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

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.

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

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

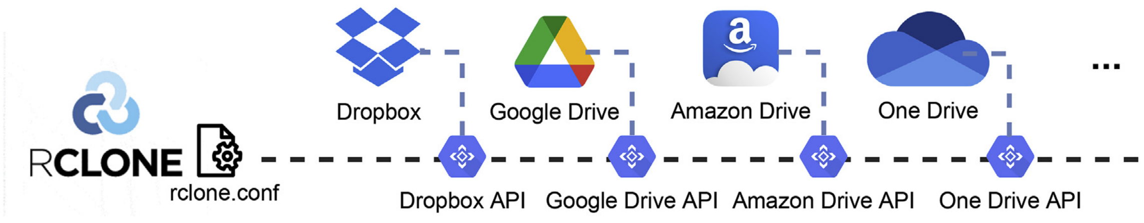


Fig. 1. Rclone authentication.

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 = MgyzqDmYgfxbzzNuNGgG5yMUkmQDiPSAAAAAAA
password2 = MyxFf3EdFJh_Mi300Z2KXfUSAAAAAAA

[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.

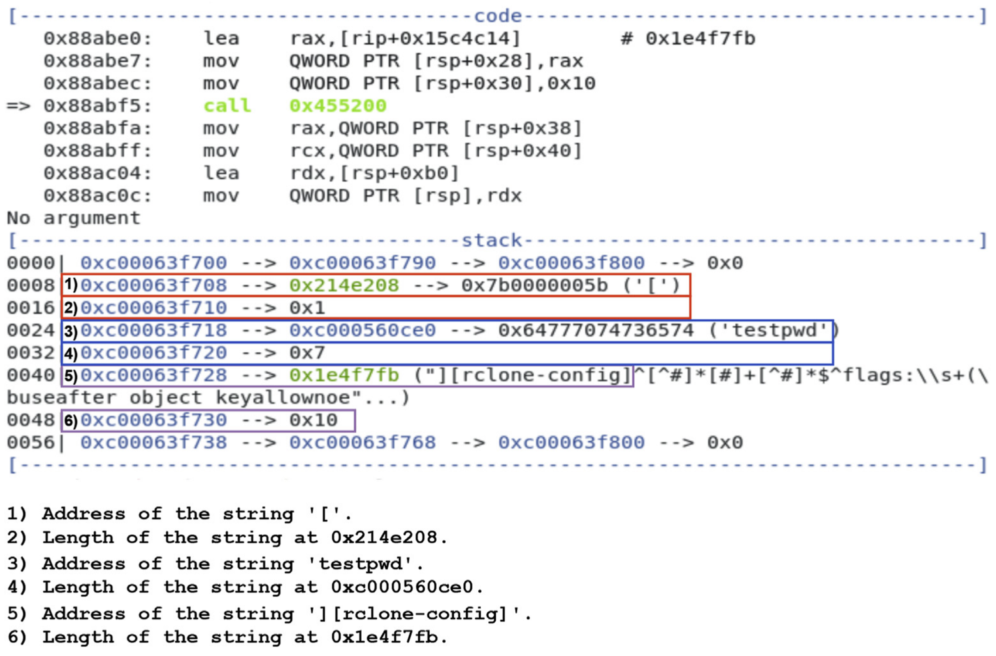
One can calculate the encrypted file size (FS) given the original-text file size by $FS_{crypt} = FS_{org} + \lceil FS_{org}/2^{16} \rceil * 16 + 32$.

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:

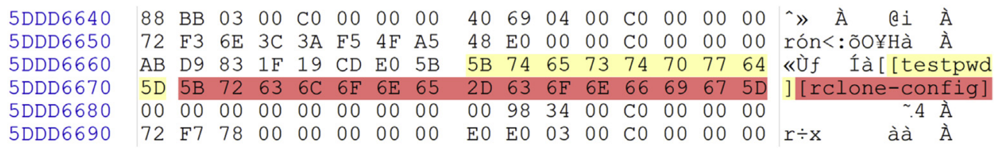
$$x = FS_{crypt} - (FS_{crypt}/2^{12} + 32)$$

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



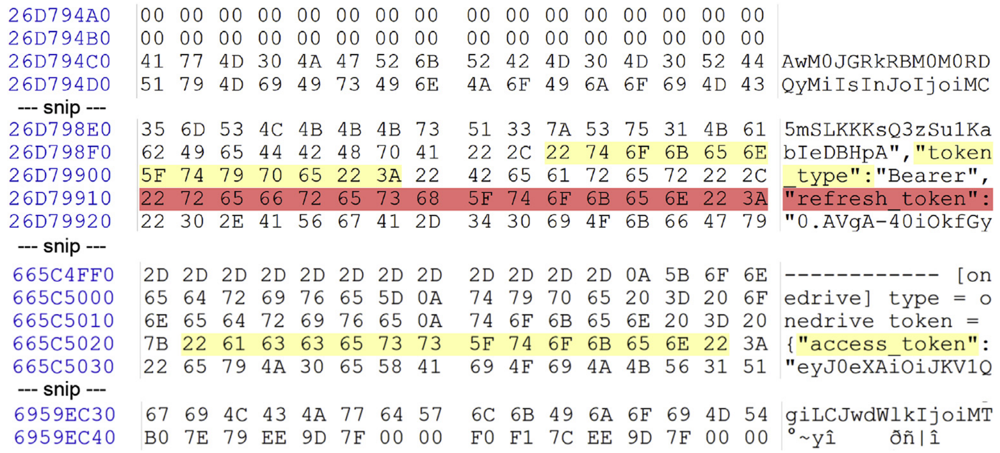
```
[------------------------------------code------------------------------------]
   0x88abe0:    lea     rax,[rip+0x15c4c14]        # 0x1e4f7fb
   0x88abe7:    mov     QWORD PTR [rsp+0x28],rax
   0x88abec:    mov     QWORD PTR [rsp+0x30],0x10
=> 0x88abf5:    call    0x455200
   0x88abfa:    mov     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.



```
5DDD6640  88 BB 03 00 C0 00 00 00  40 69 04 00 C0 00 00 00  ˆ»  À   @i  À
5DDD6650  72 F3 6E 3C 3A F5 4F A5  48 E0 00 00 C0 00 00 00  rón<:õO¥Hà  À
5DDD6660  AB D9 83 1F 19 CD E0 5B  5B 74 65 73 74 70 77 64  «Ùƒ  Íà[[testpwd
5DDD6670  5D 5B 72 63 6C 6F 6E 65  2D 63 6F 6E 66 69 67 5D  ][rclone-config]
5DDD6680  00 00 00 00 00 00 00 00  00 98 34 00 C0 00 00 00           ˜4 À
5DDD6690  72 F7 78 00 00 00 00 00  E0 E0 03 00 C0 00 00 00  r÷x     àà  À
```

Fig. 5. The password found by searching for the string `[rclone-config]` through the dump.



```
26D794A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
26D794B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
26D794C0  41 77 4D 30 4A 47 52 6B  52 42 4D 30 4D 30 52 44  AwM0JGRkRBM0M0RD
26D794D0  51 79 4D 69 49 73 49 6E  4A 6F 49 6A 6F 69 4D 43  QyMiIsInJoIjoiMC
--- snip ---
26D798E0  35 6D 53 4C 4B 4B 4B 73  51 33 7A 53 75 31 4B 61  5mSLKKKsQ3zSu1Ka
26D798F0  62 49 65 44 42 48 70 41  22 2C 22 74 6F 6B 65 6E  bIeDBHpA","token
26D79900  5F 74 79 70 65 22 3A 22  42 65 61 72 65 72 22 2C  _type":"Bearer",
26D79910  22 72 65 66 72 65 73 68  5F 74 6F 6B 65 6E 22 3A  "refresh_token":
26D79920  22 30 2E 41 56 67 41 2D  34 30 69 4F 6B 66 47 79  "0.AVgA-40iOkfGy
--- snip ---
665C4FF0  2D 2D 2D 2D 2D 2D 2D 2D  2D 2D 2D 2D 0A 5B 6F 6E  ------------ [on
665C5000  65 64 72 69 76 65 5D 0A  74 79 70 65 20 3D 20 6F  edrive] type = o
665C5010  6E 65 64 72 69 76 65 0A  74 6F 6B 65 6E 20 3D 20  nedrive token = 
665C5020  7B 22 61 63 63 65 73 73  5F 74 6F 6B 65 6E 22 3A  {"access_token":
665C5030  22 65 79 4A 30 65 58 41  69 4F 69 4A 4B 56 31 51  "eyJ0eXAiOiJKV1Q
--- snip ---
6959EC30  67 69 4C 43 4A 77 64 57  6C 6B 49 6A 6F 69 4D 54  giLCJwdWlkIjoiMT
6959EC40  B0 7E 79 EE 9D 7F 00 00  F0 F1 7C EE 9D 7F 00 00  °~yî    ðñ|î
```

Fig. 6. Sample fragments of the rclone.conf file recovered from memory dump.

3.5. Network analysis

To analyze the network traffic, we set up six remotes (FTP, FTPes,[6] SSH/SFTP, Gdrive, Webdav, Dropbox), captured the network traffic that is generated when executing

```
$ rclone lsd remote:
```

and analyzed it manually. In summary, it can be said that reliably identifying rclone-connections within network traffic is almost impossible except for SSH/SFTP.

Commonly CSPs utilize the port 443 for their APIs as this port compatible with firewalls. In our test, Gdrive, Dropbox and Webdav operated on port 443. Consequently, the network traffic shows the TCP handshake followed by the TLS session negotiation. Similarly, the FTPs (explicit TLS) connection contacts the server on port 21 and then sends the AUTH TLS request moving to an encrypted session. Besides the destination IP, there is no indication that these are rclone connections to a remote storage. Even within the regular FTP connection (unencrypted), we did not see any hints indicating that this is a rclone connection. Only for the SSH/SFTP connection, the 'client' package includes the name rclone as well as the version as shown in Fig. 10 row 4.

JA3: As connections are established on port 443, we also used TLS Fingerprinting with JA3 (Althouse, 2019)[7] which extracts parameters such as SSLVersion, Cipher, SSLExtension, EllipticCurve, and EllipticCurvePointFormat to generate a fingerprint for a particular client (see Fig. 9). Unfortunately, each connection (FTPes, Webdav, GDrive and Dropbox) resulted in different fingerprints. However, checking these fingerprints against ja3er.com/form – a JA3 SSL Fingerprint database – allows to identify the connection as `Go-http-client/1.1` (among others).

[6] FTP over explicit SSL/TLS (FTPES): the client explicitly requests to upgrade the connection to TLS sending `Request: AUTH TLS` after receiving the `FTP server ready`.

[7] As we run into problems with the original implementation from salesforce provided on https://github.com/salesforce/ja3, we utilized the GoLang implementation from https://github.com/dreadl0ck/ja3.

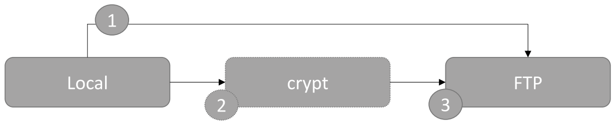


Fig. 7. Comparison of directly using a remote FTP (1) vs. using a virtual crypt remote (2) which then will access the actual FTP remote (3).

3.6. Proposed recovery methodology

Since the configuration may be encrypted, we propose the following methodology for recovery:

1. If the system is still running, try to acquire a memory dump.
2. For a given disk, locate the rclone.conf file which may require extra effort if the file is not stored in the default location. A customized location is usually stored in the environment variable `RCLONE_CONFIG`.
3. If the configuration is not encrypted or user credentials are found, use a forensic workstation to download (and decrypt) the files stored on the CSP(s).
4. If the rclone.conf file is encrypted, use the memory image for recovering the password as described in Sec. 3.2.
5. If the password of rclone.conf cannot be found in the memory dump, one may try to recover the (decrypted) fragments of the configuration file from memory as outlined in Sec. 3.3.
6. If the access in the rclone.conf file is no longer valid but there are rclone-encrypted files found on another computer, the investigator must decrypt the files manually by using the key and salt (see Sec. 3.4).

4. Rclone as a forensic tool

Rclone provides forensic investigators with a high-level file

```
0000: 5243 4c4f 4e45 0000 b5a3 42c8 b32b c8ad  RCLONE....B..+..
0010: 6da0 7132 80aa 8e1e 42d2 9b14 eea9 781b  m.q2....B.....x.
0020: f5e2 cfa9 fc54 902f eeb0 722d 798f b5d3  .....T./..r-y...
```

Fig. 8. Beginning of an encrypted file displayed using `$ xxd enc_file | less` (column one shortened for readability).

```
ftpes     "df669e7ea913f1ac0c0cce9a201a2ec1"
webdav    "e564ee1b7bcae4467d8c759df910ed9c"
gdrive    "df669e7ea913f1ac0c0cce9a201a2ec1"
dropbox   "d0ee3237a14bbd89ca4d2b5356ab20ba"
```

Fig. 9. JA3 TLS fingerprints for the rclone client for various remotes.

system to manage evidence across different cloud services. Prior to its application, it is necessary to have access to the user credentials or a non-expired access token issued by the CSP. Note, there may be additional challenges or consequences. For instance, a user may receive an email notification by the CSP that a novel client requests access or if 2-Factor-Authentication (2FA) is enabled, access to the linked device is needed. These are general acquisition challenges and unrelated to rclone which is why they are not addressed in this work.

The next section explains the *copy and mount* commands and is followed by two sample use cases which are based on rclone version 1.56.1 (no 2FA). In Sec. 4.3, we present the results of some practical acquisition tests followed by some shortcomings we observed. The last subsection is a discussion on the viability of rclone as a forensic tool.

4.1. Copy and mount

To acquire a remote storage, one may use the `copy` or `mount` commands. Using copy, the investigator runs the risk of swapping source and destination and accidently copy data to the remove drive. Therefore, we recommend using `—dry-run` to simulate the outcome. If copying is not an option, e.g., due to the size, one can mount the remote where rclone provides a `—read-only` flag. There are other possibilities as well, e.g., using the `—backup-dir` flag or `sync` command but they have a higher likelihood to make a mistake like accidently modifying/deleting data, and are not discussed.

Read-only: Interestingly, when having a closer look at the read-only flag, we realized that rclone does not rely on the CSP APIs but uses system functionality to prevent accidental writing. In detail, we analyzed sup-processes of rclone using `$ lsof -Fn -p [rclone pid]` (MacOS) and realized that the rclone-process use `/dev/macfuse` meaning that it uses system functionality to mount the network drive. Running the `$ mount`-command returned `ftp-test: on /Users/XXX/Desktop/rclone/mount (macfuse, nodev, nosuid, read-only, synchronous, mounted by XXX)`. Lastly, we tried to copy content to the mounted share via the GUI drag-and-drop as well as the terminal using the regular `cp`-command (not rclone copy)

```
$ cp test.file ~/Desktop/rclone/mount/
cp: ~/Desktop/rclone/mount/test.file: Read-only file system
```

Both attempts failed; the local OS will block attempts to write to cloud storage. This is safer than using `—dry-run` on commands that could corrupt cloud storage if used incorrectly.

4.2. Sample cases using rclone

Case 1: A forensic examiner found user credentials and assumes that these credentials also work for the suspect's Dropbox as well as FTP account. The interactive `rclone config` is used to create the config file.

1. $ rclone config followed by `n` for new remote.
2. When prompt for the name, we will use ‘case03x_FTP’ (case sensitive).
3. From the list of storage types, we utilize 13 (FTP Connection) and provide `host`, `user`, and `port`.
4. When asked for the password, we choose ‘yes type in my own password’.
5. As the FTP server does not support implicit FTPS, we chose ‘false’ but answer ‘true’ for explicit_tls to not transfer data unencrypted.

The final configuration will look like the sample provided in Sec. 3.1. Now that the remote has been created, it can either be mounted or fully copied. Due to the size of the FTP (we assume 2 TB), it is only mounted

```
$ rclone mount case03x_FTP: mnt/ -read-only
```

and relevant folders are copied using the provided OS tools

```
$ cp -r mnt/images/ /case03x/FTP
```

Alternatively, one may use rclone to copy

```
$ rclone copy case03x_FTP:/images /case03x/FTP
```

Note, the `copy` command will directly access the remote (not the read-only mounted directory). Thus, if users confuse the order, they will write to the FTP.

Case 2: Setting up Dropbox is slightly different as OAuth2 is used but rclone navigates the user through the process. We again use the interactive CLI and start with new remote and set our name to ‘case03x_Dropbox’.

1. Follow the default options (no client_id, no client_secret, no ‘Edit advanced config?’ and yes to ‘use auto config’) which brings the user to the Dropbox website.
2. Now, username and password are required to allow rclone to access Dropbox which we assume have been found by other means.[8]
3. Back in the terminal, one can see the OAuth2 token, and we acknowledge with ‘yes’.
4. As the suspect uses the Dropbox basic plan (2 GB storage), it can be directly copied
   ```
   $ rclone copy case03x_Dropbox: case03x/Dropbox
   ```

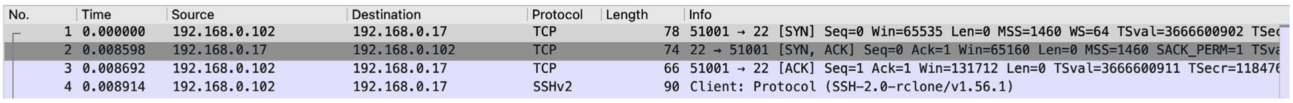
Note, the configuration file for Dropbox will be different compared to FTP as it contains the OAuth2 token instead of credentials.

4.3. Storage acquisition

Given the large amounts of data and files that may be stored in the cloud, acquisition performance is crucial. However, a comparison is not straight forward as CSP applications may maintain local copies of files, are accessed through a web interface, or it is difficult to measure the exact time it takes to download. For instance, files stored on Dropbox are also available on the local hard drive.[9] Consequently, in the following we perform several practical tests

[8] To remove the App from Dropbox, go to settings → connected Apps.

[9] We assume the free Dropbox account that currently does not support smart sync see https://www.dropbox.com/smart-sync.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.102	192.168.0.17	TCP	78	51001 → 22 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=3666600902 TSec
2	0.008598	192.168.0.17	192.168.0.102	TCP	74	22 → 51001 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSva
3	0.008692	192.168.0.102	192.168.0.17	TCP	66	51001 → 22 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=3666600911 TSecr=118476
4	0.008914	192.168.0.102	192.168.0.17	SSHv2	90	Client: Protocol (SSH-2.0-rclone/v1.56.1)

Fig. 10. Screenshot of Wireshark showing the first packages when rclone establishes a connection via SSH/SFTP.

with respect to efficiency, completeness, and correctness, to have a point of reference.

Efficiency: From the list of supported storage types, we decided to compare rclone's acquisition efficiency to FTP, SSH/SCP, Dropbox, and GDrive, where the question is: Can rclone achieve similar download rates than common tools? To compare against FTP/SCP/Dropbox, we utilized a 100 MiB file, downloaded it five times and provide the average duration. The utilized FTP client was from inetutils 2.2 on MAC OS where we first logged in and then used the `get`-command to download the file. With respect to rclone, we created the remote interface following the interactive CLI and then used

```
$ time rclone copy ftp-simple:/100MiB ./test/
```

On average, the FTP client required 8.42sec while rclone finished in less than half the time (3.90sec). Mounting the remote read-only and performing a copy achieved similar times

```
$ rclone mount ftp-simple:/ /mountpt/
$ time cp /mountpt/100MiB ./test/
```

For SSH/SCP we again navigated the interactive CLI to configure rclone using a public key. The two commands used were

```
$ time rclone copy sshpi:100MiB rclone-test/
$ time scp -i key_rsa www@pi:100MiB rclone-test/
```

where rclone required 10.91sec and SCP was slightly faster with 9.92sec on average. Next, we connected rclone to Dropbox and downloaded the file using

```
$ time rclone copy Dbox:100MiB rclone-test/
```

where the average time was 5.25sec. This time rclone was compared against Safari, i.e., HTTPs download through the website, where we manually measured the time: 8.1sec. For GDrive we tested using a 1 GiB file and like before against the web interface. On average, rclone needed 25.6s and the Safari download was completed in 25.2s. While it is hard to predict how rclone compares against all existing solutions, these rudimentary tests show that rclone has an acceptable throughput.

Accuracy: This test aimed to analyze correctness and completeness. We therefore downloaded our complete Dropbox drive via rclone which includes almost 9600 files totaling in 1.4 GB. The following command was used

```
$ time rclone copy dropbox: ~/dbox/
```

The overall time was 29m35s. We then compared it against our local Dropbox sync folder running

```
$ diff -qr Dropbox/ ~/dbox/
```

as well as the GUI application FreeFileSync[10] which allows to compare hashes. The comparison revealed 105 differences. Analyzing these differences showed that most files only exist locally and are never sync'd to the Dropbox account; consequently, they cannot be downloaded. For instance, the `.dropbox` and `.dropbox.cache` folders, `.DS_Store` (81 times), `Icon`[11] (7) and one temp file (~$eval.xlsx). Alias/symbolic Link (3) are sync'd by the Dropbox client application *but are modified*, i.e., the local copy is a proper alias (9 KB) where the online copies are empty files. Note, according to the documentation "rclone will [also] ignore symlinks or junction points (which behave like symlinks under Windows). If you supply `—copy-links` or `-L` then rclone will follow the symlink and copy the pointed to file or directory."

The remaining 11 mismatches are due to empty folders and files with special characters: By default, rclone does not sync empty folders (5) or folders that only contain empty folders. If they are needed, the user has to add the `—create-empty-src-dirs` option to the copy command. The last 6 entries are due to special characters (ö,ü,á) and are actually correctly copied. However, diff seems to consider them different due to their filenames. Example: diff lists the following

```
Only in ~/dbox/Automática e Informática.pdf
Only in Dropbox/Automática e Informática.pdf
```

However, comparing the SHA1 hashes and timestamps showed that these are identical files.

4.4. Rclone shortcomings

While performing our tests, we observed the following shortcomings of rclone:

Large files via FTP: First, when copying larger files (1 GiB) from the FTP, we had problems with rclone's multi-thread which is utilized by default. This caused incorrect downloads and the following error message: `ERROR : 1GB: Failed to copy: 450 Transfer aborted. Link to file server lost Attempt 3/3 failed with 1 errors and: 450 Transfer aborted. Link to file server lost`. To fix this, we had to add the `—multi-thread-streams 0` option. The download then took 21s which is similar to the GDrive test (see Sec. 4.3).

Timestamps: We realized that rclone does only preserve timestamps for files but not directories. After copying, all directories had the same stamp: the time of executing the copy command. In contrast, using the Dropbox client application or the web interface will preserve the original timestamp. According to Craig-Wood (2022) (rclone Bug page) this seems to be a common bug: "Rclone doesn't currently preserve the timestamps of directories. This is because rclone only really considers objects when syncing." We noticed that this also applies when doing a lsd (list directories only) which shows wrong timestamps. When mounted and copied, the regular file system rules apply, i.e., files and folders timestamps are updated.

CSPs specific features: Proprietary software offers features that rclone does not support. For instance, most clients and web interfaces allow to 'share folders' or have a trash that may still contain files. In addition, rclone relies on the API. Although we have not encountered this, there may be instances where not all files are visible via rclone.

Buckets and million files: According to Craig-Wood (2022), rclone also has difficulties if there are millions of files inside a directory or bucket. It also struggles with buckets as buckets do not utilize the concept of directories.

4.5. Discussion on rclone's prospects to be a forensic tool

While it seems a natural step to compare rclone to other

[10] https://freefilesync.org

[11] Icon? file is MAC specific and exists in directories that have a custom icon in the Finder.

acquisition possibilities such as native apps of the CSP or existing cloud forensics tools, there are several challenges. For a thorough comparison, one would have to subscribe to all CSPs, install their application, and perform a manual analysis of the acquisition efficiency as well as possible settings (e.g., is it possible to activate a 'read-only' mode). On the other hand, professional forensic tools are costly, and terms & conditions often prohibit research. Therefore, this section provides a comparison on theoretical level to answer the question if *rclone has the potential to act as a digital forensic tool?* We start with the two aspects raised in the introduction, ease of acquisition and error proneness, followed by some other points we deem relevant:

Ease of acquisition: Rclone supports a wide variety of CSPs, comes with a documented CLI and is platform independent. While there are certainly practitioners who favor a GUI over a CLI, the latter has a higher potential for automation, e.g., being integrated into existing software or proprietary scripts. The two cases, outlined in Sec. 4.2, are exemplary of how rclone could be used. In addition, it is a dynamic GitHub project with an active community around it ensuring (for the moment) that new CSPs will be added, and bugs will be fixed.

Error-proneness: Although rclone has not been developed for forensic purposes, it comes with an important feature: mount read-only. In addition, it provides a dry-run option as well as `-i/—interactive` flag which can be used in the beginning to avoid accidental data loss. Overall, we rate the error-proneness of using this application similar or even lower compared with existing approaches.

Connected apps/authentication attempts: Accessing remote storage via rclone requires that the CSP supports 'connected Apps' (depending on the CSP this terminology differs). Thus, on rclone's first connection attempt, the application is added to the connected devices (where it later can also be removed). Depending on the CSP, this may trigger a notification. For instance, when we connected to Google drive, we received an Email and a push notification to our phone.

Portability and security: As explained in Sec. 3.1, an investigator can access various cloud storages across platforms/workstations by carrying the rclone.conf file and the client software (instead of installing various cloud apps provided by different CSPs). The data and the config file can be encrypted which provides additional protection, i.e., leaked data will not reveal information.

Open-source: A general problem in digital forensics is that practitioners often (must) rely on 'black boxes' when doing their work especially for acquisition. As discussed by Ottmann et al. (2021), this poses a problem which is beyond the scope of this paper. In contrast to CSP applications and most digital forensic tools, rclone is open-source and thus an examiner can analyze the source code to understand how it exactly works. Of course, it is free of charge which is also a plus compared to other commercial tools.

5. Conclusion

This article examined rclone - an open-source tool for managing many common cloud storages such as Amazon Buckets, Dropbox, or Mega. We first investigated rclone and showed that an unencrypted configuration file provides value for forensic investigations. It was also demonstrated that in case an encrypted rclone.conf is encountered, that the encryption key (password) or the configuration file itself may be found in memory even if the rclone process has already terminated. In addition to the forensic analysis of rclone, we discussed rclone as a prospect to be a forensic tool. Specifically, we analyzed the read-only mount feature, presented two sample use cases on how rclone could be used for cloud acquisition, and looked at the performance as well as caveats of rclone. We conclude that rclone comes with some interesting features and can be valuable for forensic investigations but has the limitation that it does not maintain the directory timestamps.

Author contribution statement

Frank Breitinger: Conceptualization, Methodology, Validation, Investigation, Writing - Original Draft. **Xiaolu Zhang:** Methodology, Validation, Investigation, Writing - Original Draft. **Darren Quick:** Writing - Review & Editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Rclone overview

Rclone is an open-source GoLang program allowing to access and manage many common cloud service providers (CSPs): according to rclone.org, there are 55 supported providers/mechanisms at the time of writing this article making it the 'Swiss army knife of cloud storage'. It serves as a universal interface to access various CSPs from the command line using the `rclone` command. After installation, one first must create a new 'remote' which is the generic term used by rclone to describe all cloud storages. The creation of a remote can be done by executing

```
$ rclone config
```

which invokes the interactive CLI and walking a user through setting up all sorts of remotes. Before providing the access details (e.g., username, password), we first must set a unique remote name serving as an identifier for the storage, e.g., `germanFTP` or `businessDropbox`. After completing the setup, one can access the remote using the provided rclone commands in the format

```
$ rclone subcommand <parameters> <parameters...>
```

The full list of all commands can be found on rclone.org[12]; in the following we highlight some commands that we see relevant from an investigator's perspective.

Copy and mount: To acquire a remote storage, we recommend two procedures (detailed examples are provided in Sec. 4.2)

```
$ rclone copy remote: . —config=usr/rclone.conf
```

which will copy all files from the remote (remote needs to be replaced by the identifier, e.g., `germanFTP`) to the current directory (.) using the provided config file. Note, the colon after remote is essential and indicates that the root directory of the remote is copied. It is recommended to use `—dry-run` first to simulate the outcome. If copying is not an option, e.g., due to the size, one can mount the remote using

```
$ rclone mount remote:/pictures /mnt/ -read-only
```

which will use the default configuration file and mount the subdirectory `pictures` to the local path `/mnt/`.

Note, there are other possibilities as well, e.g., using the `—backup-dir` flag or `sync` command but they have a higher likelihood to make a mistake, e.g., modify/delete data, and will not be discussed.

Other helpful rclone commands: From a forensic/investigation perspective, the following commands may also be relevant

`rclone listremotes - List all the remotes in the config file.`

`rclone lsl - List all the objects in the path with size, mod time and path.`

[12] https://rclone.org/commands/

`rclone md5sum` - Produce an md5sum file for all the objects in the path.

`rclone sha1sum` - Produce a sha1sum file for all the objects in the path.

`rclone hashsum` - Produces a hashsum file for all the objects in the path.

`rclone size` - Return the total size and number of objects in `remote:path`.

Note that not every remote supports every command. For instance, SHA1 hashing is only supported by a handful of providers while MD5 is supported by about two-thirds of storage types. A comprehensive overview of supported features by remote can be found on the website.[13]

References

Ahmed, I., Roussev, V., 2019. Analysis of Cloud Digital Evidence. *Security, Privacy, and Digital Forensics in the Cloud*, p. 301.

Almulla, S., Iraqi, Y., Jones, A., 2014. A state-of-the-art review of cloud forensics. Journal of Digital Forensics, Security and Law 9, 2.

Alqahtany, S., Clarke, N., Furnell, S., Reich, C., 2015. Cloud forensics: a review of challenges, solutions and open problems. In: 2015 International Conference on Cloud Computing (ICCC. IEEE, pp. 1–9.

Althouse, J., 2019. TLS Fingerprinting with JA3 and JA3S. Retrieved 2022. https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967, 7 04.

Chung, H., Park, J., Lee, S., Kang, C., 2012. Digital forensic investigation of cloud storage services. Digit. Invest. 9, 81–95.

Costello, K., Rimol, M., 2021. Gartner forecasts worldwide public cloud end-user spending to grow 23% in 2021. https://www.gartner.com/en/newsroom/press-releases/2021-04-21-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-grow-23-percent-in-2021. Retrieved 2022-07-04.

Craig-Wood, N., 2022. Bugs and limitations. https://rclone.org/bugs/. Retrieved 2022-07-04.

Federici, C., 2014. Cloud data imager: a unified answer to remote acquisition of cloud storage areas. Digit. Invest. 11, 30–42.

Greetham, A., 2021. Detecting rclone – an effective tool for exfiltration. Retrieved 2022. https://research.nccgroup.com/2021/05/27/detecting-rclone-an-effective-tool-for-exfiltration/, 7 04.

Hale, J.S., 2013. Amazon cloud drive forensic analysis. Digit. Invest. 10, 259–265.

Halevi, S., Rogaway, P., 2004. A parallelizable enciphering mode. In: Cryptographers' Track at the RSA Conference. Springer, pp. 292–304.

Ottmann, J., Pollach, J., Scheler, N., Schneider, J., Rückert, C., Freiling, F., 2021. Zur blackbox-problematik im bereich mobilfunkforensik. Datenschutz und Datensicherheit - DuD 45, 546–552. https://doi.org/10.1007/s11623-021-1487-1. URL: 10.1007/s11623-021-1487-1.

Police Commissioners' Conference Electronic Crime Working Party, 2000. The virtual horizon: meeting the law enforcement challenges: developing an Australasian law enforcement strategy for dealing with electronic crime. Australasian Centre for Policing Research. http://www.police.govt.nz/resources/2001/e-crime-strategy/e-crime-strategy.pdf.

Quick, D., Choo, K.-K.R., 2014. Impacts of increasing volume of digital forensic data: a survey and future research challenges. Digit. Invest. 11, 273–294.

Quick, D., Martini, B., Choo, R., 2013. Cloud Storage Forensics. Syngress.

Ricci, J., Baggili, I., Breitinger, F., 2019. Blockchain-based distributed cloud storage digital forensics: where's the beef? IEEE Security & Privacy 17, 34–42.

Roussev, V., Ahmed, I., Barreto, A., McCulley, S., Shanmughan, V., 2016. Cloud forensics–tool development studies & future outlook. Digit. Invest. 18, 79–95.

Roussev, V., McCulley, S., 2016. Forensic analysis of cloud-native artifacts. Digit. Invest. 16, S104–S113.

Ruan, K., Carthy, J., Kechadi, T., Crosbie, M., 2011. Cloud forensics. In: IFIP International Conference on Digital Forensics. Springer, pp. 35–46.

Shariati, M., Dehghantanha, A., Choo, K.-K.R., 2016. Sugarsync forensic analysis. Aust. J. Forensic Sci. 48, 95–117.

Simou, S., Kalloniatis, C., Gritzalis, S., Mouratidis, H., 2016. A survey on cloud forensics challenges and solutions. Secur. Commun. Network. 9, 6285–6314.

Simou, S., Kalloniatis, C., Kavakli, E., Gritzalis, S., 2014. Cloud forensics solutions: a review. In: International Conference on Advanced Information Systems Engineering. Springer, pp. 299–309.

Stawski, S., 2018. Digital pollination: user impact on the document life cycle. URL: https://thesedonaconference.org/sites/default/files/conference_papers/%5B8.2%5D%20S.%20Stawski_Digital%20Pollination%20Paper_Oct%202018.pdf.

Zinzindohoué, J.-K., Bhargavan, K., Protzenko, J., Beurdouche, B., 2017. Hacl*: a verified modern cryptographic library. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, p. 1789, 1806.

[13] https://rclone.org/overview/#features