# D-Cloud-Collector: Admissible Forensic Evidence from Mobile Cloud Storage*

Mark Vella[1][0000−0002−6483−9054] and Christian Colombo[1][0000−0002−2844−5728]

Department of Computer Science, University of Malta, Malta
{mark.vella,christian.colombo}@um.edu.mt

**Abstract.** Difficulties with accessing device content or even the device itself can seriously hamper smartphone forensics. Mobile cloud storage, which extends on-device capacity, provides an avenue for a forensic collection process that does not require physical access to the device. Rather, it is possible to remotely retrieve credentials from a device of interest through undercover operations, followed by live cloud forensics. While technologically appealing, this approach raises concerns with evidence preservation, ranging from the use of malware-like operations, to linking the collected evidence with the physically absent smartphone, and possible mass surveillance accusations. In this paper, we propose a solution to ease these concerns by employing hardware security modules to provide for controlled live cloud forensics and tamper-evident access logs. A Google Drive-based proof of concept, using the SEcube hardware security module, demonstrates that *D-Cloud-Collector* is feasible whenever the performance penalty incurred is affordable.

**Keywords:** Cloud storage forensics · Digital evidence preservation · Right to privacy · Hardware security modules · Tamper-evident logs.

## 1 Introduction

Nowadays, smartphones store sufficient data about their owners to the extent that they can provide a single source of digital forensic evidence to solve incidents involving criminal behaviour [19]. On the flip side, the same sophisticated technology that comes in handy for investigators can also be used to block access to digital evidence [5]. Locked/encrypted or even missing/damaged devices are a case in point. Firmware manipulation combined with device rooting, or else hardware-level acquisition, provide some options to investigators. Yet these tend to be very intrusive or else cannot fully solve the encrypted content problem respectively [15]. In the case of the San Bernardino terror attack[1], a spectacular stand-off between law enforcement and technology vendors ensued. While in this case a third party forensics tool vendor came to the rescue, such solutions

* This work is supported by the LOCARD Project under Grant H2020-SU-SEC-2018-832735.

[1] https://www.insidescience.org/news/looming-end-smartphone-company-law-enforcement-standoff

remain largely specific to device models and operating system versions [17, 13], and taken out by security updates.

In recent years the mobile security landscape has witnessed a significant increase in malware exploiting social engineering tactics [25], as well as advanced software exploitation able to pull off successful credentials attacks of sorts [17]. Once deployed in a sufficiently controlled manner, the same malware techniques could offer a remote solution that does not even require physical access to smartphones. The key enablers of this approach comprise mobile cloud storage services and the application programmer interfaces (APIs) exposed by them (e.g. Google Drive for Developers[2] and the CloudKit framework for Apple iCloud[3]). To compensate for physical storage constraints, smartphone vendors offer cloud storage services (for file storage and app data backups) that integrate seamlessly with the mobile operating system (OS) along with a storage quota for free.

Once retrieved stealthily during an undercover operation, credentials can be used by a cloud forensics tool that consumes cloud storage APIs. Akin to the classic telephone tapping context, investigators can present a probable cause to believe that a remote undercover operation could help in solving a serious crime, such as drug trafficking, money laundering, or terrorism [21]. In this case, the arrangement executes a remote credentials theft attack on the target device. Similar to the classic phone context, however, law enforcement agencies are held to a higher standard of operational integrity due to the intrusive nature of such operations. Recent accusations of mass surveillance by governments using the Pegasus spyware have caused an uproar[4], invoking a breach of the right to privacy as described by the Universal Declaration of Human Rights Article 12 [20], and associated laws, e.g. GDPR in the EU.

### 1.1 Research Problem

A universally accepted mechanism to preserve evidence, thereby helping in having it admissible for court proceedings, is the forensic chain of custody (CoC) [4]. A comprehensive CoC involving digital investigation is required, such that for any given evidence, the following is included with proper authentication using digitally signed hashes [6]: i. the custodian (e.g. first responders, case investigators), ii. details of the evidence itself with proper identification (e.g. phone IMEI or storage image checksum), iii. relevant case details, iv. the temporal information associated with the evidence and custody, as well as v. the spatial data related to the evidence location. Overall, the CoC should be suitable to track the entire lifecycle of evidence as proof of its integrity.

In our case (see Figure 1), the CoC is also burdened with demonstrating proper usage of the remotely retrieved credentials. In particular, investigators need support in establishing that credentials usage falls within the parameters

[2] https://developers.google.com/drive
[3] https://developer.apple.com/documentation/cloudkit/
[4] https://theconversation.com/spyware-why-the-booming-surveillance-tech-industry-is-vulnerable-to-corruption-and-abuse-164917

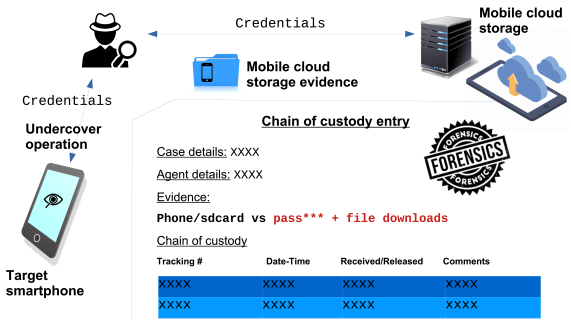


Fig. 1: *D-Cloud-Collector*: reconciling mobile cloud storage evidence obtained via remote undercover operations with forensic chains of custody.

of not impinging on the target's right to privacy. One further issue concerns compatibility with existing CoC. In the case of mobile forensics, typical CoC entries correspond to a physical device or part thereof, e.g. sd card. In this manner, the CoC entry for the device links it to the digital image, while any extracted evidence relates to the authenticated digital image. However, the stolen credentials approach does not fit this generally accepted procedure. Instead, in this case, the credentials do not immediately associate with the device in question. Digital evidence is collected through live forensics rather than forensic imaging of device storage. Device confiscation is not even required.

### 1.2 Contributions

In this paper, we propose *D-Cloud-Collector* (DCC), a live forensics solution for mobile cloud storage that reconciles with CoC requirements: 1a. Credentials are obtained through an undercover operation only after approval by the relevant authorities. 1b. All access to cloud evidence is fully authenticated and logged through the use of a removable Hardware Security Module (HSM), e.g. a USB token, which also provides 1c. A one-to-one mapping between the acquired evidence and the absent mobile device. All this whilst ensuring 2. Security and 3. Practicality in terms of performance, although at the cost of additional evidence collection time.

The scope of work presented in this paper is to validate the HSM's central role in DCC. This hardware component provides secure storage and usage of the authorization tokens and symmetric keys needed for the secure access of cloud APIs. Additionally, a secure hash-based primitive implemented inside the HSM enables tamper-evident cloud API access to logs. Ancillary features already widely used in security solutions complete the full DCC picture. Specifically, the HSM's tamper-evident features, CPU protection rings, and anti-code injection module loaded by the forensic collection tool on the investigator's workstation protect the security of the HSM itself. We defer their in-depth treatment to subsequent work.

The key contributions of this work, therefore, aim to answer the following research question: *"How can HSMs be utilised as a basis to collect evidence from mobile cloud storage, through stealthily-retrieved credentials, in a manner that is consistent with evidence preservation and the right to protect personal data?"* To this end we provide: 1. A characterization of the target use case and a conceptual description of DCC (Section 4). 2. A proof-of-concept implementation of the HSM component of DCC based on the SEcube chip [26] (Section 5). 3. A Google Drive case study (Section 6).

## 2 Background

*Credentials theft vectors on smartphones.* Spyware targeting Android [25] and iOS [11] smartphones alike propagate in the form of trojan/infected versions of legitimate apps. Statistics show that detection mechanisms employed by app stores let through a significant number of spyware samples [9]. While tech-savvy users may get suspicious by the sheer amount of permissions required by these rogue apps during the installation process, a good number of users do fall prey to their deceptive tactics [27]. Even more so, certain threat vectors allow trojan apps not to look overly suspicious [22], bypassing the need for sensitive permissions or requiring any device rooting/jailbreaking. The result is highly stealthy malware that goes unnoticed by victims for long periods.

*OAuth2.* DCC relies on cloud service providers supporting an authorization mechanism that allows account owners to delegate privileges to third party apps. While not tied to a specific mechanism, OAuth2 is a widely adopted standard framework [7], and our DCC implementation assumes it. It presupposes HTTPS and the consequently derived security services from the underpinning TLS1.3. The critical step in OAuth2 is app-flow redirection through a web browser requesting user content for the third-party app, which is granted access to the user's cloud service account. Following successful third-party application and user permission granting, the application receives an access token to be presented by all subsequent access requests. A refresh token is also obtained and is used whenever a new access token is needed following its expiration.

*Hardware security modules.* Hardware security modules aim to make up for the limitations faced by software-only protection mechanisms. While secure elements, trusted platform modules (TPM) and Trusted Execution Environments (TEE) are bound to specific hardware, HSMs offer a more flexible solution [8]. These devices typically take the form of high-performance plugin cards[5] or removable external devices[6]; their primary application being secure cryptography implementation (standard PKCS#11 [24] is dedicated for this purpose).

[5] https://cpl.thalesgroup.com/encryption/hardware-security-modules/general-purpose-hsms

[6] https://www.secube.blu5group.com/products/usecube-bundle-including-5-usecube-tokens-and-1-devkit/

## 3 Related Work

Along with the analysis of messaging apps, the importance of mobile forensics in criminal investigations involving the use of cloud storage services has already been acknowledged [3]. While cloud storage forensics has been explored from multiple aspects, to the best of our knowledge, our work is the first to focus specifically on mobile cloud storage forensics, where the collection process leverages stealthily obtaining credentials. This collection method effectively replaces the conventional confiscate-unlock-image-collect-analyze procedure [15]. DCC's role is restricted to the collection phase, replacing phone storage imaging with cloud storage collection using the obtained credentials. The need for device confiscation and unlocking is replaced by remote installation of undercover investigation software. Evidence examination and the associated CoC can proceed similarly to any phone-present approach, possibly even through a distributed CoC based on a distributed ledger [14, 12].

DCC could also extend the role of the undercover operation software beyond credentials retrieval and make it collect evidence from cloud storage. This approach would avoid the need for an HSM to protect the retrieved credentials from dishonest investigators. Instead, the cloud storage evidence could be registered directly into a CoC by the undercover tool, with the integrity of this operation safeguarded by blockchain-backed evidence storage [10]. However, this approach entails a significant extension of the covert operation. When considering the computationally intensive evidence collection operation, this approach could severely increase the chance of giving away the entire operation.

The Cloud-Forensics-as-a-Service model (CFaaS) [18] is a cloud forensic process model that requires a prior agreement between the cloud service providers and consumers. The model expects both parties to synchronise on both sides' forensic data collection process, with the correlation of evidence collected on each side carried out during a subsequent analysis stage. The bilateral agreement is finalised during a cloud forensic readiness stage. A similar approach [1] that focuses on cyberattacks targeting mobile cloud apps goes as far as requiring the synchronisation of client and server-side forensic readiness. With cloud service providers' provision of log services for forensic investigation purposes, concerns related to the possibility of malicious behaviour also abound. In this regard, logging services must be hardened against dishonest cloud users, providers, and investigators [28]. Threats to secure logging comprise scenarios where the three entities are individually malicious or collude. Public key cryptography ensures confidentiality, while chained hashes of log entries along with proofs of past logs ensure integrity. On its part, DCC employs a similar log hashing scheme in the HSM to ensure tamper-evident logs. On the other hand, OAuth2 tokens for evidence retrieval ensure that cloud storage from other accounts remains inaccessible. Furthermore, read-only tokens should be prioritized whenever provided, thus protecting evidence integrity.

In contrast, the approach taken in DCC reflects the most likely scenario occurring during a criminal investigation, where no prior agreements exist, nor is any collaboration sought from the cloud provider. This scenario is not unique to

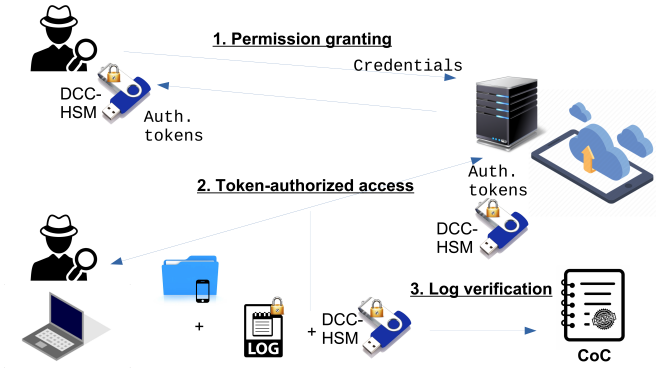


Fig. 2: A HSM-centric approach to DCC.

DCC, and in fact, some consumer-driven cloud forensic solutions that address different settings than the one studied in this work have already been explored [16]. The motivating factors are several. Besides the organisational challenge otherwise entailed, legal obstacles also loom [22]. Existing mutual legal assistance treaties are considered inadequate for the cross-border sharing of cloud evidence. Legal frameworks proposed to address this scenario have been widely criticised for violations of international law and for not being sufficiently sensitive to human rights. DCC focuses on the technical aspects of cloud forensics, providing means through which investigators can demonstrate the case-relevant usage of the stealthily obtained credentials.

## 4 Use Case and Conceptual Design

The assumed context within which DCC-based tools are to operate is as follows:

1. A bilateral agreement between the cloud service provider and the investigators is unavailable or perhaps not even possible.
2. An undercover investigation software has been implanted onto the target's smartphone using social engineering or software exploitation for delivery. Subsequently, it obtains the cloud credentials (user & password) while suppressing any alerts related to their usage by investigators.
3. A proxy server placed between the investigator's workstation and the cloud service provider, which doubles as a log management server, might offer a solution. Yet this arrangement is not deemed fully compatible with CoC practices unless sufficient resources are available to dedicate servers per case, only releasing them on case closure.

This last assumption merits further elaboration. In this hypothetical setup, during an initialisation stage, the cloud credentials are supplied directly to the proxy server and subsequently exchanged for authorization tokens. From this

point onward, an investigator workstation collects the related cloud evidence, performing all API requests through this proxy server, with the server adding the authorization tokens and performing de/encryption. Responses take the reverse route with the authorization tokens stripped from responses after decryption and ultimately delivering collected evidence to the investigator. The credentials are only used directly for a very brief period, after which they can be disposed of for further integrity. If the authorization tokens never leave the proxy server in plaintext form, there is no other way to make use of them. Similarly, access request logs are managed solely by the proxy server and are considered tamper-proof. The digitally signed access logs are to be registered with the CoC, thereby considered authentic and comprehensive of all cloud storage accesses.

However, the proxy server approach is difficult to reconcile with CoC requirements concerning linking the access logs and associated mobile cloud evidence with the absent smartphone. Suppose it was possible to dedicate a physical proxy server per case. In that scenario, it could be considered a replacement of the physically missing smartphone, with the collected cloud evidence and access logs replacing the phone's storage forensic image. Yet this comes across as an expensive proposition, especially for long-running cases. The idea of a proxy device that replaces the smartphone under custody is a concept we would like to stick to, but at the same time, it also has to be cost-effective. DCC proposes to use a dedicated HSM, referred to here as the *DCC-HSM*, as a much cheaper option than a dedicated proxy server. The tamper-evident features expected of any HSM combined with local-only access present the restricted attack surface.

As shown in Figure 2, the first two DCC stages, *permission granting* and *token-authorized access* mirror the proxy server-based approach. The DCC-HSM is attached locally to the investigator workstation, at the expense of offering reduced computational power compared to a fully-fledged server. Access logs also become prone to tampering prior to digitally signing them. An additional secure hash-based operation is therefore required and is employed during an additional *log verification* stage. The collected evidence, along with tamper-evident access logs, offer a replacement for forensic smartphone storage images. Thereby, the DCC-HSM provides a physical substitute for the missing phone.

## 5 Proposed Approach — D-Cloud-Collector (DCC)

*D-Cloud-Collector* (DCC) is a live forensics solution for mobile cloud storage obtained through the undercover retrieval of credentials from smartphones.

### 5.1 DCC Architecture

Figure 3 presents how DCC's main components are used during the first two stages of operation. The start of the 'permission granting' stage assumes that a session key has already been negotiated over TLS and is solely stored inside the HSM, referred to as *DCC-HSM*. Once the 'permission granting' procedures begin, the *DCC administration tool* forwards a cloud API authentication request

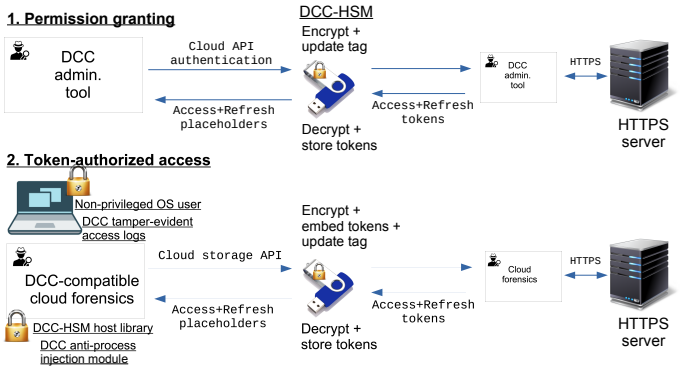


Fig. 3: The *D-Cloud-Collector* (DCC) components and their involvement in the first two stages of operation. The central role of the hardware security module (DCC-HSM) is noteworthy.

to the DCC-HSM. The request is first encrypted with the stored symmetric key and returned to the administration tool. Subsequently, it is forwarded to the cloud web server as HTTPS traffic, causing the previously discussed OAuth2 flow redirection and is the only point in time at which the investigator provides the stolen user/password. The corresponding encrypted response is sent back to the DCC administration tool (through an embedded web server) containing the OAuth2 refresh/access tokens. It is the DCC-HSMs responsibility to decrypt the response, extract the tokens and store them on the HSM, while at the same time replacing them with placeholder tokens inside the plaintext response. The availability of the OAuth2 tokens — which never leave the DCC-HSM in plaintext — is the key prerequisite for the second stage.

The token-authorized access stage represents all file download requests made to the cloud API server using the previously obtained OAuth2 tokens, renegotiating further TLS session keys as required. All API requests pass through the DCC-HSM for replacing the placeholder tokens with the actual ones, with subsequent encryption. Most responses will contain the downloaded evidence and which ones are to be decrypted again by DCC-HSM. During this second stage, any *cloud forensics tool* which is *compatible with DCC* instantiates the administration tool, i.e. a tool that can implement the flows shown in Figure 3, with DCC-HSM integration through a *host library* being a key requirement. Throughout both stages, all encryption requests also result in the computation of an authentication tag that depends on all previous encrypted requests. In this manner, the authenticity of the cloud access log, external to the HSM, can be verified.

DCC also carries several security requirements to protect the OAuth2 tokens inside the DCC-HSM, i.e. beyond the fact that the HSM needs to be tamper-evident. The key requirement comprises TLS session keys to be immediately scrubbed from memory by the DCC administration and DCC-compatible cloud forensics tool. By doing so, a malicious DCC end-user never gets the opportu-

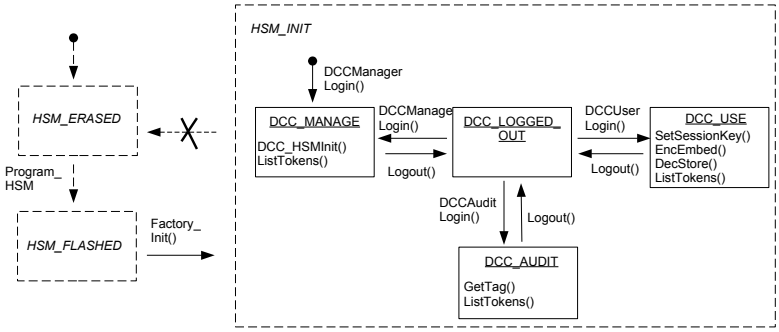


Fig. 4: DCC-HSM's lifecycle.

nity to decrypt ciphertext containing the access tokens. Besides requiring tool conformity, this requirement also entails tool hardening to protect from tool tampering, thereby fending off any future accusations that some form of foul-play was ever involved when handling cloud storage credentials. The first ancillary component comprises operating system (OS)-level access controls that prohibit *non-privileged OS users* from debugging program memory, or else to carry out process injection to leak session keys. Since process-injection can also happen through software bug exploitation, an additional *anti-process injection module* is also needed. Given that all the ancillary components secure the DCC-HSM, we choose to focus on describing this central component first for the scope of this paper. An in-depth treatment of the ancillary components makes sense only once the design of the DCC-HSM is thoroughly explored. We, therefore, start by delving deeper into the DCC-HSM's functionality.

### 5.2 The DCC-HSM Component

Figure 4 illustrates the DCC-HSM's lifecycle as a state machine. The top-level states correspond to the state transitions that take an HSM without any firmware to an operational state that is flashed with a binary image. Initially, the DCC-HSM is operated upon by the DCC administration tool. Specifically, starting with an empty device in the `HSM_ERASED` state, the HSM is flashed with software and transitions to `HSM_FLASHED`. At this point, the DCC-HSM device exposes its services through the host library. `Factory_Init()` is always the first service to be called where it initialises device parameters not strictly related to DCC functionality, e.g. device PINs and serial numbers. Once called, the DCC-HSM transitions into `HSM_INIT`. This top-level state is programmatically irreversible, implying DCC-HSM's strict association with a specific case, effectively replacing the physically missing device.

Once in `HSM_INIT` the DCC-HSM is ready to take part in the three DCC stages, providing access to its services depending on the current `HSM_INIT` substate. Initially, only the `DCC_MANAGE` state is accessible via `DCC_ManagerLogin()`.

Still within the context of the DCC administration tool, once supplied with the correct device administrator PIN, this operation takes the device to the `HSM_MANAGE` state, thereby setting DCC-specific parameters. While logged in this device in manager mode, it is possible to set the OAuth2 token placeholders and reserve space for the actual token strings on the device's persistent memory. Furthermore, the initial authentication tag value is set. `HSM_HSMInit()` is the operation responsible for all these tasks. On a newly flashed device, this operation is called automatically with default values, i.e. besides the possibility of being called explicitly with specific values. This means that from the point of calling `Logout()` onward, the device is ready for DCC participation (see Figure 3). The `DCC_USE` state requires a normal user login (`DCCUserLogin()`). In this state, TLS-negotiated session keys are provided through `SetSessionKey()`. API request/response en/decryption, and the associated access token storage and placeholder replacement services are provided by `EncEmbed()` and `DecStore()`. Both the 'permission granting' and the 'token-authorized access' DCC stages operate in the `DCC_USE` state. On completion of the first stage, the DCC-HSM stores the OAuth2 refresh and access token strings. These tokens are used by both the `EncEmbed()` and `DecStore()` operations to replace, or be replaced by, the placeholder tokens respectively. Assuming that the stealthily obtained passwords are safely disposed of, in this configuration, the DCC-HSM becomes the only medium through which the mobile cloud storage can be accessed.

Loaded with a refresh/access token pair, whose presence (not values) is verifiable through `ListTokens()`, the device is now ready to be operated upon by the DCC-compatible cloud forensics tool. The next time that a cloud evidence search-and-download procedure is needed, a call to `DCCUserLogin()` takes the DCC-HSM back to the `DCC_USE` state, and following calls to `SetSessionKey()` as necessary, a sequence of `EncEmbed()` and `DecStore()` operations are invoked to securely perform authorized requests followed by file downloads. The last two operations are also responsible for updating the internally-stored authentication tag. Each tag is computed out of the sequence of web API requests, comprising the entire plaintext HTTP request headers and any payload data. Whenever requests contain OAuth2 tokens, the tag is computed using the placeholder tokens rather than the actual token strings. Otherwise, it wouldn't be possible to verify the externally-stored logs, which should not have access to the token strings but only the placeholders. For this reason, the external log verification procedure must adhere to the following constraints: i. It synchronises the initial authentication tag with DCC-HSM, and ii. It processes an access log corresponding to the concatenation of plaintext requests as sent to the DCC-HSM for encryption. Tag computation is based on the hash extend algorithm, universally employed by TPMs [2], where for the next access log entry, a new tag $t'$ is computed out of the previous tag $t$ as: $t' \leftarrow hash(t \,||\, \text{log entry})$.

Whenever access log verification is required (DCC stage 3), `DCC_AuditLogin()` has to be called by the DCC administration tool to transition the DCC-HSM to the `DCC_AUDIT` state. This transition represents an auditor login. In this state, it becomes possible to call the `GetTag()` service, which retrieves the last computed

tag. Likewise, the access log generated by the cloud forensics tool is passed on to the DCC administration tool. It computes an external tag starting with the same initial tag. Only a pair of matching DCC-HSM/external tags constitutes proof of authenticity. On the other hand, a non-match event indicates a tampered-with log. Furthermore, since a successful 'permission granting' DCC stage leaves the DCC-HSM the only way to access the corresponding mobile cloud storage, the access log is also deemed comprehensive.

## 6 DCC-HSM Proof of Concept Evaluation

### 6.1 SEcube HSM

We prototyped DCC-HSM on the SEcube chip [26]. We chose this chip due to the tamper-evident features of its embedded STM32F4 microcontroller, a very low power ARM Cortex M4 RISC 180MHz CPU, with on-chip 256 KB SRAM and 2 MB flash. Importantly, any firmware developed on its corresponding developer board is immediately transferable to the USEcube™ USB Token[7] priced approximately just over €100. Furthermore, this multi-chip module also stacks a Lattice MachXO2-7000 device and an EAL5+ certified secure element. The two components carry potential for future enhancements for hardware acceleration and authenticated firmware updates and private-key binding, respectively. The DCC-HSM firmware is developed on top of the OpenSDK software libraries[8], comprising a device-side firmware and host-device libraries that communicate over the USB mass storage device class. WolfCrypt's ChaCha20/Poly1305 authenticated stream cipher was integrated with the firmware using STM32CubeMX to provide TLS1.3-compatible authenticated encryption. This cipher offers a popular option for hardware lacking AES acceleration. The complete DCC-HSM services (Figure 4) implemented in the SEcube's firmware are exposed as additions to the OpenSDK's host libraries. Logic related to the DCC administration and cloud forensics tools uses the Python bindings for the Drive API (V3). All sources experiment (discussed next) files are available for download[9].

### 6.2 Google Drive Case Study

The Google Drive app is available for both Android and iOS devices, providing 15GB of storage[10]. Aspects of the three DCC stages (Figure 2) follow.

*Stage 1: Permission granting.* The following snippet shows the (redacted) JSON store associated with the DCC administration and cloud forensics tools.

[7] https://www.secube.blu5group.com/products/usecube-bundle-including-5-usecube-tokens-and-1-devkit/
[8] https://github.com/SEcube-Project/SEcube-SDK
[9] https://github.com/mmarrkv/DCC
[10] https://play.google.com/store/apps/details?id=com.google.android.apps.docs

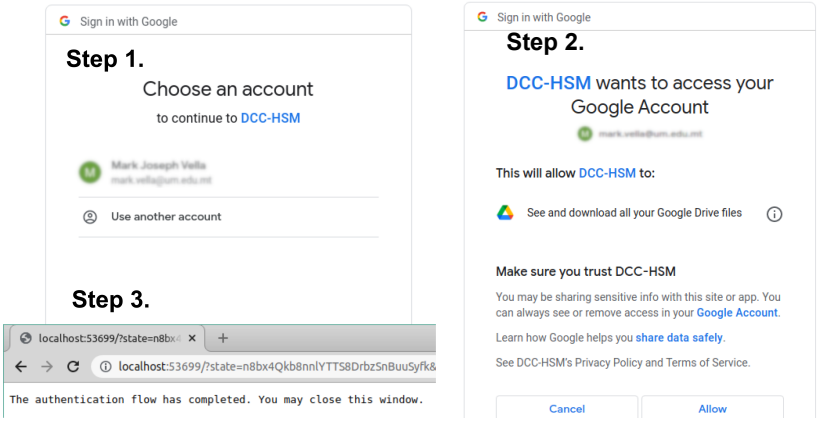


Fig. 5: DCC's permission granting stage with a Google OAuth 2.0 endpoint.

```
{"installed":
{"client_id":"<snip...>",
 "project_id":"dcc-hsm",
 "auth_uri":"https://accounts.google.com/o/oauth2/auth",
 "token_uri":"https://oauth2.googleapis.com/token",
 "auth_provider_x509_cert_url":"https://www.googleapis.com/oauth2/v1/
      certs",
 "client_secret":"<snip...>",
 "redirect_uris":["urn:ietf:wg:oauth:2.0:oob", "http://localhost"]}}
```

In particular, the application is registered to use the native application OAuth2 flow (`installed`), application name (`project_id`) and credentials (`client_id`, `client_secret`), the authorization service to which they are to be sent (`auth_uri`), and `http://localhost`, specifying that the access tokens will be sent to a local web server. Figure 5 includes screen captures from the permission granting process, involving flow redirection to a web browser (step 1) where the stolen password will have to be used, granting access to DCC-HSM (step 2), and receipt of an authorization code by the locally spawned web server (step 3).

The authorization code is intended for the OAuth2 token service (`token_uri`), with the following request/response snippets showing the receipt of both refresh and access tokens. Following response decryption, actual token strings remain on the HSM with the administration/forensics applications only getting access to the placeholder strings `PLC_ACC*` and `PLC_RFSH*`[11]. The `scope` of the tokens was chosen to provide read-only access to the Google Drive account concerned, further strengthening evidence preservation.

```
POST https://oauth2.googleapis.com/token HTTP/1.1
Host: oauth2.googleapis.com <snip...>
HTTP/1.1 200 OK<snip...>
{ "access_token": "PLC_ACC<snip...>",
  "expires_in": 3599,
```

[11] Stale Google tokens are actually used in the implementation.

```
  "refresh_token": "PLC_RFRSH<snip...>",
  "scope": "https://www.googleapis.com/auth/drive.readonly",
  "token_type": "Bearer"
```

*Stage 2: Token-authorized access.* The second stage involves the bulk of the evidence collection process, yet it is the most straightforward stage from a DCC point-of-view. Web API endpoints corresponding to stored files (`drive/v3/files/*`) are accessed with the placeholder tokens placed inside the `authorization` HTTP header. Once forwarded to the DCC-HSM, actual token strings replace placeholders followed by encryption as per the following request snippet, with the ensuing encrypted responses undergoing the reverse process.

```
GET https://www.googleapis.com/drive/v3/files/1
    Xay4B_uRSdxpmDJypmYwJi8gro1bR7B1?alt=media HTTP/1.1
Host: www.googleapis.com
x-goog-api-client: gdcl/2.28.0 gl-python/3.8.5
range: bytes=0-104857600
authorization: Bearer PLC_ACC<...snip...>
```

*Stage 3: Log verification.* The last stage of DCC needs to compare the currently stored authentication tag with an external recomputation of the tag over the access log. The key requirement is that the external computation synchronizes its initial tag with the HSM's. The log itself comprises the exact same sequence of plaintext HTTP requests sent to the HSM, i.e. containing the placeholder tokens instead of actual strings, as follows:

```
POST https://oauth2.googleapis.com/token HTTP/1.1
Host: oauth2.googleapis.com<snip...>
GET https://www.googleapis.com/drive/v3/files?q=mimeType%3D%27
    application%2Fvnd.google-apps.folder%27+and+name+%3D+%27
    file_collection%27&pageSize=10&fields=nextPageToken%2C+files%28id%2C
    +name%29&alt=json HTTP/1.1
Host: www.googleapis.com<snip...>
authorization: Bearer PLC_ACC<snip...>
GET https://www.googleapis.com/drive/v3/files/1
    Xay4B_uRSdxpmDJypmYwJi8gro1bR7B1?alt=media<snip...>
authorization: Bearer PLC_ACC<snip...>
```

If the access log is in order, log verification returns a matching authentication tag. However, by property of cryptographic hashes, even a single character modification would result in a completely different tag, exposing tampering. Therefore, it is also essential that the external verification procedure uses the same hash while ensuring no discrepancies between the implementations. For our case study, both the SEcube firmware and the log verification routine make use of WolfCrypt's SHA3-256, resulting in a matching tag for intact logs as per the following:

```
>>: Device audit in progress
>>: Tag value: 8d 88 42 1f e4 9e <snip...>
>>>>>Compute Tag value outside DCC-HSM

Tag value: 8d 88 42 1f e4 9e <snip...>
>>>>>Comparison + verdict: MATCH
```

### 6.3 Security Analysis

Besides the tamper-evident registers supported by the chosen HSM and the additionally required DCC modules needed to protect session keys from malicious users (see Section 5.1), the firmware itself may still introduce security holes. A case in point concerns our first attempt at optimizing the implementation of `DecStore()`. This version assumed that token replacement is only needed for JSON-formatted payloads returned by calls to `https://oauth2.googleapis.com/token`. Yet this approach opened up the possibility for an attack that re-injects an HSM-encrypted request, now containing the actual token string, back to the HSM and have it decrypted, as shown by the following listing:

```
>>: Decryption in progress
GET https://www.googleapis.com/drive/v3/files?q=mimeType%3D%27
    application%2Fvnd.google-apps.folder%27+and+name+%3D+%27
    file_collection%27&pageSize=10&fields=nextPageToken%2C+files%28id%2C
    +name%29&alt=json HTTP/1.1<snip...>
authorization: Bearer ya29.a0ARrd<snip...>
```

Removal of this optimisation fixed the issue even though this cat-and-mouse scenario, to which every software is prone, is far from closed.

### 6.4 Performance Analysis

Our proposal trades a fully-fledged proxy server for a low-cost removable HSM, which incurs a performance penalty. Yet, here we are speaking about a collection process carrying no real-time requirements, i.e. besides completing evidence collection within a reasonable time frame. We measured the processing time required by the HSM to process all necessary requests involving evidence ranging from 1.5-15GB, thus covering Google Drive's free storage capacity in 10 steps. For experiment repeatability, we obtained a packet capture of the traffic through a web proxy and conducted all measures in an offline manner. This web proxy setup entailed patching Python's `httplib2` and `requests` modules to deactivate X.509 certificate verification. Figure 6 shows a plot of the measurements, including both un/patched versions of the firmware, as well as a baseline comprising the `hpenc`[12] fast encryption tool and which maximises the utilisation of the 8-core Intel© Core™ i7-10700 CPU @ 2.90GHz used for experimentation.

Measurements show a quasi-linear increase in processing times for both firmware versions, ranging 0.9/1.2 - 7.4/8.1 hours for un/patched versions. Repeated runs registered very little dispersion, with $\sigma = 19.1/28.9$s, for un/patched versions with 1.5GB evidence. The baseline's impressive handling of the increase in computation load is a stark reminder of the cost of security, providing controlled cloud storage access and log integrity with CoC compatibility.

[12] https://github.com/vstakhov/hpenc

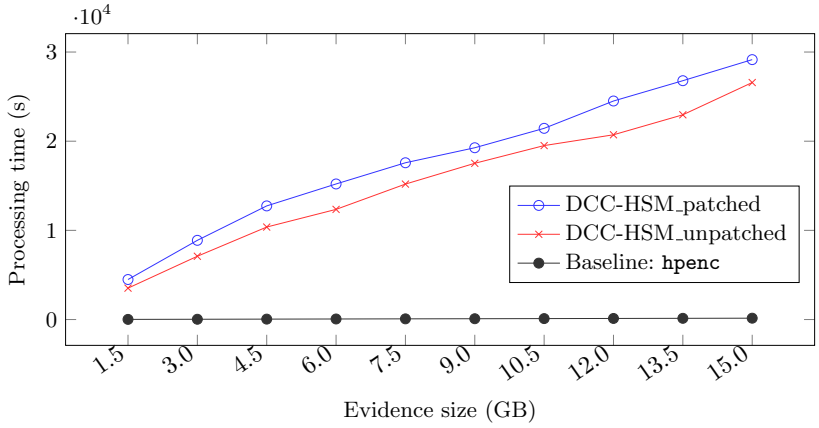


Fig. 6: DCC-HSM processing times for the 1.5-15GB evidence range.

## 7 Conclusions and Future Work

This paper proposed DCC: a live forensics solution for mobile cloud storage obtained through the undercover retrieval of smartphone credentials. In what follows, we revisit the aims and outline future directions.

*CoC requirements.* DCC needs to provide evidence preservation and address mass surveillance concerns through compatibility with CoC requirements. The SEcube proof of concept for Google Drive emphasises the central role of the HSM in being a physical replacement for the missing smartphone. The HSM links directly to the stealthily retrieved passwords by securely storing the corresponding OAuth2 tokens. Being the only source for these tokens also provides controlled access to cloud storage. Therefore, the computed authentication tags go beyond detecting access log tampering to prove comprehensiveness. CoC entries can therefore proceed with a DCC-HSM-centric approach.

*Security.* Having validated the functionality aspect of the DCC-HSM through the Google Drive case study, the focus must now shift to security. Besides further security analysis of the current DCC-HSM, the missing DCC components that provide session key protection must be looked into. In this regard, a tamper-evident approach to system logs [23], along with a seamless process to harden stock cloud forensics tools against loading of additional code widely used by modern web browsers[13] are both in the pipeline.

[13] https://blogs.windows.com/msedgedev/2017/02/23/mitigating-arbitrary-native-code-execution/

*Practicality.* Our first DCC-HSM prototype demonstrates that DCC is practical even though the additional evidence collection time is a factor. Performance results show that the SEcube HSM can process 15GB of evidence within a single working day. Yet provisions may be necessary in case of evidence tampering suspicions from the device owner's end, e.g. use the stolen password to lock owners out of their account through a password change. Avenues for speeding up the process are still available. For example, in SEcube's case, we still haven't leveraged the FPGA, while an upgrade to USB 3.0 can provide an immediate performance gain. Another issue may be posed by two-factor authentication (2FA) that would block DCC in its first stage. DCC can be updated to directly retrieve the OAuth2 tokens, although this enhancement mainly depends on what is possible in terms of undercover credentials theft. Alternatively, an extended permanence of the undercover operation tool can hijack all 2FA operations.

*Final Remark.* Besides the evidence preservation role, CoC compatibility is also needed for the 'right to protect personal data' purposes. From a technical perspective, the DCC proposition includes tamper-evident log controls, specifically to allow investigators to demonstrate the case-relevant usage of the stealthily obtained credentials. However, this aspect goes beyond the technical means of CoC compatibility. Once the security-related components of DCC are also in place, the focus can shift to legal matters, with field studies becoming necessary.

## References

1. Ab Rahman, N.H., Cahyani, N.D.W., Choo, K.K.R.: Cloud incident handling and forensic-by-design: cloud storage as a case study. Concurrency and Computation: Practice and Experience **29**(14), e3868 (2017)
2. Arthur, W., Challener, D., Goldman, K.: A practical guide to TPM 2.0: Using the new trusted platform module in the new age of security. Springer Nature (2015)
3. Cahyani, N.D.W., Ab Rahman, N.H., Glisson, W.B., Choo, K.K.R.: The role of mobile forensics in terrorism investigations involving the use of cloud storage service and communication apps. Mobile Networks and Applications **22**(2), 240–254 (2017)
4. Casey, E.: Handbook of digital forensics and investigation. Academic Press (2009)
5. Chernyshev, M., Zeadally, S., Baig, Z., Woodward, A.: Mobile forensics: Advances, challenges, and research opportunities. IEEE Security & Privacy **15**(6), 42–51 (2017)
6. Cosic, J., Baca, M.: A framework to (im) prove " chain of custody" in digital investigation process. In: Central European Conference on Information and Intelligent Systems. p. 435. Faculty of Organization and Informatics Varazdin (2010)
7. Hardt, D., et al.: The OAuth 2.0 authorization framework (2012)
8. Jauernig, P., Sadeghi, A.R., Stapf, E.: Trusted execution environments: properties, applications, and challenges. IEEE Security & Privacy **18**(2), 56–60 (2020)
9. Kotzias, P., Caballero, J., Bilge, L.: How did that get in my phone? Unwanted app distribution on Android devices. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 53–69. IEEE (2021)

10. Kumar, G., Saha, R., Lal, C., Conti, M.: Internet-of-Forensic (IoF): A blockchain based digital forensics framework for IoT applications. Future Generation Computer Systems **120**, 13–25 (2021)
11. La Porta, L.: Trojan malware infecting 17 apps on the app store. Retrieved January 2022 pp. https://www.jamf.com/blog/ios–trojan–malware/ (2019)
12. Li, M., Lal, C., Conti, M., Hu, D.: Lechain: A blockchain-based lawful evidence management scheme for digital forensics. Future Generation Computer Systems **115**, 406–420 (2021)
13. Liu, F., Liu, K.S., Chang, C., Wang, Y.: Research on the technology of iOS jailbreak. In: 2016 Sixth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC). pp. 644–647. IEEE (2016)
14. Lone, A.H., Mir, R.N.: Forensic-chain: Blockchain based digital forensics chain of custody with poc in hyperledger composer. Digital investigation **28**, 44–55 (2019)
15. Mahalik, H., Tamma, R., Bommisetty, S.: Practical mobile forensics. Packt Publishing Ltd (2016)
16. Manral, B., Somani, G., Choo, K.K.R., Conti, M., Gaur, M.S.: A systematic survey on cloud forensics challenges, solutions, and future directions. ACM Computing Surveys (CSUR) **52**(6), 1–38 (2019)
17. Meng, H., Thing, V.L., Cheng, Y., Dai, Z., Zhang, L.: A survey of Android exploits in the wild. Computers & Security **76**, 71–91 (2018)
18. Moussa, A.N., Ithnin, N., Zainal, A.: CFaaS: bilaterally agreed evidence collection. Journal of Cloud Computing **7**(1), 1–19 (2018)
19. Mylonas, A., Meletiadis, V., Tsoumas, B., Mitrou, L., Gritzalis, D.: Smartphone forensics: A proactive investigation scheme for evidence acquisition. In: IFIP International Information Security Conference. pp. 249–260. Springer (2012)
20. Nieto, A., Rios, R., Lopez, J., Ren, W., Wang, L., Choo, K.K.R., Xhafa, F.: Privacy-aware digital forensics. (2019)
21. Sherr, M., Shah, G., Cronin, E., Clark, S., Blaze, M.: Can they hear me now? A security analysis of law enforcement wiretaps. In: Proceedings of the 16th ACM conference on Computer and communications security. pp. 512–523 (2009)
22. Siry, L.: Cloudy days ahead: Cross-border evidence collection and its impact on the rights of eu citizens. New Journal of European Criminal Law **10**(3), 227–250 (2019)
23. Soriano-Salvador, E., Guardiola-Múzquiz, G.: SealFS: Storage-based tamper-evident logging. Computers & Security p. 102325 (2021)
24. Standard, O.: PKCS# 11 cryptographic token interface base specification version 2.40 (2015)
25. Stefanko, L.: Android trojan steals money from PayPal accounts even with 2FA on. Retrieved January 2022 pp. https://www.welivesecurity.com/2018/12/11/android–trojan–steals–money–paypal–accounts–2fa/ (2018)
26. Varriale, A., Vatajelu, E.I., Di Natale, G., Prinetto, P., Trotta, P., Margaria, T.: Secube™: An open-source security platform in a single soc. In: 2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS). pp. 1–6. IEEE (2016)
27. Yaswant, A.: GriftHorse Android trojan steals millions from over 10 million victims globally. Retrieved January 2022 pp. https://blog.zimperium.com/grifthorse–android–trojan–steals–millions–from–over–10–million–victims–globally/ (2021)
28. Zawoad, S., Dutta, A.K., Hasan, R.: Towards building forensics enabled cloud through secure logging-as-a-service. IEEE Transactions on Dependable and Secure Computing **13**(2), 148–162 (2015)