The Overhead of Confidentiality and Client-side Encryption in Cloud Storage Systems

Eric Henziger
Linköping University
Linköping, Sweden

Niklas Carlsson
Linköping University
Linköping, Sweden

ABSTRACT

Client-side encryption (CSE) is important to ensure that only the intended users have access to information stored in public cloud services. However, CSE complicates file synchronization methods such as deduplication and delta encoding, important to reduce the large network bandwidth overheads associated with cloud storage services. To investigate the overhead penalty associated with CSE, in this paper, we present a comprehensive overhead analysis that includes empirical experiments using four popular CSE services (CSEs) and four popular non-CSEs. Our results show that existing CSEs are able to implement CSE together with bandwidth saving features such as compression and deduplication with low additional overhead compared to the non-CSEs. The most noticeable differences between CSEs and non-CSEs are instead related to whether they implement delta encoding and how effectively such solutions are implemented. In particular, fewer CSEs than non-CSEs implement delta encoding, and the bandwidth saving differences between the applications that implement delta encoding can be substantial.

1 INTRODUCTION

Since being introduced, there has been a rapid growth in the use of cloud storage applications. Today, popular services such as Dropbox, Google Drive, Microsoft OneDrive, and iCloud each have hundreds of millions of active users each month. Cloud storage services have also changed how people store and accesses important data. Today, many users use these services to transparently back up file data, with many services allowing users to easily access the files using all their devices, regardless of geographical location.

However, while these services typically provide flexible low-cost synchronization and high accessibility of the data, most services require users to fully trust their cloud providers with the data and do not provide any guarantees regarding the confidentiality and integrity of the data stored. In fact, many services are fairly blunt regarding the lack of confidentiality they provide. For example, by accepting the terms in Dropbox's end-user agreements, the user agrees to give them, including their affiliates and trusted third parties, the right to access, store and scan the data [11]. Similarly, agreeing to Google's terms of service [16] gives Google "a worldwide license to use, host, store, reproduce, modify, [...], publicly display and distribute such content." where "such content" refers to the user's stored content. Clearly, granting such rights might not be acceptable for some users and content. Moreover, with software bugs such as the ones that allowed hackers to log in to Dropbox accounts without the correct password [1] or implementation of government surveillance backdoors such as the NSA Prism program [19], the need for stronger data and privacy protection is expected to increase.

A solution to provide confidential cloud storage is to use client-side encryption (CSE). With CSE, the user's data is encrypted before being transferred to the cloud provider. This makes sure that the content is transferred and stored in an encrypted format and helps ensure that only the clients with the appropriate decryption keys have access to the non-encrypted information. However, CSE complicates file synchronization techniques, such as deduplication and delta encoding, commonly used to reduce the traffic overheads associated with personal cloud storage systems.

To investigate the potential overhead penalty associated with CSE, this paper presents empirical experiments and analysis of CSE-related overheads. Experiments are used to compare and contrast the security and bandwidth saving features implemented by both CSE services (CSEs) and non-CSEs, to compare non-traffic related client-side overheads (e.g., CPU, disk, memory), and to demonstrate some weaknesses in existing delta encoding solutions. To the best of our knowledge, this is the first research paper that focuses on the difference between CSE and non-CSE supporting services. We next break down our contributions into three parts.

First, we present controlled experiments comparing what security and bandwidth saving features that four popular CSEs (Mega, Sync.com, SpiderOak, Tresorit) and four popular non-CSEs (Dropbox, iCloud, Google Drive, Microsoft OneDrive) have implemented. Interestingly, beyond differences in the underlying server infrastructure and whether services provide CSE, there is no clear differences in the features implemented. Instead, we observe large variations within both categories. In fact, only Dropbox (non-CSE) and

UCC '19, December 2–5, 2019, Auckland, New Zealand
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

SpiderOak (CSE) implement all three bandwidth saving features considered here (i.e., compression, deduplication, and delta encoding). Furthermore, despite effective delta encoding perhaps being most important for bandwidth savings, only three services (the two above and iCloud) implement some form of delta encoding.

Second, to glean some insights whether there are other resource overheads associated with CSEs, we present performance measurements, focusing primarily on CPU, disk, and memory usage. Again, we observed no obvious penalty associated with the CSEs. Instead, overheads appear to depend more on what other features are implemented. For example, the services implementing the most traffic reducing features in each category (Dropbox and SpiderOak) sees the highest non-traffic related client-side resource overheads.

Finally, we use targeted experiments to illustrate the delta encoding problem associated with CSEs, placing particular focus on synchronization between multiple devices. The experiments show that much of the bandwidth and storage overheads associated with CSEs are due to CSE cloud providers not being able to un-encode delta encoding messages, and highlights that there are significant differences in the effectiveness in how delta encoding is implemented and that there is much room for improvements.

Outline: Section 2 describes the services analyzed. Section 3 presents a head-to-head comparison of the different state-of-the-art CSEs and non-CSEs, including an analysis of their security features, bandwidth saving features, and client-side performance. Section 4 presents targeted empirical experiments to look closer at the delta-encoding problem associated with CSEs. Finally, related work and conclusions are presented in Sections 5 and 6, respectively.

2 SERVICES EVALUATED

To allow a broad comparison, we evaluated four CSEs (**Mega**, **Sync.com**, **SpiderOak**, **Tresorit**) against four popular non-CSEs (**Dropbox**, **iCloud**, **Google Drive**, **Microsoft OneDrive**). The four non-CSEs provide a nice baseline as they are among the most popular cloud services, each with hundreds of millions of users, and often are considered in the related literature (Section 5). All four non-CSEs have globally distributed servers, providing good access from our EU-based location. In contrast, for two of the CSEs (Sync.com, SpiderOak), the closest servers were located in North America (Toronto, Canada and mid-western USA, respectively). The four CSEs were selected based on recommendations in online reviews (e.g., [4]), claimed CSE properties, and high prevalence in past measurements on a university campus network [20].[1] In general, however, the popularity as measured by the 2018 Alexa ranks of the CSEs (201; 42,412; 123,448; 140,711; respectively) are lower than the corresponding ranks of the non-CSEs (109; 370; 83,106; 45,006). We next describe the four CSEs and the client-side encryption they provide.

Mega is the only of the services that makes their source code public [38] and the only that uses HTTP rather than HTTPS. While HTTPS is an option, Mega argues (in their settings) that since the payload is encrypted, unauthorized access to the user's data is prevented anyway. For encryption, Mega uses symmetric cryptography based on AES-128 [37].

Sync.com implements a zero-knowledge policy [46] using asymmetric encryption. For each user, a 2048 bit RSA private encryption key is used to encrypt the user's AES encryption keys, used to encrypt file data. The private key is itself encrypted with 256 bit AES Galois counter mode, locked using the user's password, stretched with Password-Based Key Derivation Function 2 (PBKDF2).

SpiderOak uses AES-256-CFB to encrypt user data [44]. Furthermore, every file and folder is encrypted with a unique key, and through the use of different keys for different versions of a single file, SpiderOak implements support for versioned file retrieval. The collection of encryption keys are secured by the user's password, hashed and salted using PBKDF2. During file backup, SpiderOak makes an encrypted file copy that it temporarily writes to the local hard drive [47].

Finally, Tresorit uses AES-256 in cipher feedback mode to encrypt user data. Other popular CSEs include pCloud and Sugarsync. There also exist hybrid CSE-based solutions, including cloud encryption gateways (e.g., BoxCryptor) that encrypts all data before placing the encrypted data in the cloud storage folder (e.g., Dropbox). Although BoxCryptor integrates nicely with Dropbox, BoxCryptor and similar services renders some of the cloud providers performance features (e.g., compression and delta encoding) useless. In this paper, we focus on pure CSEs.

3 CSE vs NON-CSE: FEATURES AND PERFORMANCE

This section provides a high-level comparison of the features implemented and the performance observed.

3.1 Methodology and test environment

Our experiments were performed using a Macbook Air, running macOS High Sierra version 10.13.3 on a 1.3 GHz Intel Core i5 CPU with two physical cores, 8 GB RAM, and 128 GB SSD. The laptop was connected to a high-speed university network through a 10 Gb/s Thunderbolt to Ethernet adapter.

[1] For example, during the week Oct. 11-17, 2015, we observed tens of thousands of HTTPS connection to each of the three CSEs with a web interface (Mega, Sync.com, Tresorit). These services were all among the 2,000 domains with the most connections during that week. Since SpiderOak did not have a web interface to its cloud service, the dataset did not capture any traffic to/from its cloud service. Yet, we did observe 17 HTTPS connections to its website, suggesting (at least) local interest in the service.

Client configurations: To the greatest extent possible, all clients were run with default settings. This included always using the latest client version, and is consistent with what a typical user would have seen during spring 2018, when we conducted our experiments, as only Mega and Sync.com at that time allowed clients to disable automatic updates. The main exceptions that required some (re)configurations were some basic configurations needed for the automated test cases and the disabling of SpiderOak's LAN-Sync feature (which otherwise interfered with the laptop's firewall).

Baseline methodology: Most experiments were conducted by adding files to the cloud services' sync folders and performing targeted system and network measurements during the sync process. The methodology was based on that by Bocchi et al. [3]. We first extended and modified the benchmarking scripts provided by the authors to suit our test environment. For example, since we run the scripts on the same machine as the clients the files could simply be copied using the function `shutil.copy2()`, rather than involving an FTP server to move files between folders on different VMs.

Testing one application at a time, the network traffic was recorded throughout the entire sync process using the Python modules `netifaces` and `pcapy`, among others. The packet capture was executed as a separate thread to allow concurrency between the packet capture process and the main test procedure. To measure CPU, memory and network utilization, the Python module `psutil` was used. These measurements were executed in a dedicated thread and provided per-process measurements at a 40 ms granularity. Some services ran multiple processes; e.g., Dropbox ran three processes and Tresorit two. In these cases, our scripts aggregated the final results over the corresponding processes. To minimize the background traffic and number of competing processes, we closed all programs except the sync client under test.

Finally, on our macBook, iCloud is more tightly integrated with the OS than the other applications. For this reason, rather than starting/closing the application between each test, for iCloud, we kept the application running throughout a series of tests, while monitoring the three processes that we determined were associated with the iCloud application.

3.2 Basic security properties

Our classification of cloud providers is based on their official claims. However, since only Mega's source code is public, it is difficult to fully validate if and how CSE is actually implemented. Validation is further complicated by all services (except Mega) using Transport Layer Security (TLS) to encrypt the end-to-end communication.

MITM-based CSE sanity checks: To check whether the self-claimed CSEs indeed provide further data protection, we used a man-in-the-middle (MITM) methodology. In particular, we setup the client's traffic to go through a trusted proxy (`mitmproxy` [40]), running on a Ubuntu 17.10 machine, and then added the `mitmproxy` certificate as a trusted root CA to the macOS keychain. While all applications had native support for HTTP proxy configuration, all applications except Mega prevented the TLS connection negotiation from succeeding when a foreign TLS certificate is used. (Mega did display two warning messages regarding the security risks of trusting the mitmproxy certificate before we could connect via the proxy.) Dropbox, Google Drive, and Tresorit all sent a TLS alert message with code 48 (Unknown CA). OneDrive sent an alert message with code 86 (Inappropriate Fallback). Finally, SpiderOak and Sync.com did not send any TLS alert messages, but (similar to the others) did not allow the TLS negotiation to be completed. The above MITM prevention behavior appears to be due to the use of certificate pinning and/or similar techniques, as claimed to be implemented by Dropbox and SpiderOak (e.g., [10, 44]).

Clearly, the use of certificate pinning prevents us from easily sanity check services' CSE claims. However, in addition to their native applications, all services except SpiderOak also provide a web interface. For these services the trusted proxy approach was therefore successful. Here, we set up a Firefox browser to trust the `mitmproxy` and extracted all HTTP messages delivered over TLS (using HTTPS). For our experiments, we then upload text files through the web interface of each service, identify the corresponding POST requests, and inspect the payload.

While encryption still complicates validation, we did not find any signs that the CSEs did not properly encrypt the data. In contrast, within the TLS connection, Dropbox, iCloud and OneDrive sent the data in plain text, and Google Drive encoded it in base64. Based on these tests we were sufficiently convinced that the CSEs indeed provide some further encryption (or obfuscation) to help protect client data. We did not try to validate the use of claimed encryption algorithms and/or where different keys are stored. While bugs or implementation artifacts that give cloud providers access to the decryption keys would render these services useless for users not okay giving the provider data access, we leave such investigation for future work. Instead, in the remainder, we compare and contrast how other features differ between CSEs and non-CSEs.

Connection security: Not all entities along the internet path between a client and the cloud storage may be trusted. Combined with increasing use of HTTPS, it is therefore perhaps not surprising that all clients considered here support TLS. One standout, however, is Mega, who does not have TLS enabled by default. Furthermore, with exception of Mega and SpiderOak (both run TLS v1.0), all clients run TLS v1.2, preventing known exploits against TLS v1.0 [12, 41].

While the focus in this paper is on the native applications, a service is only as strong as its weakest interface. When

Table 1: Summary of bandwidth saving features

		Feature/capability		
	Services	Compression	Deduplication	Delta Sync
non-CSE	Dropbox	**Yes**	**Yes**	**Yes**
	iCloud	No	**Yes**	**Yes**
	Google Drive	**Conditional**	No	No
	OneDrive	No	**Sometimes**	No
CSE	Mega	No	**Yes**	No
	Sync.com	No	**Yes**	No
	SpiderOak	**Yes**	**Yes**	**Yes**
	Tresorit	**Yes**	No	No

briefly summarizing the keys and certificates used, we therefore consider both applications and web interfaces. First, all services use state-of-the-art signature algorithms (SHA256 with RSA or SHA256+ECC) and typically use RSA 2048 (or corresponding EC) together with AES 128/256 for transfer. Second, while three out of four non-CSEs (Dropbox, iCloud, Google Drive) delivered signed certificate timestamps with their certificates, none of the CSEs implemented such certificate transparency functionality [27, 42] yet (Nov. 2017). Third, Google Drive and SpiderOak use self-signed certificates for the application, while Google Drive and iCloud use it for the web interface. While self-signed certificates come with risks [50], Google's and Apple's use is perhaps not surprising since they operate their own CAs (chaining back to GlobalSign and GeoTrust, respectively). SpiderOak's use of such certificates raises some concerns. (SpiderOak does not have a web interface but uses RapidSSL as CA, signed by DigiCert, for its website. The iCloud application uses Amazon certificates issued by Digicert.) Among the other services, only Sync.com (Comodo+RapidSSL) and Tresorit (GoDaddy+Microsoft) use a mix of CAs. Mega uses Comodo (chaining back to AddTrust). The remaining services use the same chain for both application and website, always chaining back to Digicert (Dropbox directly and OneDrive via Microsoft and Baltimore CyberTrust Root).

3.3 Bandwidth saving features

We next present test results determining whether each application implements three bandwidth saving features: compression, deduplication, and delta encoding. Table 1 summarizes these results.

Compression: Half of the non-CSEs (Dropbox, Google Drive) and half of the CSEs (SpiderOak, Tresorit) used compression. For these tests, plain text files were added to the sync folders, while measuring the bytes uploaded to the cloud. If the number of uploaded bytes was less than the file size, we attributed the difference to compression. The use of plain text allows efficient compression and provides easy validation whether compression was used.

Figure 1 shows example results where we vary the file size of the original files from 10 MB to 28 MB. For each data point,

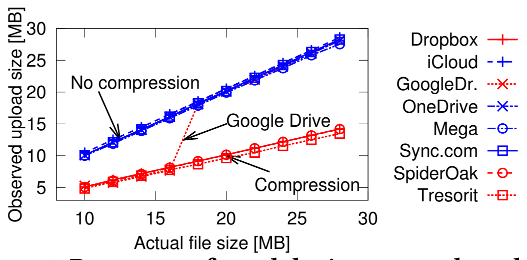

Figure 1: Bytes transferred during example uploads.

we report average values over 15 tests. Three distinct behaviors can be identified. First, four services (iCloud, OneDrive, Mega, Sync.com) did not use compression at all. Second, three services (Dropbox, SpiderOak, Tresorit) applied compression with a compression ratio close to two, regardless of file size. Finally, Google Drive used compression (with similar compression ratio as the other services using compression) for file sizes up to 2^{24} bytes (or 16 MB), but no compression for file sizes beyond this threshold. The exact threshold was identified using binary search. In summary, we observed no significant differences in the usage or effectiveness of the compression used by CSEs and non-CSEs.

Deduplication: Despite seemingly easier to implement in non-CSEs, deduplication is implemented in equally many CSEs (Mega, SpiderOak, Sync.com) and non-CSEs (Dropbox, iCloud, OneDrive). To test for use of deduplication, we performed four scenario-based tests. In each scenario, a 20 MB file made up of random bytes, referred to as the "original" file, was first placed in the sync folder of the application under test. Then, a second file with identical content was uploaded. In the first three scenarios, the second file was identical with the exception that it (i) had a different name, (ii) was uploaded to a different folder, or (iii) both had a different name and was uploaded to a different folder. Finally, in the fourth scenario, the original file was instead deleted and then re-uploaded. For all tests, if the second upload required as much data to be transferred as the original upload, this indicated that deduplication was not used. However, if the second upload resulted in much fewer bytes being transferred, but was still accessible in the desired format, deduplication was used.

The deduplication results were consistent for all services except OneDrive. For this reason, we ran our tests 15 times per service for all services except OneDrive, for which we ran 40 tests per scenario. Table 2 summarizes our deduplication results. We note that Dropbox, iCloud, Mega, SpiderOak, and Sync.com all performed client-side deduplication. None of these services upload an identical file that is already stored in the cloud, albeit in another folder and/or with another name. In contrast, Google Drive, OneDrive and Tresorit all re-upload identical files. However, OneDrive stands out for the fourth scenario ("deletion and re-upload"). For this scenario, OneDrive did not re-upload the file in 12 out of 40 (30%) of the test runs, but instead must have "undeleted" the file in

Table 2: Deduplication test results.

	Deduplication Scenarios			
Service	Name	Folder	Name+Folder	Delete+upload
Dropbox	**Yes**	**Yes**	**Yes**	**Yes**
Google Drive	No	No	No	No
OneDrive	No	No	No	**Sometimes**
iCloud	**Yes**	**Yes**	**Yes**	**Yes**
Mega	**Yes**	**Yes**	**Yes**	**Yes**
SpiderOak	**Yes**	**Yes**	**Yes**	**Yes**
Sync.com	**Yes**	**Yes**	**Yes**	**Yes**
Treosorit	No	No	No	No

the cloud storage. This was, however, the only inconsistency found during the tests. For all other clients and scenarios, deduplication either occurred for all tests or not at all.

Basic delta encoding: One CSE service (SpiderOak) and two non-CSEs (Dropbox, iCloud) implement delta encoding. To determine whether or not each service used (at least) some form of delta encoding, all clients underwent three basic tests. In all tests, we started with a 5 MB file, that we incrementally increased the size of in steps of 5 MB until the size reached 25 MB. In each step, we inserted 5 MB random bytes (i) at the end (append), (ii) at the beginning (prepend), or (iii) into a random position (random insert) of the file. For each test, we again measured the number of uploaded bytes. For these scenarios, in the ideal case, the number of uploaded bytes (in each step) by a client using delta encoding would be similar to the size of the change; i.e., 5 MB. On the other hand, for clients that did not take advantage of delta encoding the number of uploaded bytes is expected to be close to the file size after each modification; i.e., 5, 10, 15, 20, 25 MB. While the results of these tests show that CSEs (exemplified by SpiderOak) can implement delta encoding, they say nothing about the relative effectiveness of the delta encodings implemented by the three services. Section 4 presents a more detailed analysis of the clients that perform delta encoding and demonstrates the additional delta encoding overheads inherent to CSEs.

3.4 Performance evaluation

To shed some initial light whether or not there is a significant performance penalty of CSE, we next present basic client-side performance results.

CPU utilization: The CPU utilization differ significantly between the eight services, but also varies a lot over time. This is illustrated in Figure 2, where we show the CPU utilization and uploaded bytes as a function of time, during example uploads for each service. For comparison, we therefore identify and report statistics for four phases: idle, pre-processing, transfer, and cooldown. A client is classified as idle when it is up-to-date with the cloud storage and not yet actively syncing. The pre-processing phase begins when a file is copied into the sync folder and continues until the upload starts. The transfer phase lasts from this time until all data is fully uploaded, after which we include a 5 second cooldown phase, before considering the client back in idle.

For each application, we uploaded a 30 MB file containing random data, with each experiment repeated 25 times, and report 95% confidence intervals. Figures 3(a) and 3(b) show the average utilizations observed during the idle/cooldown phases and the more CPU intensive phases (pre-processing, transfer), respectively, and Figure 3(c) shows the aggregate non-idle CPU volumes during these CPU intensive phases. Here, the non-idle CPU volume is defined as the time-integral over the additional CPU utilizations during the respective phases, and can easily be calculated by multiplying the duration of the phase with the difference between the average CPU utilization during the phase minus when idle. Ignoring the (small) idle values, 50% utilization over 2 seconds results in the same CPU volume as 100% over 1 second.

In general, we do not observe any significant penalty to CSE. Instead, the CPU usage is highest for the feature-rich services (i.e., Dropbox, SpiderOak) that support all three of the tested capabilities (Table 1). Dropbox has the highest CPU volume across all services, and with the exception of SpiderOak, the CSEs typically have CPU volumes in-between Dropbox and the less feature-rich non-CSEs (Google Drive, OneDrive), but more similar to iCloud (who implements a subset of the bandwidth saving features in Table 1).

A few additional things stand out. First, comparing Dropbox and SpiderOak (only applications implementing all three bandwidth saving features), we note that Dropbox had a significantly higher CPU utilization during the pre-processing phase, but that SpiderOak had by far the highest CPU volume. The reason for this is that Dropbox was much faster at detecting the file change, and could start the processing and file transfer almost immediately. Second, the average CPU utilization during transfer for Dropbox and Tresorit were significantly higher than for the other clients. The values above 100% indicates that the clients are multithreaded and that at least two threads of the application were heavily utilizing separate CPU cores simultaneously. Finally, SpiderOak was the only client with a lower CPU volume during transfer compared to during pre-processing. This is likely due to SpiderOak creating a temporary copy of the file, compress the file, check for duplication, and encrypt the file before uploading can begin. In sharp contrast, the transfer utilization for Mega was significantly higher than its pre-processing utilization. A closer look at the transfer phase suggests that Mega alternate between applying encryption and uploading data to the cloud.

CPU volume under matching network conditions: All services except SpiderOak and Sync.com had data centers in Europe (where the experiments were performed). To reduce the impact of location differences, we used the network link simulator tool `Network Link Conditioner` [36] to set

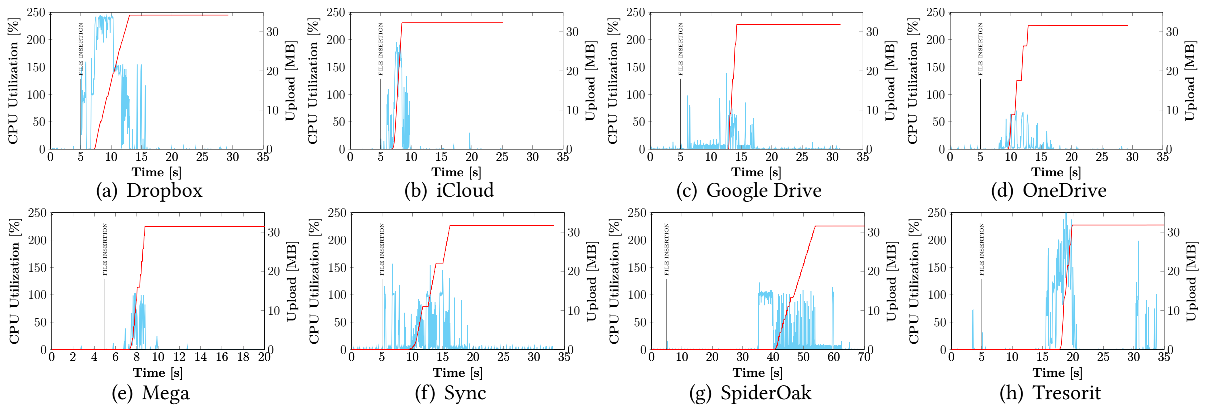
(a) Dropbox (b) iCloud (c) Google Drive (d) OneDrive (e) Mega (f) Sync (g) SpiderOak (h) Tresorit

Figure 2: CPU utilizations during example runs. Top-row are non-CSEs and bottom row are CSEs. Also, note the different time-scales for Mega (shorter) and SpiderOak (longer) than the rest.

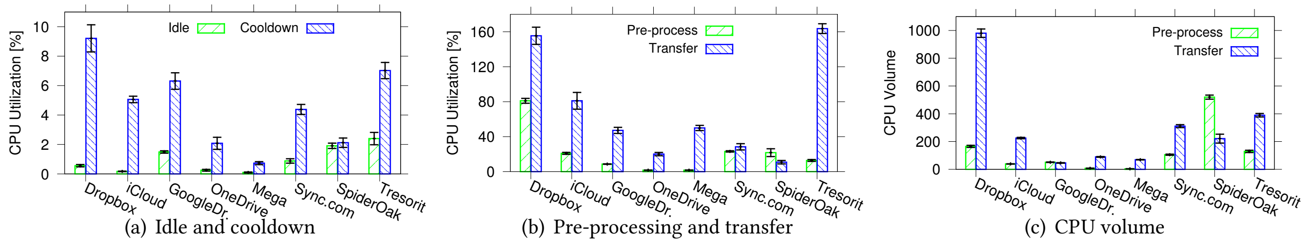
(a) Idle and cooldown (b) Pre-processing and transfer (c) CPU volume

Figure 3: CPU utilizations for different services.

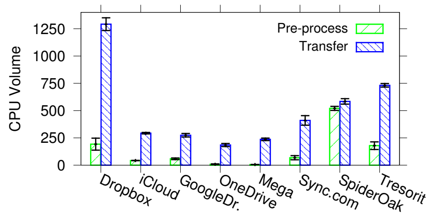
Figure 4: CPU volumes with equalized conditions.

bandwidth and latencies of the network interface so that the bandwidth bottleneck (10 Mbps) was on the client interface and the RTTs matched the RTT (145ms) of the service with the largest RTTs (SpiderOak).

Figure 4 shows the CPU volume results for these experiments. Compared with the original results (Figure 3(c)), the CPU volumes during the transfer phase have increased for all services. This is due to increased RTTs and introduced bandwidth limitations. Still, the relative performance between the different clients have not changed. Dropbox still has the highest CPU volume during transfer and Google Drive, OneDrive and Mega have the lowest. One exception to this was SpiderOak which surpassed Sync.com (who also has its servers in North America) in CPU volume for the transfer phase. Although SpiderOak was used as the original baseline, we found that the introduced client-side bottleneck hurt SpiderOak's transfer times more than Sync.com.

HTTP vs HTTPS comparison: To set the above CPU comparisons of CSE vs non-CSE services in perspective, we looked closer at the penalty of using HTTPS (encrypted) rather than HTTP (non-encrypted). We therefore turned to Mega, which was the only service allowing us to switch between using HTTPS (optional) and HTTP (default). Table 3 shows comparison results based on 50 experiments. We note a statistically significant penalty of using HTTPS (as indicated by non-overlapping confidence intervals); however, the differences are small compared to the differences observed across cloud services. This shows that the CPU overheads associated with the bandwidth saving features (that Mega do not implement) may be substantially larger than the CPU overheads associated with using HTTPS. Unless much more complex algorithms are used for CSE, this also helps explain why no noticeable CPU penalty is observed with the CSEs.

Table 3: HTTPS vs HTTP comparison of CPU utilization for Mega. (Idle CPU utilization with/without HTTPS is 0.10 ± 0.02%.)

	CPU utilization (%)		CPU volume	
	Pre-proc.	Transfer	Pre-proc.	Transfer
HTTPS	2.48±0.08	63.41±1.72	5.71±0.20	107.98±1.21
HTTP	1.72±0.06	42.91±2.55	3.61±0.12	58.70±3.96

Disk usage: We have only observed per-file disk usage for SpiderOak. Figure 5 shows the averages number of bytes written to disk (with 95% confidence intervals) by any active process during the syncing of a 300 MB file, as calculated over 40 experiments per application. For these measurements, we used the `psutil` module, and since macOS did not have

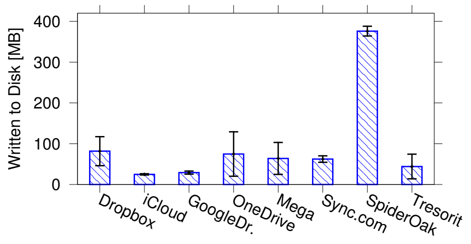

Figure 5: Bytes written to disk during a 300 MB upload.

support for per-process I/O counters, the measurements were performed on OS level (rather than per process). In contrast to the other services (that do not appear to write to disk, and if they do only write a small amount), the writes during a SpiderOak transfer exceed the file size (300 MB) plus an excess amount typically exceeding the writes of the other applications (capturing SpiderOak's longer transfer times).

Memory usage: Although significant individual differences, we have not observed any systematic differences in the memory usage between CSEs and non-CSEs. Instead, differences in the memory profiles (e.g., see example traces in Figure 6) mostly appear to be due to implementation differences and the memory footprints are relatively stable (e.g., comparing footprints during idle and active states).

Of the services we tested, Dropbox (2.89%) and Google Drive (2.72%) had the largest memory usage, and none of the services appears to keep a full copy of the files in memory. For example, when uploading five large files, each with 300 MB random data (to minimize the risk of deduplication), with each upload separated by roughly 50 seconds, the maximum memory usage of any service had a memory usage of roughly 240MB (or 3% on our 8GB system). Dropbox (2.89%) and SpiderOak (1.99%) again stand out, as they again are among the three applications with the largest memory footprints. While Google Drive (2.72%) does not implement the bandwidth saving features we tested for, we expect that the larger memory footprint is due to other services that it offers. Furthermore, compared to their respective idle levels, the largest increase in memory utilization was small. For example, taking the average over 12 runs per service, only four services had an increase larger than 0.25% (or 20 MB). These were Dropbox (0.68%), Tresorit (0.60%), Sync.com (0.58%), and Google Drive (0.55%). Finally, while most services see a slight drift in memory footprint over time, this drift was only substantial for Sync.com, for which the average footprint (per upload) increased from 0.98% to 1.27%, and Google Drive (2.44% to 2.72%). For the other services the average differences between the uploads remained within 0.04%.

4 DELTA ENCODING ANALYSIS

While the CSEs that we have studied have been equally successful as the top-four non-CSEs to achieve bandwidth savings using compression and deduplication, it is much harder for CSEs to implement effective delta encoding. This is perhaps why only SpiderOak of the tested CSEs implement delta encoding, and why, as we will show here, both Dropbox and iCloud (the two non-CSEs performing delta encoding) significantly outperform SpiderOak.

The delta encoding problem with CSEs: Delta encoding is made difficult for CSEs mainly by the cloud provider not having access to the non-encrypted data and delta encoding being extremely inefficient on encrypted file versions. Therefore, to allow the provider to seamlessly share the file with other devices of the client there are two main alternatives: (i) the client always upload the full file whenever they make a change, ensuring that the provider always has the latest copy to deliver, or (ii) the client submit encrypted versions of delta encodings that the provider can store and deliver together with the original encrypted file.[2] The first option comes at significant upload overhead, since even a very small change result in the full file (or block) being uploaded. In contrast, the second option has low upload overhead, but much larger storage and download bandwidth overhead, since the provider must store and deliver the full change-log sequence needed by the downloading device to recreate the most recent file copy.

Referring back to Table 1, three out of the four CSEs (Mega, Sync.com, Tresorit) do not use delta encoding, but instead replace the full file when changes are made. We again note that this approach (option one above) can be extremely inefficient when changes are small. In fact, it is possible to show that the solution can be arbitrarily bad. For example, consider a small file change to a file of size N that requires a delta encoding of size Δ. In this case, a CSE replacing the full file would require an upload bandwidth proportional to N, whereas one that uses delta encoding would only need to upload Δ, resulting in a relative penalty of $\frac{cN-\Delta}{\Delta}$. This penalty is unbounded when $N \rightarrow \infty$ and also becomes very large when Δ is small.

SpiderOak's bandwidth and server-side storage overhead: Among the CSEs, only SpiderOak performs delta encoding. However, their implementation is proprietary, making it non-trivial to analyze all details of their solution. Here, we use targeted experiments to provide initial insights into their delta encoding and the associate overheads.

First, and most importantly, it is easy to see that SpiderOak indeed stores a sequence of delta encoding on the server side, and that a second device downloads both the original file and the change-log of delta encodings. For example, consider a basic experiment in which we start with a file of size 10MB, consisting of random bytes, and then modify bytes 0-0.5 MB,

[2]While focus here is on files, it is possible to apply both the above approaches also on a per-block basis. This case typically increases complexity significantly, may reduce confidentiality, and requires the block structure to be passed along with block changes.

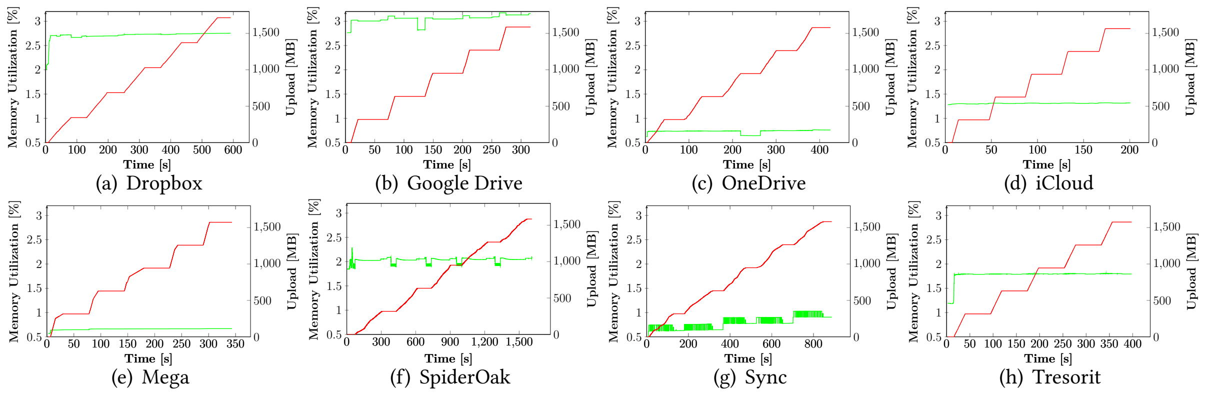

Figure 6: Memory usage during example runs: Non-CSEs (top) and CSEs (bottom). Note the different time scales.

bytes 1-1.5 MB, and so forth over 10 file changes. In this scenario, the original upload was of size 10.049 MB, and the 10 delta encoding updates were (measured in MB): 0.531, 1.058, 1.058, 1.058, 1.058, 0.531, 0.794, 0.794, 0.794, 0.794.[3] In total, this resulted in 18.521 MB uploaded data; 3 MB more than the theoretic bound of 15 MB (if uploading only the size of the original file plus the changed data). When syncing with a second client, we could also confirm that the SpiderOak client indeed downloaded the full (18.5 MB) change log, and then recreated the file as seen on the first client. Again, this download size is expected since a provider should not be able to take advantage of the delta encodings to save server storage or download bandwidth for the second device.

The corresponding tests with Dropbox (non-CSE) looked quite different. While the original upload was somewhat larger (10.578 MB), the following 10 updates were smaller (measured in MB): 0.662, 0.540, 0.535, 0.535, 0.537, 0.535, 0.535, 0.540, 0.537, 0.535. In total, this resulted in 16.070 MB uploaded data. This shows that Dropbox uses more efficient delta encoding, only requiring 1 MB extra overhead. Furthermore, when syncing with a second device, we could also confirm that the second device only had to download 10.353 MB, confirming that Dropbox efficiently applies delta encodings on the servers. The above examples clearly demonstrate some of the added delta encoding overheads required for CSEs.

Random changes comparison with Dropbox+iCloud: Second, SpiderOak's block-based implementation can perform very poorly. To illustrate this we use a simple head-to-head comparison with Dropbox and iCloud in which we randomly picked *n* bytes to change and then measured the number of bytes uploaded by the application. Figure 7 shows the results of these tests, when applied on a 10 MB file. (The insert zooms in on the low parameter range.)

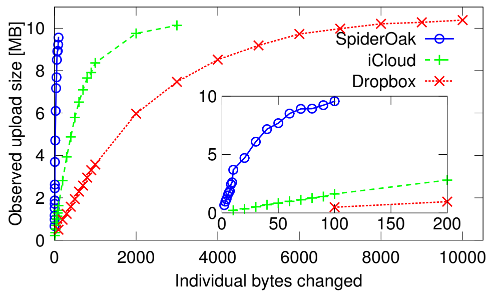

Figure 7: Bytes uploaded during delta-encoding tests with random changes.

These results (i) confirm that the delta encodings work poorly on random file changes (and these techniques would hence not be useful on encrypted file data) and (ii) show that both Dropbox and iCloud significantly outperform SpiderOak. For example, consider the number of random bytes that can be changed before each service have uploaded the equivalent of another 10 MB file. For SpiderOak, on the order of 100 bytes are needed. In contrast, with iCloud and Dropbox approximately 2,000 and 6,000 bytes need to change, respectively. While the large differences partially may be due to implementation differences for sparse use cases, the results show that the room for improvements in SpiderOak's solution is significant. For example, as long as the changes are small, an application could simply encrypt the delta changes that a non-CSE would make, store those changes in the cloud (in encrypted format), and rely on the clients to later download the full sequence of such changes to their other devices, where they themselves can apply the delta changes. We are currently investigating optimized variations of such policies.

5 RELATED WORK

Prior work has not empirically evaluated the impact that CSE has on the performance of existing applications.

[3]A more detailed analysis, not included in this paper due to lack of space, reveals that SpiderOak uses block-based delta encoding, with a block size of 256 kB (plus a small overhead). Here, we simply note that the updates listed here correspond to changes of 2 blocks (0.531 MB), 3 blocks (0.791 MB), or 4 blocks (10.058 MB), respectively.

Most performance studies of personal cloud storage have focused on Dropbox [9, 32] and other popular services [8, 17, 22]. Similar to Section 3.3, this include works that determine whether different services (Dropbox, Google Drive, OneDrive and Box) implement different performance features [5, 8]. Some variation in features have been observed over time and across devices [6, 34]. However, none of these works consider the impact of CSE on the features implemented, the performance obtained, or even evaluated the performance of CSEs. In contrast, we focus on CSEs and their relative performance penalty. Having said that, the above studies provide a nice baseline for three of the most popular non-CSEs (Dropbox, Google Drive, OneDrive) and allows us to contrast our results (we are also first to determine features for these three services using macOS).

Many different types of algorithms have been proposed to reduce cloud storage and bandwidth costs. With early works primarily targeting storage savings, most such works have focused on deduplication [21, 39, 49]. In the context of CSE, the most related works have demonstrated how effective and secure deduplication can be achieved by combining convergent encryption and clever key management [23, 29, 43, 45]. This may at least partially explain why three of the four considered CSEs implement effective deduplication. Other common techniques include delta encoding [28, 32], device-to-device synchronization [15], compression [30], and caching [14]. Yet others have implemented middleware solutions (e.g., that can be used in conjunction with Dropbox) to improve the synchronization process [32, 33].

Wilson and Ateniese [50] provide an overview of CSEs and uncovered some weaknesses when enabling data sharing. Their work focuses on the issuing of certificates, and highlights the problem when the provider acts as a CA for itself. The founders of Tresorit, Lam and Szebeni, have proposed and patented solutions (based on the TGDH protocol) for sharing data in dynamic groups over an untrusted cloud storage service [24–26]. Others, like SpiderOak, revokes their "No Knowledge" policy for files shared through a so called "ShareRoom" [48]. Mager et al. [35] studied the now discontinued CSE service Wuala, and found that, similar to what we find for SpiderOak, Wuala encrypted and stored files locally before syncing the encrypted contents to the cloud.

On the topic of delta encoding, we note that aggregating multiple delta encoding updates before propagating changes can be an efficient way to further save bandwidth [28, 32]. Others have studied the most beneficial user behaviors to exploit when optimizing file sync operations [18] or the client behavior itself [9, 13, 30]. For example, Drago et al. [9] showed that Dropbox primarily is used for small files that are changed frequently, while Li et al. [30] have confirmed that there is a long tail of smaller files also on other services and that most files (84%) are changed at least once. These characteristics confirm the importance of effective delta encoding. Finally, we note that aggregating sync events can be particularly valuable for capped mobile users [2, 7, 31], but also that monitoring and access control may be more complicated in such environments [2]. To the best of our knowledge, we are the first to empirically evaluate the overhead costs observed by popular CSEs.

6 CONCLUSIONS

Client-side encryption (CSE) is important to ensure that only the intended users have access to information stored in public cloud services, but complicates the implementation of bandwidth saving file synchronization features. This paper is the first to focus on the performance overhead of existing CSE services (CSEs). Using empirical experiments with four popular CSEs (Mega, Sync.com, SpiderOak, Tresorit) and the four most popular non-CSEs (Dropbox, iCloud, Google Drive, OneDrive), we characterize the current state-of-the-art and their relative overheads. First, by comparing the security and bandwidth saving features implemented, as well as the performance, of the eight services, we highlight both positives and negatives. On the positive side, bandwidth saving features such as compression and deduplication appears to come with low additional overhead and achieve similar efficiency. Instead, the performance overheads (as measured using CPU utilization, CPU volume, disk writes, and memory footprint) appears to depend on the set of bandwidth saving features implemented, and the main penalty associated with CSE appears to be due to bandwidth, storage, and processing overheads associated with implementing (or not implementing) different forms of delta encoding together with CSE. While helping reduce bandwidth between the client and the servers, services implementing delta encoding typically have significantly higher resource usage on the client. We also observe differences between the CSE (SpiderOak) and the two non-CSEs (Dropbox, iCloud) implementing delta encoding. For example, SpiderOak comes with a higher storage footprint both on the client and on the servers, has higher bandwidth overhead for both uploaders and downloaders, and implements less effective delta encoding than Dropbox and iCloud. Future work include the development of optimized delta encoding policies for CSEs, which minimize the bandwidth and storage overhead associated with CSE, and that close the gap seen compared to non-CSEs (e.g., gaps between SpiderOak and Dropbox+iCloud in Figure 7).

REFERENCES

[1] Arash Ferdowsi - Dropbox Inc. 2011. Yesterday's Authentication Bug. (2011). https://blogs.dropbox.com/dropbox/2011/06/yesterdays-authentication-bug/

[2] Y. Bai and Y. Zhang. 2017. StoArranger: Enabling Efficient Usage of Cloud Storage Services on Mobile Devices. In *Proc. IEEE ICDCS.*

[3] E. Bocchi, I. Drago, and M. Mellia. 2017. Personal Cloud Storage Benchmarks and Comparison. *IEEE Trans. on Cloud Computing* 5, 4 (2017).

[4] Cloudwards. 2018. Best Cloud Storage Providers of 2018. (2018). https://www.cloudwards.net/comparison/

[5] Y. Cui, Z. Lai, and N. Dai. 2016. A First Look At Mobile Cloud Storage Services: Architecture, Experimentation, and Challenges. *IEEE Network* 30, 4 (2016).

[6] Y. Cui, Z. Lai, X. Wang, and N. Dai. 2017. QuickSync: Improving Synchronization Efficiency for Mobile Cloud Storage Services. *IEEE Trans. on Mobile Computing* 16, 12 (2017).

[7] Y. Cui, Z. Lai, X. Wang, N. Dai, and C. Miao. 2015. QuickSync: Improving Synchronization Efficiency for Mobile Cloud Storage Services. In *Proc. ACM MobiCom.*

[8] I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras. 2013. Benchmarking Personal Cloud Storage. In *Proc. IMC.*

[9] I. Drago, M. Mellia, M. M. Munafò, A. Sperotto, R. Sadre, and A. Pras. 2012. Inside Dropbox: Understanding Personal Cloud Storage Services. In *Proc. IMC.*

[10] Dropbox. 2018. Under the hood: Architecture overview. (2018). https://www.dropbox.com/business/trust/security/architecture

[11] Dropbox Inc. 2019. Dropbox Terms of Service. (2019). https://www.dropbox.com/terms

[12] Z. Durumeric et al. 2014. The Matter of Heartbleed. In *Proc. IMC.*

[13] G. Goncalves, I. Drago, A. da Silva, A. B. Vieira, and J. M. Almeida. 2014. Modeling the Dropbox Client Behavior. In *Proc. IEEE ICC.*

[14] G. Gonçalves, I. Drago, A. P. C. Da Silva, A. B. Vieira, and J. M. Almeida. 2016. The impact of content sharing on cloud storage bandwidth consumption. *IEEE Internet Computing* 20, 4 (2016), 26–35.

[15] G. Gonçalves, A. B. Vieira, I. Drago, A. P. C. Da Silva, and J. M. Almeida. 2017. Cost-Benefit Tradeoffs of Content Sharing in Personal Cloud Storage. In *Proc. IEEE MASCOTS.*

[16] Google LLC. 2018. Google Terms of Service. (2018). https://www.google.com/intl/en/policies/terms/

[17] R. Gracia-Tinedo, M. S. Artigas, A. Moreno-Martinez, C. Cotes, and P. G. Lopez. 2013. Actively Measuring Personal Cloud Storage. In *Proc. IEEE CLOUD.*

[18] R. Gracia-Tinedo, Y. Tian, J. Sampe, H. Harkous, J. Lenton, P. G. Lopez, M. Sanchez-Artigas, and M. Vukolic. 2015. Dissecting UbuntuOne: Autopsy of a Global-scale Personal Cloud Back-end. In *Proc. IMC.*

[19] G. Greenwald, E. MacAskill, L. Poitras, S. Ackerman, and D. Rushe. 2013. Microsoft handed the NSA access to encrypted messages. *The Guardian* (2013). https://www.theguardian.com/world/2013/jul/11/microsoft-nsa-collaboration-user-data

[20] J. Gustafsson, G. Overier, M. Arlitt, and N. Carlsson. 2017. A First Look at the CT Landscape: Certificate Transparency Logs in Practice. In *Proc. PAM.*

[21] D. Harnik, B. Pinkas, and A. Shulman-Peleg. 2010. Side Channels in Cloud Services: Deduplication in Cloud Storage. *IEEE Security & Privacy* 8, 6 (2010).

[22] W. Hu, T. Yang, and J. N. Matthews. 2010. The Good, the Bad and the Ugly of Consumer Cloud Storage. *ACM SIGOPS Operating Systems Review* 44, 3 (2010).

[23] J. Hur, D. Koo, Y. Shin, and K. Kang. 2016. Secure data deduplication with dynamic ownership management in cloud storage. *IEEE Trans. on Knowledge and Data Engineering* 28 (2016), 3113–3125.

[24] I. Lam, S. Szebeni, and L. Buttyan. 2012. Invitation-oriented TGDH: Key management for dynamic groups in an asynchronous communication model. In *Proc. IEEE ICPP Workshops.*

[25] I. Lam, S. Szebeni, and L. Buttyan. 2012. Tresorium: Cryptographic file system for dynamic groups over untrusted cloud storage. In *Proc. IEEE ICPP Workshops.*

[26] I. Lam, S. Szebeni, and T. Koczka. 2015. Client-side encryption with DRM. (2015). US Patent 9,129,095.

[27] B. Laurie, A. Langley, and E. Käsper. 2013. *RFC6962: Certificate Transparency.* IETF.

[28] G. Lee, H. Ko, and S. Pack. 2017. An Efficient Delta Synchronization Algorithm for Mobile Cloud Storage Applications. *IEEE Trans. on Services Computing* 10, 3 (2017).

[29] J. Li, X. Chen, M. Li, J. Li, P. P. Lee, and W. Lou. 2013. Secure deduplication with efficient and reliable convergent key management. *IEEE Trans. on Parallel and Distributed Systems* 25, 6 (2013), 1615–1625.

[30] Z. Li, Y. Dai, G. Chen, and Y. Liu. 2014. Towards Network-level Efficiency for Cloud Storage Services. In *Proc. IMC.*

[31] Z. Li, X. Wang, N. Huang, M. A. Kaafar, Z. Li, J. Zhou, G. Xie, and P. Steenkiste. 2016. An Empirical Analysis of a Large-scale Mobile Cloud Storage Service. In *Proc. IMC.*

[32] Z. Li, C. Wilson, Z. Jiang, Y. Liu, B. Y. Zhao, C. Jin, Z.-L. Zhang, and Y. Dai. 2013. Efficient Batched Synchronization in Dropbox-like Cloud Storage Services. In *Proc. ACM/IFIP/USENIX Middleware.*

[33] P. G. Lopez, M. Sanchez-Artigas, S. Toda, C. Cotes, and J. Lenton. 2014. StackSync: Bringing Elasticity to Dropbox-like File Synchronization. In *Proc. ACM Middleware.*

[34] X. Luo, H. Zhou, L. Yu, L. Xue, and Y. Xie. 2016. Characterizing mobile*-box applications. *Computer Networks* 103 (2016).

[35] T. Mager, E. Biersack, and P. Michiardi. 2012. A Measurement Study of the Wuala On-line Storage Service. In *Proc. IEEE P2P.*

[36] Mattt. 2018. Network Link Conditioner. (2018). http://nshipster.com/network-link-conditioner/

[37] MEGA. 2018. MEGA - Developers Documentation. (2018). https://mega.nz/doc

[38] MEGA. 2018. Mega Limited. (2018). https://github.com/meganz

[39] D. T. Meyer and W. J. Bolosky. 2012. A Study of Practical Deduplication. *ACM Trans. on Storage* 7, 4 (2012).

[40] Mitmproxy. 2018. (2018). https://mitmproxy.org/

[41] B. Möller, T. Duong, and K. Kotowicz. 2014. This POODLE Bites: Exploiting the SSL 3.0 Fallback. *Security Advisory* (2014).

[42] C. Nykvist, L. Sjostrom, J. Gustafsson, and N. Carlsson. 2018. Server-side Adoption of Certificate Transparency. In *Proc. PAM.*

[43] P. Puzio, R. Molva, M. Önen, and S. Loureiro. 2013. ClouDedup: Secure deduplication with encrypted data for cloud storage. In *Proc. IEEE CloudCom.*

[44] SpiderOak Inc. 2018. No Knowledge, Secure-by-Default Products. (2018). https://spideroak.com/no-knowledge/

[45] M. Storer, K. Greenan, D. Long, and E. Miller. 2008. Secure data deduplication. In *Proc. ACM Storage Security and Survivability workshop.*

[46] Sync.com Inc. 2015. *Privacy White Paper.* Technical Report. https://www.sync.com/pdf/sync-privacy.pdf

[47] A. Tervort. 2017. Disk Space Use During File Backup - SpiderOak Support. (2017). https://support.spideroak.com/hc/en-us/articles/115001891163-Disk-Space-Use-During-File-Backup

[48] A. Tervort. 2018. ShareRooms and No Knowledge - SpiderOak Support. (2018). https://support.spideroak.com/hc/en-us/articles/115001854223-ShareRooms-and-No-Knowledge

[49] R. N. Widodo, H. Lim, and M. Atiquzzaman. 2017. A new content-defined chunking algorithm for data deduplication in cloud storage. *Future Generation Computer Systems* 71 (2017).

[50] D. C. Wilson and G. Ateniese. 2014. "To Share or not to Share" in Client-Side Encrypted Clouds. In *Proc. ISC.*