case, they could also be overlapping. In this case, an investigator must remove the overlapped data in terms of the possible length of the field and the structure of the template. If the recovery fails to restore the complete rclone.conf file, the snippets (such as access token and user credentials) found in the fragments may still be sufficient to access the CSP.

3.4. Rclone remote encryption

In addition to encrypting the configuration file, rclone allows to create 'crypt remotes' which are virtual remotes adding a layer of encryption on top of an existing remote as depicted in Fig. 7 (a sample configuration is shown in Fig. 2 `[crypto]`). To utilize a 'crypt remote' one first creates the regular remote (e.g., an FTP remote) followed by the crypt wrapper. Then, instead of copying data/accessing the remote directly (1), one connects to the crypt (2) which internally connects to the FTP (3) and access the data. As 'crypt' is a remote itself, it is possible to stack them and have two or more layers of encryption.

When setting up a crypt remote, the user is given several security relevant choices:

> Filename encryption can be set to full filename encryption (default), simple obfuscation, or none which only changes the extension to `.bin`.
> Directory name encryption can be set to true (default) and false (but will only work if filename encryption is activated).
> Password is the passphrase used to encrypt the data.
> Salt can be generated for strengthening the file encryption, which can be set by a user or rclone (by default).

Rclone utilizes NaCl SecretBox which is based on XSalsa20 cipher and Poly1305 for integrity which is state-of-the-art encryption and secure (Zinzindohoué et al., 2017). The algorithm divides the input into chunks of 64 KiB, encrypts them, and lastly creates a checksum of 16 bytes that is appended to each chunk. After reassembling the chunks to one file, a header consisting of an 8-byte

Table 1
Function call stack for the rclone commands that can cause a user entering the rclone.conf file's password.

Command	Option/prompt	Layer	Function	Offset
`rclone config`	"s) Set configuration password"	01	SetPassword()	`0x8923c0`
	"a) Add password"	02	changeConfigPassword()	`0x88b220`
		03	ChangePassword()	`0x0928a0`
		03	**SetConfigPassword()**	`0x88aae0`
`rclone [command]`	"Enter configuration password:"	01	getConfigPassword()	`0x89a9c0`
		02	GetPassword()	`0x8926e0`
		03	readPassword()	`0x888b60`
		02	**SetConfigPassword()**	`0x88aae0`

magic number `RCLONE\x00\x00` (blue) and a 24-byte random IV (Initialization Vector, orange) are prepended as shown in Fig. 8.

Rclone also offers the possibility to use a salt which is a second password (not stored with the encrypted content) and used to permute the encryption key. If no salt is provided, a default salt `A80DF43A8FBD0308A7CAB83E581F86B1` is applied (see on GitHub path/rclone/backend/crypt/cipher.go). Lastly, rclone supports file/folder name encryption, which uses EME (Halevi and Rogaway, 2004) with a 256-bit key.

Decrypt remotes: In a situation where the cloud authentication data in the rclone.conf is expired but the encrypted files are accessible (e.g., a local copy is found or the CSP provides them), one can still recover the data if the 'password' and the salt (i.e., 'password2', not set by default) are found. Therefore, first de-obscure the passwords running `rclone reveal`. Second, store the encrypted files locally (or on a remote storage). Third, create a remote `local disk` (local, option 22) which is not a remote but a reference to the local hard drive. Lastly, create a new crypt remote pointing to the local disk and provide the found credentials. Alternatively, forensic investigators can generate the key with the `Key(password, salt string)` function and then decrypt the files with `DecryptData()` (in the source code file backend/crypt/cipher.go. Note, to the best of our knowledge, we are the first describing this procedure which therefore can be seen as a minor contribution.

In addition, we found the following that could be interesting for forensic investigation:

- If no salt is set (default), the file and folder names are only encrypted with the user password. In return, this means that identical file/folder names on different crypt remotes will be identical if they have the same user password. Note, names only sharing a common prefix or substring will not have this commonality after encryption.
- For the encryption of the file content, an initialization vector/salt is used which means that identical files are slightly larger than their unencrypted version and that identical cleartext files result in different encrypted files.

One can calculate the encrypted file size (FS) given the original-text file size by $FS_{crypt} = FS_{org} + \lceil FS_{org}/2^{16} \rceil * 16 + 32$.

In case an encrypted file is found, one can calculate the original file size as follows: $FS_{org} = FS_{crypt} - (\lfloor FS_{crypt}/2^{16} \rfloor * 16 + 32)$

which provides the correct result in most cases or following these two steps for the definite answer:

$$x = FS_{crypt} - (FS_{crypt}/2^{12} + 32)$$

$$FS_{org} = FS_{crypt} - (\lfloor x/2^{16} \rfloor * 16 + 32)$$

```
[----------------------------------code----------------------------------]
   0x88abe0:    lea    rax,[rip+0x15c4c14]        # 0x1e4f7fb
   0x88abe7:    mov    QWORD PTR [rsp+0x28],rax
   0x88abec:    mov    QWORD PTR [rsp+0x30],0x10
=> 0x88abf5:    call   0x455200
   0x88abfa:    mov    rax,QWORD PTR [rsp+0x38]
   0x88abff:    mov    rcx,QWORD PTR [rsp+0x40]
   0x88ac04:    lea    rdx,[rsp+0xb0]
   0x88ac0c:    mov    QWORD PTR [rsp],rdx
No argument
[----------------------------------stack-----------------------------------]
0000| 0xc00063f700 --> 0xc00063f790 --> 0xc00063f800 --> 0x0
0008 1)0xc00063f708 --> 0x214e208 --> 0x7b0000005b ('[')
0016 2)0xc00063f710 --> 0x1
0024 3)0xc00063f718 --> 0xc000560ce0 --> 0x64777074736574 ('testpwd')
0032 4)0xc00063f720 --> 0x7
0040 5)0xc00063f728 --> 0x1e4f7fb ("][rclone-config]^[^#]*[#]+[^#]*$^flags:\\s+(\
buseafter object keyallownoe"...)
0048 6)0xc00063f730 --> 0x10
0056| 0xc00063f738 --> 0xc00063f768 --> 0xc00063f800 --> 0x0
[--------------------------------------------------------------------------]

1) Address of the string '['.
2) Length of the string at 0x214e208.
3) Address of the string 'testpwd'.
4) Length of the string at 0xc000560ce0.
5) Address of the string '][rclone-config]'.
6) Length of the string at 0x1e4f7fb.
```

Fig. 4. The assembly code snippet of the `SetConfigPassword()` function that is about to construct the `[password][rclone-config]` string.