What's next, cloud? A forensic framework for analyzing self-hosted cloud storage solutions

Michael Külper, Jan-Niclas Hilgert, Frank Breitinger, Martin Lambertz

Angaben zur Veröffentlichung / Publication details:

Külper, Michael, Jan-Niclas Hilgert, Frank Breitinger, and Martin Lambertz. 2026. "What's next, cloud? A forensic framework for analyzing self-hosted cloud storage solutions." In *Digital forensics and cyber crime: 16th EAI International Conference, ICDF2C 2025, Miami, FL, USA, November 17–19, 2025, proceedings, part I*, edited by Kim-Kwang Raymond Choo, Alexander Perez-Pons, Himanshu Upadhyay, Rongxing Lu, and Kam Pui Chow, 241–63. Cham: Springer. https://doi.org/10.1007/978-3-032-22542-9_15.

What's Next, Cloud? A Forensic Framework for Analyzing Self-hosted Cloud Storage Solutions

Michael Külper[1(✉)], Jan-Niclas Hilgert[1], Frank Breitinger[2], and Martin Lambertz[1]

[1] Fraunhofer FKIE, Wachtberg, Germany
{michael.kuelper,jan-niclas.hilgert,martin.lambertz}@fkie.fraunhofer.de
[2] University of Augsburg, Augsburg, Germany
frank.breitinger@uni-a.de

Abstract. Self-hosted cloud storage platforms like Nextcloud are gaining popularity among individuals and organizations seeking greater control over their data. However, this shift introduces new challenges for digital forensic investigations, particularly in systematically analyzing both client and server components. Despite Nextcloud's widespread use, it has received limited attention in forensic research. In this work, we critically examine existing cloud storage forensic frameworks and highlight their limitations. To address the gaps, we propose an extended forensic framework that incorporates device monitoring and leverages cloud APIs for structured, repeatable evidence acquisition. Using Nextcloud as a case study, we demonstrate how its native APIs can be used to reliably access forensic artifacts, and we introduce an open-source acquisition tool that implements this approach. Our framework equips investigators with a more flexible method for analyzing self-hosted cloud storage systems, and offers a foundation for further development in this evolving area of digital forensics.

Keywords: Cloud Storage · Digital Forensics · Evidence Acquisition · Nextcloud

1 Introduction

Cloud storage has become a core element of modern computing, shaping how individuals and organizations store, access, and synchronize data across multiple devices. Its widespread adoption is driven in part by the integration of cloud services into major user ecosystems, as companies such as Microsoft, Google, and Apple bundle cloud storage with user accounts, effectively lowering the barrier to entry. In parallel, dedicated providers like Dropbox and pCloud offer free or low-cost plans with robust features, further expanding access to cloud-based storage solutions. As users increasingly access their data across multiple devices, cloud storage has also become a common element in modern forensic investigations [9].

While commercial cloud services dominate the landscape, the growing popularity of affordable, low-power computing platforms, such as the Raspberry Pi, combined with privacy concerns, has made self-hosted cloud storage solutions increasingly viable. Concurrently, regulatory pressures, including the General Data Protection Regulation (GDPR) and the EU-U.S. Data Privacy Framework, are motivating individuals and organizations to seek greater control over their data. These developments point to a trend: forensic investigations must increasingly consider not just client-side artifacts from commercial services, but also server-side data from privately hosted environments.

A range of self-hosted cloud storage platforms has emerged in response to this demand, including ownCloud, Nextcloud, OpenCloud, Seafile, and Pydio. While some have evolved into comprehensive groupware platforms with features like calendaring and communication tools, many continue to focus primarily on file storage and synchronization. This paper centers on the latter category. Although extended features such as calendars, address books, and communication tools (e.g., video, audio, and chat) are forensically significant, the core functionalities of file synchronization and sharing are common to many platforms and present a wide range of unique forensic considerations. These include the handling of timestamps, the attribution of user actions to digital artifacts, and the underlying reasons for a file's presence on a given device, each of which warrants focused examination.

Research into the forensic investigation of cloud storage applications is well established. Over a decade ago, Martini and Choo [11] proposed general guidelines for the forensic analysis of cloud platforms. To validate their framework, they conducted a case study on the cloud storage application ownCloud [12], defining categories of evidential data and artifact sources, and identifying relevant forensic artifacts. Building on this foundation, Teing et al. [25] extended the model to offer a more balanced view—addressing the prior emphasis on client-side analysis—and presented a case study on the Seafile application.

In this paper, we revisit and re-evaluate Teing et al. [25]'s forensic framework in the context of Nextcloud. We chose Nextcloud for three reasons: First, it is among the most widely used self-hosted cloud platforms, yet to date, no comprehensive forensic analysis has been published. Second, as a fork of ownCloud, Nextcloud enables direct comparison with the original case study by Martini and Choo [11], allowing us to examine the evolution of artifacts over more than a decade. Third, it allows for evaluating whether cloud storage forensic frameworks remain relevant and identifying needed updates for modern platforms.

To this end, our contributions include:

- a comprehensive forensic analysis of the Nextcloud client and server applications;
- an exploration of under- and unexplored aspects relevant to modern forensic investigations in cloud storage environments;
- a review of existing cloud storage forensic frameworks, accompanied by a revised framework that addresses identified gaps;

– the release of an open-source forensic tool that facilitates API-based data acquisition from Nextcloud environments.

2 Related Work

Martini and Choo [11] proposed an early framework for conducting cloud forensic investigations, building on the four phases of the forensic process as defined by NIST: Collection, Examination, Analysis, and Reporting. Their model comprises the following phases: (i) Evidence Source Identification and Preservation, (ii) Collection, (iii) Examination and Analysis, and (iv) Reporting and Presentation. The framework is iterative, allowing newly discovered evidence sources to initiate a fresh cycle of investigation. To demonstrate their model, they conducted a forensic analysis of ownCloud [12], where artifact identification was closely tied to the specific functions and features of the application.

Quick et al. [21] extended this model by adding a preliminary phase, Commence and Preparation, to define the scope of the investigation, select appropriate tools, and coordinate necessary expertise. They also highlighted challenges such as server volatility and the risk of evidence tampering or deletion by suspects using unknown devices.

Building on these foundations, Teing et al. [25] introduced a seven-phase framework tailored to self-hosted cloud storage environments. This model refines the original four phases and introduces three new ones: Readiness (pre-investigation planning and understanding of the environment), Case Investigation Preparation (legal coordination, stakeholder engagement, and technical setup), and Investigation Review and Lessons Learned (post-investigation analysis for future improvement). Unlike the original framework, preservation is treated as a distinct step during the Collection and Preservation of Evidence phase. Their approach also underscores the importance of aligning evidence collection with the architectural and functional specifics of the cloud application under investigation.

Based on the frameworks, both Martini and Choo [12] and Teing et al. [25] analyzed the cloud storage applications ownCloud and Seafile to identify forensic artifacts. Apart from studies on applications that support self-hosting and allow for server-side analysis, there is also a range of research focused solely on the client side of cloud storage services hosted by cloud storage service providers. This includes works on Dropbox [6,10,13,14,19], OneDrive (formerly SkyDrive) [6,7,13,18], GoogleDrive [6,20], Box [6,13], iCloud [17], Amazon Drive [7,8], CloudMe [24], pCloud [1,4,16], IDrive [22,26], MEGA [5,15,26], Sync.com [2], FlipDrive [2], SpiderOak [16], and JustCloud [16].

3 Forensic Analysis of Nextcloud

This section presents the results of our forensic analysis of the Nextcloud ecosystem. Unlike previous approaches based on system architecture, our findings are

organized by forensic relevance—focusing on user behavior and actionable evidence. We identified seven key artifact categories: User Data and Accounts, Device Information, Files, Synchronization, User Activity and File Version, File Deletion and Trash, and File Sharing. The following subsections summarize the client and server analysis.

3.1 Experimental Setup

To identify forensic artifacts in Nextcloud, we analyzed both the Android client and a personal server instance.

Our client-side analysis focused on the official Nextcloud Android app (v3.31.1), chosen for its popularity (over 1M downloads) and open-source availability[1]. The app was executed on a rooted virtual device via Android Studio, enabling full access to internal storage and system-level activity. Forensic artifacts were primarily found in the internal SQLite database `filelist` (`/data/user/0/com.nextcloud.client/databases`), which contains 13 tables tracking file metadata, directories, and user interactions.

On the server side, we examined a long-running NextcloudPi instance (v23.0.0.10) used continuously for over four years. It supports a single user syncing across devices, mostly using default settings except for a relocated data directory. Forensic data was collected from the live system, but backup ZIP archives containing full SQL dumps (`nextcloud-sqlbkp_<date>.bak`, with 101 tables) are also available. While our analysis used the live instance, these .bak files provide a viable option for offline inspection—though they may not include the most recent data. Key configurations reside in `/var/www/nextcloud/config/config.php`, including database credentials, paths, and logging options such as `trashbin_retention_obligation`. Additionally, `nextcloud.log` offers a valuable source of system event records.

While our analysis focused on application-level artifacts, OS-level and web server logs were excluded. To validate the observed behaviors, we performed controlled experiments complemented by source code reviews.

3.2 User Data and Accounts

This section outlines how user data and account information are stored and managed on both the client and server sides of a Nextcloud installation.

Client-Side. The file `com.nextcloud.client_preferences.xml`, located in the `shared_prefs` directory of the internal storage, contained the app's account information. For self-hosted instances, the `select_oc_account` field stores an identifier in the format `<username>@<host>[:<port>]`.

Further metadata is stored in the internal `filelist` database. Initially, the `arbitrary_data` table may contain only a path to media folders. However, after login, multiple entries for users identified by the `cloud_id` value are added, including avatar information, user status, and other metadata.

[1] https://github.com/nextcloud/android.

The `capabilities` table stores server-related metadata linked to each account. While the database includes entries for all configured accounts, the XML file retains only the currently active one.

Stored Credentials and Authentication Artifacts. After login, the client receives an `appPassword` saved by Android's Account Manager in the `accounts_ce.db` database. Each Nextcloud account creates entries in the `accounts` and `auth_tokens` tables that store this password. It is used with the username for HTTP Basic Authentication in a base64-encoded header.

These credentials are significant for forensic analysis, enabling access to the same Nextcloud instance and linking network activity to specific devices. Additional metadata such as the server base URL, version, and user display name is found in the `extras` table, with separate entries for each account.

Additional account-related metadata, such as the server's base URL, version information, and the user's display name, is stored in the `extras` table of the `accounts_ce.db` database. In cases where multiple Nextcloud accounts are configured on the device, corresponding entries are maintained for each of them.

Server-side. Nextcloud's database includes several tables that store user-related information. A summary of these tables, along with the relevant forensic data they contain, is provided in Table 1.

3.3 Device Information

This section describes the types of device-specific information that can be identified to support the attribution of user activity to specific devices.

Table 1. Overview of Nextcloud User-Related Database Tables

Table	Description
`oc_users`	Contains a unique identifier (UID) for each user, referenced across multiple tables. Includes display name and password. Passwords are stored using a hashed format: `algorithm|hash`, generated by PHP's `password_hash()` function.
`oc_user_status`	Tracks user availability status (online, offline, away), along with optional custom messages and the timestamp of the last status change.
`oc_accounts_data`	Stores user profile information set during account creation. Includes structured fields like `displayname`, `address`, `phone`, and `twitter`.
`oc_accounts`	A simplified table containing only `uid` and `data`. The `data` column contains the same profile information in JSON format.
`oc_groups`, `oc_group_user`	Define and manage group memberships. Used to distinguish roles such as `admin` and `user`.
`oc_preferences`	Stores per-user application settings. Includes `userid`, `appid`, `configkey`, and `configvalue`. For instance, `core|lang` stores UI language; `login|lastLogin` stores the last login timestamp.
`oc_authtoken`	Stores authentication tokens for users. Upon login via client applications, two rows are created: one for the client and one for the browser session. Each row includes UID, `last_activity`, and user agent string.

Client-side. From the client-side, dedicated artifacts for identifying the device are generally redundant, as the analysis is already being conducted on the device itself. However, the application password, which is uniquely generated for each client during the authentication process, can serve as an implicit device identifier. This token enables the correlation of client activity with server-side logs and observed network traffic, effectively linking actions to a specific device instance. Note, however, that if a user logs out and logs back in, a new application password is issued, which prevents consistent device identification across sessions.

Server-side. The web interface under Settings » Personal » Security displays connected devices, including current and past sessions. This includes user agent strings that reveal device details such as phone models (e.g., Samsung SM-S926B), operating systems (e.g., Android, macOS), browsers (e.g., Firefox 137), and sync clients (e.g., DAVx5 or macOS dataaccessd). The same information can be retrieved via the database table `oc_authtoken` table by running `SELECT id, uid, name, last_activity FROM oc_authtoken;`. We observed that each time a new OAuth token is generated, a new row is added to this table. Additionally, the `nextcloud.log` file—which location is specified in `config.conf`—contains entries related to failed login attempts, providing further visibility into authentication activities.

3.4 Files

In this section, we describe where to find user data and metadata stored by Nextcloud.

Client-Side. On Android, Nextcloud maintains metadata for files and directories in a local SQLite database `filelist`. Each file or directory is represented by a record in the table `filelist`, which contains the following file metadata:

Hierarchy. Name, relative path to root directory, and a parent ID referencing the immediate parent. The root directory is represented as a special entry.
Content. MIME type (e.g., `DIR`, `text/plain`) and file size. File contents themselves are not stored.
Metadata. Timestamps for creation and modification (typically only *modified* is set, while *created* timestamp was often missing or set to zero). Includes permissions, an owner account string, and—if applicable—image dimensions in `metadata_size`.
Attributes. Flags indicating whether the item is marked as favorite, hidden, or currently downloading.
Identification. Each entry includes a `local_id`, `remote_id`, and a primary key.

The database is populated incrementally as the user navigates the file structure. As a result, it may reflect only a partial view of the remote file hierarchy at any given time. Deleted files may persist until the user revisits their parent directory, triggering a refresh. Consequently, any interaction with the app

after acquisition can modify or remove potentially valuable artifacts. Forensic practitioners should ensure the database is backed up before conducting further analysis.

In multi-account setups, all file and directory metadata is stored in the same `filelist` database. Entries are distinguished using the `file_owner` field, which contains the corresponding account identifier.

Importantly, our experiments indicate that deleting an account from the Nextcloud application does not immediately remove its associated metadata from the local database. Instead, existing records persist and are gradually overwritten as the app continues to operate. This behavior allows for the recovery of remnants related to previously configured accounts during forensic analysis.

Server-side. By default, Nextcloud stores user data in `/var/www/nextcloud/data/`, but the actual location is set by the `datadirectory` key in the configuration file. In our setup, this points to a different folder, referred to as `$DATADIR`. Each user has a subfolder named after their UID. `$DATADIR` also includes log files (`nextcloud.log`, `updater.log`), a `tmp/` directory for temporary data, and two system folders[2]: `updater-*`, which stores pre-update backups for rollback, and `appdata-*`, which holds app-specific data.

Nextcloud also supports external storage[3], which can refer to remote sources (SMB, FTP, S3, other Nextclouds) or local paths/drives. These are configured via the web GUI (Administration > External Storage) or the `oc_external_config` table (Table 2). The type (e.g., SMB, FTP) can be found in `oc_external_mounts`.

The `oc_filecache` table accelerates file access by caching metadata. It tracks filenames, paths, timestamps, sizes, and mimetypes[4]. Useful queries include:

- Recently modified files (within the last day, i.e., 86400 s):
  ```
  SELECT name, mtime FROM oc_filecache WHERE mtime > (UNIX_TIMESTAMP() -
  86400);
  ```
- Files matching `report.pdf`:
  ```
  SELECT fileid, path, size FROM oc_filecache WHERE name LIKE
  '%report.pdf%';
  ```
- Top 10 largest files:
  ```
  SELECT name, size FROM oc_filecache WHERE mimetype != 2 ORDER BY size
  DESC LIMIT 10;
  ```

3.5 Synchronization

To keep the client up to date with changes on the server, Nextcloud relies on a process called synchronization. For Nextcloud, this process is triggered by

[2] https://help.nextcloud.com/t/how-to-clean-up-nextcloud-data-appdata-and-updater/123746.

[3] https://docs.nextcloud.com/server/23/admin_manual/configuration_files/external_storage_configuration_gui.html.

[4] Mimetypes are mapped via `oc_mimetypes`; directories typically have `mimetype = 2`.

the client. This section covers what kind of synchronization metadata can be identified within artifacts.

Client-side. Each entry in the `filelist` table within the `filelist` database contains the following information regarding synchronization:

- `last_sync_date`: A millisecond Unix timestamp indicating the last time a file or directory was synced. For root directories of a share, this value seems to be set and stay at `0`.
- `last_sync_date_for_data`: A millisecond Unix timestamp indicating the last time the actual content of a file or directory was synced. This also applies to the root directory of a share.
- *ETag:* Instead of relying on differences in modification times, which can be unreliable in certain circumstances, Nextcloud uses ETags to track versions of files. These randomly chosen numbers are updated when a file changes and are thus used to signal the need for synchronization.
- `media_path`: This field is populated when a file is downloaded to the device and specifies the local file system path where the file is stored.

Manually refreshing the file view, such as by pulling down the screen, updates synchronization-related timestamps within the `filelist` database. Similarly, navigating through files and directories in the application also causes the synchronization timestamps of the corresponding entries to be updated, even if no file is downloaded or modified.

Downloads. Nextcloud allows users to download individual files or entire directories for offline access. In the `filelist` table, such files can be identified by the presence of a value in the `media_path` field, which points to the file's local storage path on the device. The default download location can be configured within the application's settings and is stored in the shared preferences XML file under the key `storage_path`, for example: `/storage/emulated/0/Android/media/com.nextcloud.client`.

The application creates a subdirectory named after the account identifier within the storage location. Downloaded content is saved there, maintaining the structure of the shared directory as seen by the user. When a file is downloaded, its `last_sync_date_for_data` timestamp is updated to reflect synchronization with the server.

Uploads. The Nextcloud application allows users to upload arbitrary files, including photos and videos captured directly from the device's camera. A record of all uploads performed by the application is maintained in the `list_of_uploads` table within the `filelist` database. Each entry includes the local source path (`local_path`), the remote target path on the server (`remote_path`), the file size, and a Unix timestamp (in milliseconds) indicating the completion time of the upload.

While an upload is still in progress, the file size is set to `-1` and the completion timestamp remains `NULL`. The associated account is recorded in the `account_name` column using the known account identifier format. In our experiments, entries in this table were not automatically removed and remained accessible for several weeks after the upload had completed.

Synchronized Folders. When selecting a directory, the Android application offers the option to 'Sync' it. In our experiments, this action resulted in the corresponding update of records in the `filelist` table, which belong to any files or subdirectories of the synced directory, similar to the automatic sync that is performed when a directory is opened. However, the sync is performed recursively for all layers of the hierarchy, e.g., when a new file is created or deleted multiple layers down compared to the synced directory. In case no changes are detected, the table is not altered.

If *two-way sync* is enabled for folders, it sets the `internal_two_way_sync_timestamp` value for the corresponding record in the `filelist` table from `-1` to `0` for the first time. Afterwards, the folder is synced in certain intervals from 15 minutes to 24 h as defined in the application's settings. The status as well as the interval can be found in the shared preferences XML file under the `two_way_sync_status` and `two_way_sync_interval` keys, respectively. Each time a synchronization is triggered, the timestamp stored in `internal_two_way_sync_timestamp` is updated to reflect the corresponding execution time.

Server-side. Since Nextcloud synchronization is primarily initiated and managed by client applications, limited synchronization metadata is directly available on the server side. However, information about recently synchronized files and folders can be inferred from database tables such as `oc_filecache` and `oc_activity`. As previously mentioned, the `oc_filecache` table indirectly reflects synchronization activity: any modification, upload, or update to a file or folder alters the `mtime` (modification time) and `storage_mtime` (storage modification time) fields. Therefore, to identify the ten most recently modified files, the following SQL query can be used: `SELECT name, path, mtime FROM oc_filecache ORDER BY mtime DESC LIMIT 10;`.

3.6 User Activity and File Versions

Nextcloud allows users to modify files, tag files, and keep different file versions after changes are made. Within the following we look what kind of information can be retrieved via the artifacts about this user activities.

Client-side. The Android application does not offer a dedicated interface for managing file versions. However, version-related events—such as the creation of a new version—are visible in the application's *Activities* view, which provides a summary of recent interactions including file creation, renaming, and

modification. For version events, the view includes a button to restore a previous version. In our testing, attempts to use this restore function consistently resulted in application crashes.

It is important to note that the activity data shown in this view is not stored locally on the device. Instead, it is retrieved dynamically from the server when the user opens the view. As a result, this information is not accessible for offline forensic analysis or in cases where server communication is no longer possible.

Server-side. File versions are accessible via the web portal under Details » Versions, or by querying the `oc_filecache` table with `path LIKE '%files_versions%'`. On disk, they are stored in the `datadirectory` under each user's `files_versions` directory, preserving the original folder structure and appending a UNIX timestamp to filenames (e.g., `File.docx.v1746040832`).

User activity is recorded in the `oc_activity` table, with key columns including `timestamp`, `type`, `user`, `affecteduser`, `subject`, and `subjectparams`. The `type` column indicates the general action category (e.g., `file_changed`), while `subject` and `subjectparams` provide more specific context.

We found 12 distinct `type` values (`addressbook`, `calendar`, `calendar_event`, `calendar_todo`, `file_created`, `group_settings`, `shared`, `card`, `file_changed`, `file_deleted`, `file_restored`, and `security`) and 34 unique `subject` entries. The distinction between `user` and `affecteduser` allows us to differentiate who performed an action and who was impacted by it. For example, if `userA` deletes their own file, the `subject` includes the suffix `_self`; if they delete a file belonging to `userB`, this suffix is absent.

File version and activity log retention is managed by the versions_retention_obligation » auto setting, which triggers automatic cleanup based on internal policies.

3.7 File Deletion and Trash

Similar to file versioning, Nextcloud retains deleted files in a trash bin for a defined period.

Client-side. Deleting a file using the Android application merely causes its entry to be removed from the `filelist` table. While the Nextcloud application allows users to browse through deleted files and also recover them, this information is not kept anywhere on the device itself. Furthermore, a downloaded and deleted file is immediately removed from the storage location on the device.

Server-side. When a file is deleted through the system interface, an entry in the `oc_activity` database table with the type `file_deleted` is created. The file is then moved to the trash directory located at `datadirectory/username/files_trashbin/files`. Similar to file versioning, deleted files are appended with a timestamp suffix (e.g., `screenshot.jpg.d1728237675`) to distinguish them from other entries. Metadata regarding the original file path is retained in

the oc_files_trash table. Notably, if a file is manually removed from disk (e.g., via the rm command), its corresponding entry may persist in oc_files_trash until a cleanup operation is triggered.

3.8 File Sharing

A core feature of cloud storage applications, is the ability to share files with other users or groups.

Client-side. Nextcloud allows users to share files and directories with other users as well as using a link. When a sharing link is created, an entry in the ocshares table in the filelist database is created. Most importantly, this entry contains the path of the shared file or directory, a Unix timestamp indicating the time the share was created, the share URL as well as the user who shared it. When sharing with a user, the entry contains the user's name in the shate_with [sic] column. For a shared file, the share_by_link attribute in the filelist table is set to 1.

Server-side. The web portal provides a tab that lists all shares, i.e., files shared with the user as well as files shared by a user. The same information can be found in the table oc_share, where the most relevant columns are share_type (in our testing, we only encountered the values 3 for link-sharing and 0 for sharing directly with another user), shared_with, which is NULL for link-sharing or the corresponding user otherwise, and stime, which is the timestamp when the share was created. Other information that can be found in the table is note (to recipient), password (if protected), the file name, expiration (if set), or the token, which is how link-shared files are accessed (e.g., https://pi-ip/index.php/s/3kXXKCtn7WyyNS3 the token is 3kXXKCtn7WyyNS3).

3.9 Artifact Comparison: Nextcloud vs. OwnCloud

The comparison with ownCloud is based on the server side on "Cloud storage forensics: ownCloud as a case study" [12] and on the client side on both this and "Mobile Cloud Forensics" [13] as both contain information about Android ownCloud artifacts.

On the server side, we were able to verify that most of the findings identified for ownCloud still apply to Nextcloud. One of the main differences, however, lies in the file sharing mechanism. Specifically, the oc_sharing table used in ownCloud has been replaced by a table called oc_share in Nextcloud. While the oc_sharing table in ownCloud contained only basic information—such as the file owner, the users the file was shared with, and their access rights (read-only or write access)—the oc_share table in Nextcloud includes more detailed metadata. As previously described, this includes the type of share (e.g., with another user or via a public link), an optional password for protected shares, and an expiration

date for time-limited access. Additionally, we identified several aspects that were not covered in the analysis of ownCloud, either because they did not exist at the time or were not examined. These include new database tables and changes in how the application handles file deletions.

Similar to the server side, many of the existing artifacts identified in ownCloud are still present in Nextcloud. Those that have changed appear to be associated with specific feature updates. One key difference lies in how passwords are handled. While both applications utilize the Android Account Manager API, the analysis revealed that ownCloud stores the username and password in plaintext. In contrast, Nextcloud stores a revocable `appPassword`, which is used for authentication instead. Moreover, the Nextcloud client introduces two additional tables in the `filelist` database (`arbitrary_data` and `capabilities`), which store user-specific and server instance–specific information, respectively.

Summarized we can see that a lot of the identified artifacts also exist for ownCloud in the same or a very similar way, as they are related to certain features that got reworked throughout update in features such as file sharing or authentication. It is unclear whether these changes may also be found in an investigation of an modern ownCloud instance or if their artifacts may differ.

4 Rethinking Cloud Storage Forensic Frameworks

Beyond the selected artifact classes, we identified additional aspects that existing research and frameworks have overlooked. This section highlights these underexplored areas and introduces an enhanced forensic framework for the analysis of cloud storage applications.

4.1 Underexplored Aspects

API Acquisition. A common feature of cloud storage applications is an API provided by the storage server, typically used by client or web applications to perform operations such as uploading, downloading, or managing data. In a forensic context, this API can offer a practical entry point for accessing server-side data, particularly in early investigative stages when the server's physical location is not yet known or accessible. Even in self-hosted scenarios, where the server might eventually be identified and examinable, the API provides an opportunity for timely and remote data acquisition.

Despite the ubiquity of such APIs, their forensic potential has received limited attention in existing research, even though they can provide access to server-side artifacts that may not be present on the client device. One of the few comprehensive studies exploring this area is the work by Roussev et al. [23], which evaluates services like Google Drive, Microsoft OneDrive, Dropbox, and Box, and emphasizes the benefits of API-based acquisition in forensic investigations.

When considering API-based collection, it is important to evaluate the possible impact on the server's state. Interactions through the API may alter or generate server-side artifacts, which can compromise the forensic soundness of subsequent analyses. This is especially relevant in self-hosted environments, where

the server might later become available for physical examination. Some API requests may also trigger logging, alerts, or other forms of administrative attention. Therefore, investigators should adopt a cautious and controlled approach, favoring selective and granular data acquisition over large-scale extraction, to reduce detection risks and preserve the integrity of the investigation.

Server Evidence Preservation Process. Another often overlooked aspect is the careful consideration required when preserving evidence from a live server instance. While a dead acquisition can eliminate the risk of user tampering or unintentional corruption of files and databases, the benefits of live forensics often outweigh this, particularly because it enables the retrieval of volatile information such as cryptographic keys and potentially active session data.

To reduce the risk of users modifying or destroying evidence during live acquisition, investigators may deauthenticate active users and block new logins. However, this can alert users and prompt evidence tampering. A more discreet approach is to display a generic maintenance message while disabling functionality, or to isolate the server from the network. While these methods help prevent interference, their likelihood of raising suspicion varies by context and relies on the investigator's judgment.

Monitoring. Maintaining system operation beyond an initial preservation and collection phase is crucial for capturing ongoing activities and uncovering additional forensic insights. Continuous monitoring provides significant value, particularly for server devices, where observation can reveal critical events such as the creation of new files, user logins from previously unknown devices, or rapid data deletions. These actions may highlight files of interest or indicate usage patterns that suggest targeted behavior directed at specific users or datasets. On the client side, monitoring is also vital. It can help identify when shared files with the inspected user are deleted, when access permissions are revoked, or when new data is shared. Furthermore, API-based monitoring is beneficial for retrieving data and metadata that may not be persistent on the client device. Repetitive checks for changes via the server can offer valuable insights into potential alterations, similar to server-based monitoring.

File Attribution. File attribution involves linking files to specific users, groups, or conceptual user entities based on actions such as uploading, creating, editing, deleting, sharing, or accessing. Attribution metadata varies across artifact types, and accurately tracing user actions and ownership requires careful collection and interpretation of this data.

However, attribution can be challenging. For instance, anonymous file drops obscure the uploader's identity. Similarly, federated sharing—where interconnected application instances enable cross-instance file access—can complicate attribution depending on the implementation. A comprehensive attribution analysis would require executing a broad range of user actions and systematically monitoring resulting artifact changes. As this necessitates an automated framework, it lies beyond the scope of this work and is left for future research.

Further Aspects. A further challenge arises in multi-server architectures where databases or cloud storage reside on external servers. This setup complicates data acquisition, often requiring API access. Although API usage may be feasible in certain cases, it typically demands authentication credentials configured within the server—details which may be accessible during forensic analysis. However, due to its rarity and dependency on backend-specific implementations, this approach is not pursued in the present work.

Cloud storage applications also differ in their internal file storage strategies. Some use file fragmentation for deduplication, necessitating chunk identification and reassembly prior to analysis. Others apply delta encoding, storing only modified file segments across versions.

In network traffic analysis, techniques such as chunked uploads and differential syncing—where only file segments are transferred—can complicate reconstruction.

Since Nextcloud does not implement fragmentation, delta encoding, chunked uploads, or differential syncing, these mechanisms fall outside the scope of this study. Likewise, features specific to individual applications or client software, such as multi-account support or virtual files, are not addressed.

4.2 An Improved Cloud Storage Forensic Framework

Building upon the underexplored aspects identified in the previous section, we propose a refined forensic framework tailored to the challenges of modern cloud storage environments.

A key distinction of the newly proposed framework is the inclusion of a monitoring phase. As discussed in the previous section, we believe that continuous monitoring can significantly enhance the forensic value of investigations, providing deeper insights into data access and usage patterns. To streamline the model, we exclude three phases from Teing et al.'s [25] framework: Readiness, Case Investigation Preparation, and Investigation Review and Lessons Learned. These are general to forensic practice and not specific to cloud storage forensics.

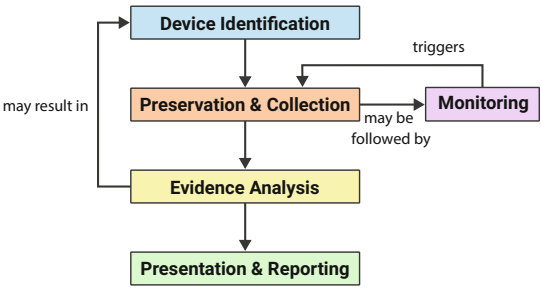

Fig. 1. An improved cloud storage forensic framework including a new Monitoring phase.

Other identified gaps do not require structural modifications but instead call for refinements within individual phases. Additionally, some gaps pertain to specific features or scenarios and are therefore not addressed in the overall framework overview. Consequently, our proposed framework consists of five phases, as illustrated in Fig. 1 and detailed below:

1. *Device Identification*: The first phase involves identifying devices such as mobile phones, laptops, computers, or servers. Device identification can occur by physically locating the device with a suspect or by identifying unique identifiers, such as a server's IP address or associated domain.
2. *Preservation & Collection*: In this second phase, the first objective is to preserve the data by creating a copy, ideally unmodified, from the data source. This is often achieved through classic acquisition methods like drive imaging and memory dumping, but can also be achieved through API acquisition. The collection aspect focuses on extracting relevant evidence, typically through the retrieval of artifact content. Depending on the data source and the tools used, preservation and collection may be performed simultaneously, as exemplified for API analysis. We therefore decided to place both Preservation & Collection (as already done by Teing et al. [25]) into a single phase.
3. *Monitoring*: The monitoring phase involves continuous observation of a device. Based on detected changes or after a set time interval, the preservation and collection of selected or all data sources may be repeated. This phase effectively becomes an iterative extension of the Preservation & Collection phase. While this ongoing process is generally advisable, it may not always be feasible in practice due to legal limitations or time constraints.
4. *Evidence Analysis*: Following the Preservation & Collection phase, evidence analysis involves scrutinizing the extracted artifacts to identify and extract relevant data for the investigation. Common categories of artifacts include user data and accounts, device information, files and file metadata, synchronization metadata, user activity, file versions, file deletion and trash, and file sharing. Additionally, further devices, such as cloud storage server instances or other devices with client instances, may be identified. Depending on the case, these additional devices may undergo similar analysis, starting with the Preservation & Collection phase, thus triggering an iteration within the framework.
5. *Presentation & Reporting*: This phase is identical to the phase of the same name described by Teing et al. [25], emphasizing the structured communication of findings through clear documentation and, when necessary, expert testimony. It ensures that evidence is conveyed accurately and comprehensibly to stakeholders, including legal professionals, while maintaining the integrity and admissibility of the data.

5 API Acquisition

Building on our earlier discussion of API access, this sections examines the server-side artifacts retrievable through API-based acquisition for Nextcloud.

5.1 API Analysis

In this section, we present our findings on relevant Nextcloud API endpoints, as shown in Table 3, which can be leveraged for forensic acquisition. We also evaluate the forensic soundness of these endpoints to assess their suitability for use in evidence collection.

We identified two primary APIs that can be used to query information from a Nextcloud instance: the Open Collaboration Services (OCS) API[5] and the WebDAV API.[6] As outlined in Sect. 3.2, both APIs support HTTP Basic Authentication, which can be performed using the account username and an application-specific password obtained from a previously acquired client device. Alternatively, if the user's primary password is available, a new app password can be generated.

User Data and Accounts. Nextcloud provides the OCS API endpoint `ocs/v2.php/cloud/user` to query information about the currently authenticated user, i.e. , the user that belongs to the provided app password. This endpoint returns metadata such as the user's display name, email address, storage quota, and other account-related details.

Additionally, the `ocs/v1.php/cloud/users` endpoint allows administrators to retrieve a list of all users on the server. To obtain more detailed information, such as last login time or group memberships, about each user, the `ocs/v2.php/cloud/users/username` endpoint can be used. While administrators can query information for all users using this endpoint, default users can only obtain information about themselves. Another endpoint is `ocs/v2.php/cloud/users/details`, which requires a `search` parameter to search for user details based on a search term.

Device Information. Nextcloud's public APIs do not provide direct access to device metadata. However, device and session details are available via the web interface under *Devices & Sessions* (`settings/user/security`). Each entry typically represents a device linked to an application-specific password and includes metadata such as session name (e.g., browser or app identifier), device type (e.g., mobile or desktop), and the timestamp of last activity.

Users with appropriate privileges can manage individual sessions through this interface: sessions can be revoked (invalidating the associated application password), file system access can be restricted, and a remote wipe can be initiated. The remote wipe command both revokes server access and instructs the client to delete locally stored Nextcloud data.

[5] https://docs.nextcloud.com/server/latest/developer_manual/client_apis/OCS/index.html.

[6] https://docs.nextcloud.com/server/latest/developer_manual/client_apis/WebDAV/index.html.

Files. Nextcloud exposes a WebDAV interface at `remote.php/dav/files/username` for enumerating user files and directories via the HTTP `PROPFIND` method. The request includes an XML body specifying desired properties, and the response returns metadata such as IDs, names, paths, sizes, MIME types, timestamps, and ownership. The `Depth` header (commonly set to `1`) controls directory traversal depth.

Metadata for specific files or directories can be queried by appending their full path to the endpoint. The response includes a `<d:href>` element indicating the resource location, which can be used in a subsequent HTTP `GET` request to retrieve the content. For property specifications and request structure, see the official documentation[7].

Synchronization. The WebDAV API endpoint `remote.php/dav/files/` provides metadata such as `etag` and `getlastmodified`, which can be used to infer file and directory changes and potential synchronization activity. However, it does not expose explicit synchronization events or client-specific timestamps. Such data is typically maintained on the client side, as demonstrated earlier for the Android application, limiting the server API's utility for reconstructing precise synchronization timelines.

User Activity and File Versions. For retrieving a file's activity history the OCS API provides the endpoint `ocs/v2.php/apps/activity/api/v2/activity/filter`. By specifying the file's `object_id` as a GET parameter, this endpoint returns a list of recorded file activities. Each entry includes an activity ID, a type (e.g., creation, modification, or deletion), and a corresponding timestamp. Notably, activity entries remain accessible through this endpoint even after the associated file has been permanently deleted from the trash bin.

For versioning, Nextcloud offers the `remote.php/dav/versions/username/versions/fileId` endpoint, which returns a list of all available versions of a given file. Each version entry contains a last modified timestamp, an `etag`, and a reference path, where the timestamp serves as a unique identifier for the version (e.g., `remote.php/dav/versions/username/versions/fileId/timestamp`).

This path can be used to retrieve metadata or download the file content corresponding to that specific version.

Deleted Files and Trash. A users's trash bin is accessible via a WebDAV `PROPFIND` request to the endpoint `remote.php/dav/trashbin/username/trash`. This returns deleted files and directories with metadata such as display name, size, and unique identifier. Additional trashbin-specific properties include: `trashbin-location` (name in the trash), `trashbin-original-location` (original path), and `trashbin-deletion-time` (Unix timestamp).

[7] https://docs.nextcloud.com/server/20/developer_manual/client_apis/WebDAV/basic.html.

Restoring a file involves a `MOVE` request to `remote.php/dav/trashbin/username/restore/`, which reintegrates it into the user's file hierarchy. As this operation alters the cloud environment, we instead locate and directly download deleted files from the trash to preserve the existing storage structure for forensic purposes.

File Sharing. The OCS API endpoint `/ocs/v2.php/apps/files_sharing/api/v1/shares` can be used to query the shares of a user supporting several optional parameters, such as `reshares`, `shared_with_me`, and `subfiles`, which allow filtering the results to include reshares, shares received by the user, or subfiles within shared directories, respectively. The response contains detailed metadata about each share, including the share type (e.g., user, group, or public link), permissions, expiration date, and the path to the shared resource.

5.2 Implementation

To support the acquisition and forensic analysis of Nextcloud environments, we developed and publicly released a set of specialized tools[8]. While general-purpose tools like rclone—a command-line program supporting various cloud storage backends—exist, they are not designed for forensic purposes and lack support for metadata extraction (e.g., activity logs, file versions, deletion traces) [3]. Our tool focus on preserving exactly these artifacts to ensure forensic soundness and transparency.

The implementation supports data retrieval and examination via Nextcloud's APIs and web interface. It typically requires the instance URL and user credentials (username and application-specific password).

Since Nextcloud does not expose a public API for device/session management, we reimplemented relevant web client functionality. Our approach directly interacts with specific web endpoints to retrieve and revoke connected devices while avoiding full web interface emulation, thereby minimizing the risk of unintended state changes.

Our implementation offers the following capabilities:

- **User Information:** Retrieve user-related data, such as detailed account info, user listings, and user search capabilities (`user-info`, `list-users`, `search-user`).
- **File System Analysis:** List files, resolve file IDs to paths, and download file contents
  (`list-files`, `file-id-to-path`, `download-file`).

[8] https://github.com/miqsoft/nextcloudforensic.

- **Deleted Content Recovery:** Access and extract information from the trash bin
  (`trash-bin`).
- **File History and Activity:** Retrieve file version history and activity logs (`file-versions`, `file-activity`).
- **Device Management:** List connected devices and revoke their access (`list-devices`, `revoke-device`, `revoke-all`). This allows investigators to prevent further interaction with the cloud storage from other devices during forensic acquisition.
- **Full Share Dump:** Recursively acquire the entire content of a user's file share (`dump`).

Detailed usage instructions and documentation are available in the project's repository.

Nextcloud: The Sleuth Kit (TSK)-Inspired Utilities. To provide a familiar interface for practitioners, we developed a set of command-line tools inspired by The Sleuth Kit, adapted to cloud-based storage:

`nc-fsstat`	Displays general information about the Nextcloud instance, including server capabilities and available users.
`nc-fls`	Provides a file system view similar to traditional forensic tools. Supports specifying a subdirectory, recursive traversal (`-r`), and showing deleted files (`-d`) marked with an asterisk. Deleted files are identified by correlating trash bin data with the current hierarchy. Each file or directory is listed with its object ID.
`nc-istat`	Retrieves metadata about a file or directory based on its object ID, including file activity and available versions.
`nc-icat`	Outputs the content of a file identified by its object ID.

6 Conclusion

Cloud storage systems challenge traditional forensic methods due to their distributed, dynamic nature and limited client-side visibility. This work shows how API-based acquisition offers a practical, forensically aware entry point for accessing server-resident data—especially in early investigation stages. When combined with continuous monitoring, API integration enables timely, targeted capture of both static and ephemeral evidence, enhancing visibility into user actions and system changes across client and server environments.

Centered on Nextcloud, our analysis demonstrates how native APIs, often overlooked in previous research, can provide structured, reliable, and repeatable access to critical artifacts. To support this, we developed and released an open-source acquisition tool tailored to Nextcloud's API, but extendable to further cloud storage applications.

Our analysis revealed that many artifacts—persisting even after more than a decade—remain consistent with those identified in earlier studies of ownCloud, the software from which Nextcloud was forked. Apart from the artifacts, we identified a major shortcoming in existing cloud storage forensic frameworks: their reliance on static, one-time evidence collection of a device. This approach misses opportunities to detect events after a first acquisition, such as new file uploads, user interactions, or newly connected client devices—all of which may be crucial during an investigation. In response, we proposed a revised forensic framework incorporating a monitoring phase and refining several stages to better reflect the demands of modern, self-hosted environments.

Our work lays the groundwork for further research in this area. While it analyzes existing artifacts, it does not comprehensively map artifact changes to specific user actions—such as creating, sharing, or deleting files—as this would require an automated system capable of simulating a wide range of user scenarios. Many modern self-hosted platforms also include groupware features like calendars, messaging, and video conferencing, which may contain valuable forensic artifacts. Exploring whether the API acquisition method and our framework can be extended to these services is a promising avenue for future work.

While we have conceptually outlined the adapted forensic framework for analyzing self-hosted cloud storage applications, validating it through real-world case studies remains a key task for future research. Finally, while decentralized systems like SyncThing differ significantly from cloud-based platforms, the iterative, monitoring-focused design of our framework may still be applicable—offering another opportunity for adaptation and evaluation.

A Server Side Database Tables

Table 2. Content of the server side database `oc_external_config`

config_id	mount_id	key	value
1	1	datadir	/media/crypt/Cloud/folder1
2	2	datadir	/home/hans/folder2
5	3	datadir	/var/foobar/folder3
19	6	host	thisIs.MyURL.de
20	6	root	
21	6	secure	1
22	6	user	ftpSecrect-user
23	6	password	75779a1b1fd4105186fda1250a70f|d51ce64f0bfbb167dfc...

B Nextcloud API Endpoints

Table 3. Key Nextcloud API Endpoints for Forensic Analysis by Information Type

Information Type	API Endpoint(s)	Method

User Data	`ocs/v2.php/cloud/user`
	`ocs/v1.php/cloud/users`
	`ocs/v2.php/cloud/users/{username}`
		GET

Server Capabilities	`ocs/v1.php/cloud/capabilities`	GET
File System Data	`remote.php/dav/files/{username}` `remote.php/dav/files/{username}/{path}`	PROPFIND
Deleted Files	`remote.php/dav/trashbin/{username}/trash`	PROPFIND
File Versions	`remote.php/dav/versions/{username}/versions/{fileId}` `remote.php/dav/versions/{username}/versions/{fileId}/{timestamp}`	PROPFIND, GET
Activity Data	`ocs/v2.php/apps/activity/api/v2/activity/filter`	GET
Sharing Data	`ocs/v2.php/apps/files_sharing/api/v1/shares`	GET
Security & Sessions	`ocs/v2.php/core/apppassword` `ocs/v2.php/core/apptokens`	DELETE, GET

References

1. Ahmad, N.H., Hamid, A.S.S.A., Shahidan, N.S.S., Ariffin, K.A.Z.: Cloud forensic analysis on pCloud: from volatile memory perspectives. In: Miraz, M.H., Excell, P.S., Ware, A., Soomro, S., Ali, M. (eds.) Emerging Technologies in Computing, pp. 3–15. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-60036-5_1
2. Bhat, W.A., Jalal, M.F., Khan, S.S., Shah, F.F., Wani, M.A.: Forensic analysis of sync.com and FlipDrive cloud applications on android platform. Forensic Sci. Int. **302**, 109845 (2019). https://doi.org/10.1016/j.forsciint.2019.06.003, https://linkinghub.elsevier.com/retrieve/pii/S037907381930252X
3. Breitinger, F., Zhang, X., Quick, D.: A forensic analysis of Rclone and Rclone's prospects for digital forensic investigations of cloud storage. Forensic Sci. Int. Digital Investigation **43**, 301443 (2022)
4. Dargahi, T., Dehghantanha, A., Conti, M.: Chapter 12 - investigating storage as a service cloud platform: pCloud as a case study. In: Choo, K.K.R., Dehghantanha, A. (eds.) Contemporary Digital Forensic Investigations of Cloud and Mobile Applications, pp. 185–204. Syngress, Waltham, MA (2017). https://doi.org/10.1016/B978-0-12-805303-4.00012-5, https://www.sciencedirect.com/science/article/pii/B9780128053034000125
5. Daryabar, F., Dehghantanha, A., Choo, K.K.R.: Cloud storage forensics: MEGA as a case study. Aust. J. Forensic Sci. **49**(3), 344–357 (2017). https://doi.org/10.1080/00450618.2016.1153714, https://www.tandfonline.com/doi/full/10.1080/00450618.2016.1153714
6. Daryabar, F., Dehghantanha, A., Eterovic-Soric, B., Choo, K.K.R.: Forensic investigation of OneDrive, box, GoogleDrive and dropbox applications on android and iOS devices. Aust. J. Forensic Sci. **48**(6), 615–642 (2016). https://doi.org/10.1080/00450618.2015.1110620

7. Easwaramoorthy, S., Thamburasa, S., Samy, G., Bhushan, S.B., Aravind, K.: Digital forensic evidence collection of cloud storage data for investigation. In: Proceedings of the 2016 International Conference on Recent Trends in Information Technology (ICRTIT), pp. 1–6. IEEE, Chennai, India (2016). https://doi.org/10.1109/ICRTIT.2016.7569516. Accessed 12 May 2025
8. Hale, J.S.: Amazon cloud drive forensic analysis. Digit. Investig. **10**(3), 259–265 (2013). https://doi.org/10.1016/j.diin.2013.04.006, https://www.sciencedirect.com/science/article/pii/S1742287613000352
9. Hargreaves, C., Breitinger, F., Dowthwaite, L., Webb, H., Scanlon, M.: DFPulse: the 2024 digital forensic practitioner survey. Forensic Sci. Int. Digital Inv. **51**, 301844 (2024)
10. Lim, S.Y., Johan, A., Daud, P., Ismail, N.A.: Dropbox forensics: forensic analysis of a cloud storage service. Int. J. Eng. Trends Technol. 68(CAT 2020-III), 45–49 (2020). https://doi.org/10.14445/22315381/CATI3P207, https://www.ijettjournal.org/Special%20issue/CAT-2020-III/CATI3P207.pdf. Accessed 23 June 2021
11. Martini, B., Choo, K.K.R.: An integrated conceptual digital forensic framework for cloud computing. Digit. Investig. **9**(2), 71–80 (2012). https://doi.org/10.1016/j.diin.2012.07.001, https://linkinghub.elsevier.com/retrieve/pii/S174228761200059X
12. Martini, B., Choo, K.K.R.: Cloud storage forensics: ownCloud as a case study. Digit. Investig. **10**(4), 287–299 (2013). https://doi.org/10.1016/j.diin.2013.08.005, https://www.sciencedirect.com/science/article/pii/S1742287613000911
13. Martini, B., Do, Q., Choo, K.K.R.: Mobile cloud forensics. In: The Cloud Security Ecosystem, pp. 309–345. Elsevier, Amsterdam, Netherlands (2015). https://doi.org/10.1016/b978-0-12-801595-7.00015-x, http://dx.doi.org/10.1016/B978-0-12-801595-7.00015-X, iSBN: 9780128015957
14. Mehreen, S., Aslam, B.: Windows 8 cloud storage analysis: dropbox forensics. In: 2015 12th International Bhurban Conference on Applied Sciences and Technology (IBCAST), pp. 312–317. IEEE, Islamabad, Pakistan (2015). https://doi.org/10.1109/IBCAST.2015.7058522
15. Mishra, H., Sihag, V., Choudhary, G., Dragoni, N., You, I.: Cloud storage client forensic: analysis of mega cloud. In: Singh, P.K., Wierzchoń, S.T., Chhabra, J.K., Tanwar, S. (eds.) Futuristic Trends in Networks and Computing Technologies, pp. 1099–1110. Springer Nature Singapore, Singapore (2022)
16. Mohtasebi, S., Dehghantanha, A., Choo, K.K.: Chapter 13 - cloud storage forensics: analysis of data remnants on spideroak, justcloud, and pcloud. In: Choo, K.K.R., Dehghantanha, A. (eds.) Contemporary Digital Forensic Investigations of Cloud and Mobile Applications, pp. 205–246. Syngress, Cambridge, MA, USA (2017). https://doi.org/10.1016/B978-0-12-805303-4.00013-7, https://www.sciencedirect.com/science/article/pii/B9780128053034000137
17. Oestreicher, K.: A forensically robust method for acquisition of iCloud data. Digit. Investig. **11**, S106–S113 (2014). https://doi.org/10.1016/j.diin.2014.05.006, https://www.sciencedirect.com/science/article/pii/S1742287614000498
18. Quick, D., Choo, K.K.R.: Digital droplets: microsoft SkyDrive forensic data remnants. Futur. Gener. Comput. Syst. **29**(6), 1378–1394 (2013). https://doi.org/10.1016/j.future.2013.02.001, https://www.sciencedirect.com/science/article/pii/S0167739X13000265
19. Quick, D., Choo, K.K.R.: Dropbox analysis: data remnants on user machines. Digit. Investig. **10**(1), 3–18 (2013). https://doi.org/10.1016/j.diin.2013.02.003, https://www.sciencedirect.com/science/article/pii/S174228761300011X

20. Quick, D., Choo, K.K.R.: Google drive: forensic analysis of data remnants. J. Netw. Comput. Appl. **40**, 179–193 (2014). https://doi.org/10.1016/j.jnca.2013.09.016, https://www.sciencedirect.com/science/article/pii/S1084804513002051
21. Quick, D., Martini, B., Choo, K.K.R.: Chapter 2 - cloud storage forensic framework. In: Quick, D., Martini, B., Choo, K.K.R. (eds.) Cloud Storage Forensics, pp. 13–21. Syngress, Boston (2014). https://doi.org/10.1016/B978-0-12-419970-5.00002-8, https://www.sciencedirect.com/science/article/pii/B9780124199705000028
22. Rochmadi, T., Wicaksono, Y., Nisa, N.D.: Digital evidence identification of android device using live forensics acquisition on cloud storage (iDrive). Int. J. Comput. Appl. **175**(26), 40–43 (2020). https://doi.org/10.5120/ijca2020920815, http://www.ijcaonline.org/archives/volume175/number26/rochmadi-2020-ijca-920815.pdf
23. Roussev, V., Barreto, A., Ahmed, I.: API-Based forensic acquisition of cloud drives. In: Peterson, G., Shenoi, S. (eds.) Advances in Digital Forensics XII, pp. 213–235. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-46279-0_11
24. Teing, Y.Y., Dehghantanha, A., Choo, K.K.R.: CloudMe forensics: a case of big data forensic investigation. Concurr. Comput. Practice Exp. **30**(5), e4277 (2018). https://doi.org/10.1002/cpe.4277, https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4277
25. Teing, Y.Y., et al.: Private cloud storage forensics: seafile as a case study. In: Dehghantanha, A., Choo, K.K.R. (eds.) Handbook of Big Data and IoT Security, pp. 73–127. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-10543-3_5
26. Thamburasa, S., Easwaramoorthy, S., Aravind, K., Bhushan, S.B., Moorthy, U.: Digital forensic analysis of cloud storage data in IDrive and mega cloud drive. In: 2016 International Conference on Inventive Computation Technologies (ICICT). vol. 3, pp. 1–6. IEEE, Coimbatore, India (2016). https://doi.org/10.1109/INVENTIVE.2016.7830159