12th International Conference on Innovative Internet Community Services (I2CS 2012)
https://dl.gi.de/handle/20.500.12116/18193

Drizzle: The RAIN Prototype

Martin Gilje Jaatun[1], Christian Emil Askeland[2] and Anders Emil Salvesen[2]

[1]Department of Software Engineering, Safety and Security
SINTEF ICT
Trondheim, Norway
Email: Martin.G.Jaatun@sintef.no

[2]Dept. of Telematics, NTNU
Trondheim, Norway

Abstract: Internet communities are moving to the Cloud, but in addition to the advantages regarding cost and convenience, this also means that cloud service providers are increasingly in a position to aggregate large amounts of personal data, which means that it is becoming prudent to develop mechanisms that can contribute to limiting the information available to providers. In this paper we present a prototype cloud security solution for protection against an "honest but curious" cloud provider. The solution is based on splitting up data and distributing it to multiple cloud providers, without encrypting the individual pieces. Our initial tests indicate that our solution is sufficiently efficient for normal use.

1 Introduction

Internet communities are increasingly employing Cloud Computing; social networks such as Facebook may have started out as hosted applications, but are now moving in the direction of offering cloud-based Software-as-a-Service (SaaS) and even Infrastructure-as-a-Service (IaaS) to external customers [RNJ12].

Cloud Computing is a tantalizing concept that promises flexible computing solutions at an affordable price, without requiring large up-front investments in computer hardware. However, since Cloud Computing can be considered the ultimate evolution of outsourcing, there is lingering concern in the target groups when it comes to security – we trust cloud providers to take care of our holiday snapshots, but many stakeholders are not ready to "bet their business on the cloud". To quote Microsoft's John Howie: "...there are things that will never go into Azure, for example, our SAP back end." [GHR+10]

Since internet community members are predestined to disseminate a lot of (not necessarily sensitive) information through community interaction, it becomes even more important to have an improved mechanism for storing files in the cloud, without adding to the data aggregation problem. Many people have tried to tackle the problem of how to securely store files in an untrusted location. Proposed solutions include fully homomorphic encryption [Gen09], secure multiparty computation [BCD+09] and trusted computing using Trusted

Platform Modules [SGR09].

Our path is on a somewhat different tangent, in that we seek to explore a solution that can offer sufficient security through the splitting and dispersion of data in the cloud. The ultimate goal is to achieve this without the use of encryption, both to save processing cost on mobile devices with limited processing power, but also to avoid having to administrate a large number of encryption keys.

2 Related Work

Surprisingly many researchers have ventured to solve secure distributed storage by splitting up data and spreading it in the wind. Singh et al. [SKZ11] present a scheme for *n-out-of-m* secret sharing of data [Sha79], but do not provide an algorithm for the actual splitting of the data to be stored. Another *n-out-of-m* scheme is proposed by Parakh and Kak [PK09], but they do not discuss why their scheme should be better than e.g. the one proposed by Rabin [Rab89]. Furthermore, they do not discuss that cryptographic keys based on passwords tend to be insecure, and that such schemes become even more vulnerable in the event of password reuse. From the field of Grid computing, Luna et al. [LFMB08] present yet another solution based on Rabin's *n-out-of-m* scheme, but add an additional concept of *Quality of Security* (QoSec) to rate individual storage providers. In this manner, they show that the number of partitions (or the n-to-m ratio) can be adjusted depending on the aggregated trustworthiness of the providers.

The RACS system [ALPW10] aims at preventing vendor lock-in and data loss through failures by performing striping of data (in RAID-5 fashion) across multiple cloud providers, but has no privacy or confidentiality goals.

The Mnemosyne [HR02] system offers steganographic storage which "gives a user strong protection against being compelled to disclose (all) its contents." The idea here is not only to hide data from prying eyes, but also to prevent anyone from determining that there is anything hidden in the first place. Mnemosyne encrypts each block, and uses Rabin's Information Dispersal Algorithm [Rab89] to replicate data and avoid data loss. To protect against traffic analysis, random other blocks are also read/written periodically, only to be discarded.

3 RAIN

The Redundant Array of Independent Net-storages (RAIN) was proposed by Jaatun et al. [JNAZ11, ZJV+11, JZA11], using multiple cloud storage and data processing providers in a fairly complex hierarchy. For our prototype we have adopted an architecture very much like the one seen in Figure 2b in the original paper [JNAZ11], except that we have focused solely on the storage part. The resulting architecture is seen in Figure 1.

For our solution, the Command & Control (C&C) node and the RAIN Dashboard are both

installed on a server within an organisation, maintaining the only database of where in the cloud the file parts are stored. This server runs on Apache with SSL for secure uploading of files within the organisation. Placing it in the cloud will allow for better scaling, and will also permit access from multiple locations.

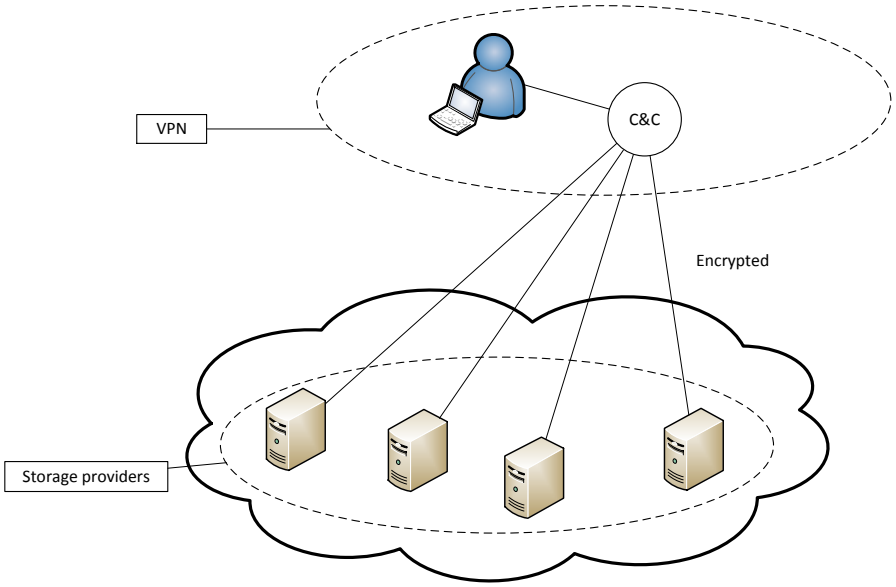

Figure 1: The C&C within a single organisation, communicating with different storage providers.

4 Implementing the C&C

The C&C is the single point of entry for users of RAIN. Users interact with the C&C to split and merge, upload and download files to and from the cloud. It maintains a database, storing all information on files uploaded through it, mainly: *What files were uploaded when by whom?* and *To which providers are the different chunks uploaded?*

The C&C is developed in Python using Eclipse with the Pydev-extension, including the following packages:

- Python 2.7.1
- SQLAlchemy 0.7.3
- SQLite3
- python-cloudfiles 1.7.9.1
- boto 2.1.0

- Django 1.3.1

The C&C consists of the following components:

- The Receiver, providing the API for different User Interfaces.
- The CompleteFile- and Chunk-objects which are mapped to the database.
- A splitter-class, providing the splitting and merging of the files.
- Interfaces to interact with the different storage providers.

4.1 The Receiver

The receiver provides an API for the user interfaces to interact with the C&C. The main functions are:

- showAllFiles – returns a list with all the CompleteFile-objects in the database.
- sendFile – takes a file as argument, splits the file and uploads the chunks.
- getFile – takes the filename as argument, initiates the download of all the chunks belonging to the CompleteFile.
- decider – analyses the argument and decides whether to upload or download, and checks if the file already exists in the database.

4.2 The Splitter

The splitter splits and merges a CompleteFile. The split process read a file, and splits it into a predefined number of chunks, which are stored temporarily on the disk. It returns a list of all these chunks. The merge process does the exact opposite, takes in all chunks of a CompleteFile, assembles them, and stores the result to disk. It then returns the path to the file.

The splitting process is a critical part of the system. To split the data we use a simple algorithm which loops through the files and samples every n-th byte into a new file (chunk) where n is the number of chunks. We store the chunks in temporary files in the C&C before we send them to the cloud storage providers.

4.3 The Storage Provider Interfaces

Most cloud storage providers offer a solution to interact with the platform in different programming languages. Currently we've implemented interfaces for Swift, the storage

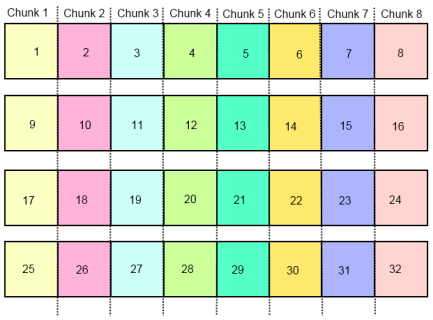

Figure 2: A file is split into 8 chunks. All parts with the same color goes in the same chunk.

solution in OpenStack, and Amazon S3. These implement the following functions:

- sendChunksInList
- getChunksInList

5 Process Description

The process of uploading a file is illustrated in Figure 3.

1. A user uploads a file.
2. The dashboard makes sure it is stored in the correct folder, and sends the file through to the C&C-receiver.
3. The C&C-decider first checks if the file exists in its database. If it doesn't, the file is handed to the sendFile-method.
4. In sendFile, all info of the file is serialized to a CompleteFile-object.
5. The desired splitting algorithm is used to split the file into the desired number of chunks.
6. The chunks are dedicated their cloud storage provider, and are uploaded in threads by each provider.
7. When all chunks are transferred successfully, the CompleteFile-object gets a signal, whereupon it saves all info about itself and its chunks in the database.
8. A message is sent to the dashboard and displayed to the user real time.

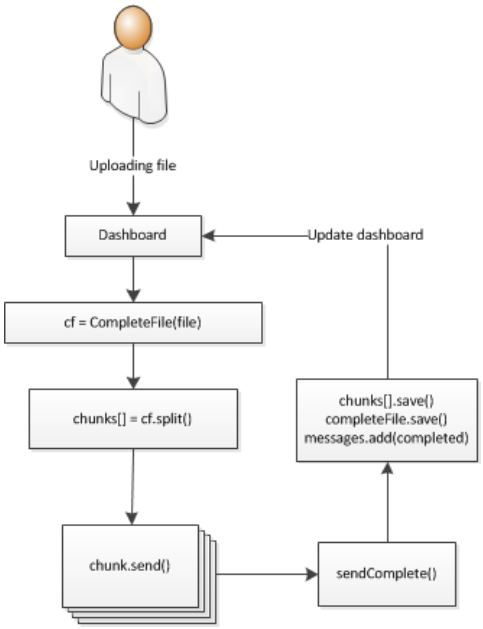

Figure 3: The process of uploading the file

6 Results

The C&C performs IO operations on the file when splitting and merging, and we therefore wanted to measure the IO overhead when uploading and downloading. The preferred protocol for data transfer is HTTP. We have performed measurements with different file sizes and varied the number of chunks to find the best conditions for using the C&C. Overhead is reported on top of 100%, i.e. if sending a complete file takes 100 seconds, and sending all chunks of a certain size takes 150 seconds, the overhead is 50%.

We first uploaded files with different sizes without splitting them, followed by downloading these files again. The uploading and downloading without splitting is the reference we measured the overhead against. We then repeated the procedure with split files. We started by splitting into chunks, then downloading and merging these chunks into a file. Then we proceeded by uploading the same files split into different numbers of chunks.

1. Upload all files in the set to the cloud providers, without splitting.
2. Download these files.
3. Repeat steps 1-2, with splitting and merging the files in a varying number of chunks.
4. Repeat steps 1-3 until the sample space is large enough.

Each upload and download was recorded. After the completion of the benchmark, the average upload and download duration and upload and download bitrate was calculated. The results are shown in Figure 4 and 5 below. The time taken to split and merge was also recorded. Since splitting is done asynchronously with uploading, the split time is not of much value. The merge time, however, is very important since we need to download all chunks before merging them.

Every value presented is a result of a set of minimum ten uploads or downloads, where each value is within 90% of the median value in the result set. When analyzing the data, the values seemed to be so consistent that the result set obtained was deemed large enough to draw conclusions.

6.1 Uploading Results

Figure 4 shows the duration of uploads in seconds. From the graphs we see that more chunks leads to faster uploads. Control measurements have been performed, with 30 and 50 chunks, without including these in the graphs, and they show that the tendency continues.

When looking at the overhead in percent, see table 1, we see that the overhead is off the charts when regarding the 7MB file. If we exclude that file, we notice that the overhead is almost consistent per the number of chunks. We conclude that the average overhead is a little above 140% when the number of chunks is between 12 and 20.

	4 chunks	8 chunks	12 chunks	16 chunks	20 chunks
7 MB	214%	237%	260%	289%	342%
62 MB	151%	148%	143%	145%	147%
175 MB	155%	151%	139%	144%	139%
323 MB	157%	154%	142%	141%	139%
486 MB	-	153%	147%	143%	141%
Average	169%	169%	166%	173%	182%
Average w/o 7MB	154%	152%	143%	143%	141%

Table 1: The overhead in percent when uploading files.

6.2 Downloading

Figure 5 shows the duration of downloads in seconds. From the graphs we see that optimum number of chunks is between 12 and 20 chunks, represented by 16 chunks here. From table 2 we see that the average overhead is 73% when using 16 chunks, so we will operate with this number as the overhead when downloading.

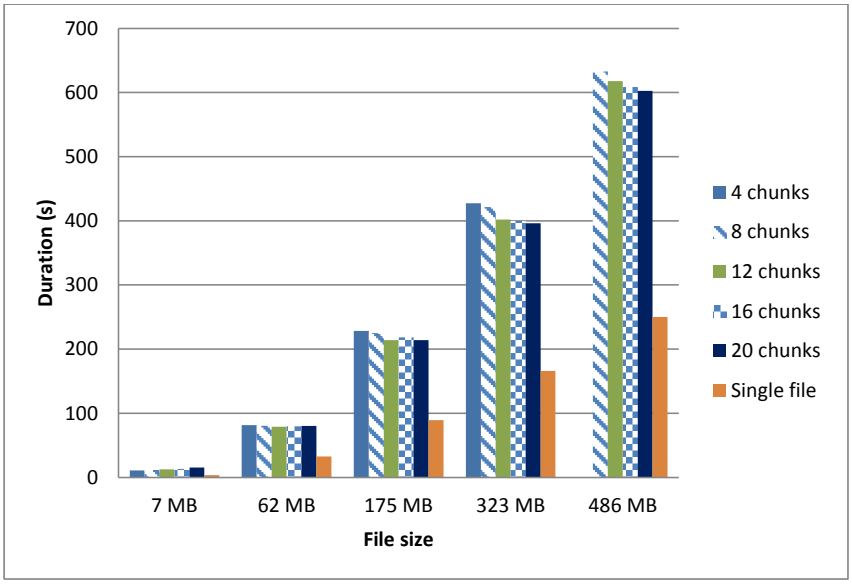

Figure 4: The duration of uploading files of different sizes and different number of chunks to the provider.

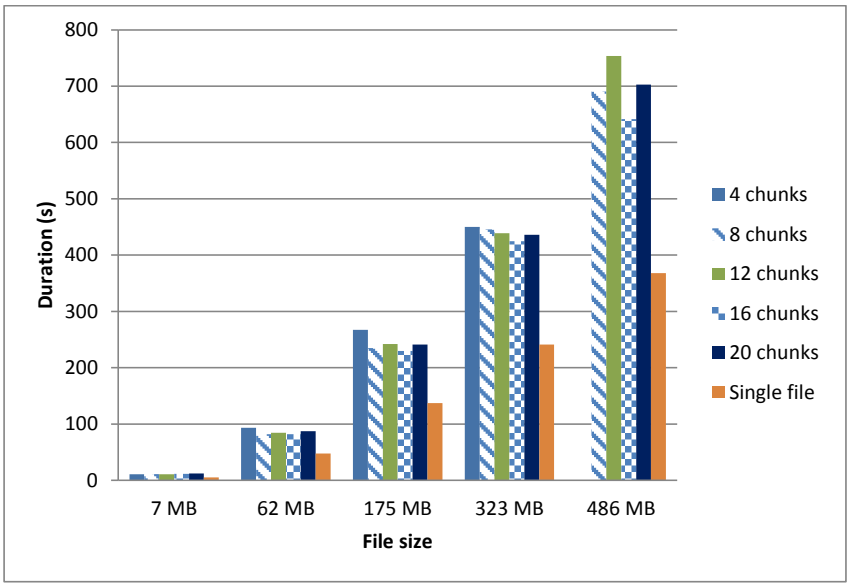

Figure 5: The duration from starting download to all chunks are merged.

	4 chunks	8 chunks	12 chunks	16 chunks	20 chunks
7 MB	103%	106%	102%	122%	130%
62 MB	96%	73%	78%	72%	84%
175 MB	95%	71%	76%	68%	76%
323 MB	87%	85%	82%	76%	81%
486 MB	-	88%	105%	75%	91%
Average	95%	84%	89%	83%	92%
Average w/o 7MB	93%	79%	85%	73%	83%

Table 2: The overhead in percent when downloading files.

7 Discussion

The solution we have presented here has a number of limitations. When we split up the data and send it to different cloud providers, we also lose the possibility to use the cloud for processing of the data. The data has to be merged together in the C&C before it can be processed. We don't have any redundancy in our system, which makes it vulnerable if one of the cloud providers loses a chunk. If the chunk can't be recovered, the data in the original file will be lost. The splitting algorithm is naïve, without any mathematical proofs. In real life, a more sophisticated splitting mechanism would be necessary to prevent providers from extracting interesting information from a single chunk.

In the following we will discuss some possible attacks on our solution.

7.1 Eavesdropping on the Network

If an attacker eavesdrops on the network connection out from the C&C it will be possible for him to collect all the data that are transferred. Collecting the chunks from one file can be done by assuming that chunks sent at approximately the same time, belong together. If the attacker does this, he still has to put the chunks together. If the attacker knows the splitting algorithm and have collected all the chunks from the file, he also needs to know the order of the chunks. If the number of chunks is low, this can be done easily with some trying and failing.

To make it a little harder for the attacker to reassemble the file, there are some countermeasures that can be used. First the C&C can randomize the number of chunks for each file. The randomization will make it harder for attackers to know the exact number of files to reassemble since they don't know how many chunks each file consists of.

Another method is to send file chunks in batches, i.e. let the C&C send chunks from two different files at the same time. If the attacker can't distinguish the chunks from each other it will be a lot harder to reassemble the original files.

A third method is to introduce garbage chunks that are transferred together with the real chunks. Even if the attacker knows that the data contains garbage chunks, he needs to sort

out *which* chunks are garbage in order to be able to obtain the original file.

7.2 Attacking the C&C

The C&C is definitely the most valuable component in the system from an attacker's point of view. If the C&C is exposed, either physically or digitally, the attacker will be able to access the data or perform a denial of service (DoS) attack.

In our system design we propose that the C&C should be within the company's own infrastructure. This means that they have to secure the C&C physically by making sure it is located in an locked room where only authorized people can access it. They also have to make sure that the machine is not remotely accessible through the network. This means that they have to have a proper firewall, authentication, updated software and disabled unused services.

7.3 Attacking the Cloud Provider

An attack on a single cloud provider will not compromise any data, since the chunks don't contain any information by themselves. An attacker has to attack all the cloud providers and find the chunks that belongs together, e.g. by looking at the file creation timestamps. This attack is subject to the same countermeasures as the eavesdropping attack mentioned above.

7.4 Alternative Approaches

Instead of storing unencrypted chunks, it is possible to generate a keystream, XORing the plaintext with this keystream, and then storing both ciphertext and keystream as chunks at disjunct storage providers. This will double the upload/download time and storage space required, and also add some time for generating the keystream for each storage operation. Alternatively, only the ciphertext could be stored, but then this degenerates into an encrypted cloud storage, and all the splitting etc. is pointless.

8 Further Work

Since this is little more than a proof-of-concept prototype, there are abundant opportunities for further work. One obvious area is the splitting algorithm, which needs to be more advanced if an attacker with access to even a few chunks is to be thwarted. One specific improvement may be to randomize the number of chunks for each file. The randomization will make it harder for the cloud provider or attacker to know the exact number of files to

reassemble since they don't know how many chunks each file consist of.

There is also currently no protection against data loss and corruption. One way to approach this might be to employ automatic duplication of cloud storage providers as suggested by Zhao et al. [ZRJS10]. Another solution could be to duplicate the Redundant Array of Independent[1] Disks (RAID) concept of striping blocks across providers, e.g. dividing content among 4 providers, and placing "parity chunks" on a fifth provider, thus compensating from the failure of any one of the other providers.

9 Conclusion

We have shown that implementing a simplified version of the RAIN design is feasible and working. We have also shown that our solution integrates well with open source cloud solutions such as OpenStack, as well as a commercial version like Amazon's S3. This is a great advantage in preventing vendor lock-in, and makes migration from one cloud provider to another easier. We have tested and measured the prototype and we have pointed out some improvements that can be applied when it comes to performance and security.

We believe that privacy concerns may be a major stumbling block for further development of internet communities, and our hope is that the RAIN concept may contribute to alleviating some of those privacy concerns in the future.

Acknowledgements

This paper is partly based on a minor thesis project at NTNU [AS11]. We thank the anonymous reviewers for useful comments.

References

[ALPW10] Hussam Abu-Libdeh, Lonnie Princehouse, and Hakim Weatherspoon. RACS: a case for cloud storage diversity. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 229–240, New York, NY, USA, 2010. ACM.

[AS11] Christian Askeland and Anders Emil Salvesen. Secure Use of Public Cloud Services. NTNU minor thesis (fall project), 2011. `http://sislab.no/drizzle`.

[BCD+09] Peter Bogetoft, Dan Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Nielsen, Jesper Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Secure Multiparty Computation Goes Live. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography and Data Security*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343. Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-03549-4_20.

[1] Some prefer the variant "Redundant Array of Inexpensive Disks".

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 169–178. ACM, 2009.

[GHR+10] Eric Grosse, John Howie, James Ransome, Jim Reavis, and Steve Schmidt. Cloud Computing Roundtable. *Security Privacy, IEEE*, 8(6):17 –23, nov.-dec. 2010.

[HR02] Steven Hand and Timothy Roscoe. Mnemosyne: Peer-to-Peer Steganographic Storage. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 130–140, London, UK, 2002. Springer-Verlag.

[JNAZ11] Martin Gilje Jaatun, Åsmund Ahlmann Nyre, Stian Alapnes, and Gansen Zhao. A Farewell to Trust: An Approach to Confidentiality Control in the Cloud. In *Proceedings of the 2nd International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless Vitae Chennai 2011)*, 2011.

[JZA11] Martin Gilje Jaatun, Gansen Zhao, and Stian Alapnes. A Cryptographic Protocol for Communication in a Redundant Array of Independent Net-storages. In *Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2011)*, 2011.

[LFMB08] Jesus Luna, Michail Flouris, Manolis Marazakis, and Angelos Bilas. Providing security to the Desktop Data Grid. In *Parallel and Distributed Processing 2008 IPDPS 2008 IEEE International Symposium on*, pages 1–8, 2008.

[PK09] Abhishek Parakh and Subhash Kak. Online data storage using implicit security. *Information Sciences*, 179(19):3323 – 3331, 2009.

[Rab89] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM (JACM)*, 36(2):335–348, 1989.

[RNJ12] Chunming Rong, Son Nguyen, and Martin Gilje Jaatun. "Beyond Lightning: A Survey on Security Challenges in Cloud Computing". *Computers & Electrical Engineering*, to appear 2012.

[SGR09] Nuno Santos, Krishna P. Gummadi, and Rodrigo Rodrigues. Towards Trusted Cloud Computing. In *HOTCLOUD*. USENIX, 2009.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22:612–613, November 1979.

[SKZ11] Yashaswi Singh, Farah Kandah, and Weiyi Zhang. A secured cost-effective multi-cloud storage in cloud computing. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 619 –624, april 2011.

[ZJV+11] Gansen Zhao, Martin Gilje Jaatun, Athanasios Vasilakos, Åsmund Ahlmann Nyre, Stian Alapnes, Qiang Ye, and Yong Tang. Deliverance from Trust through a Redundant Array of Independent Net-storages in Cloud Computing. In *Proceedings of IEEE Infocom*, 2011.

[ZRJS10] Gansen Zhao, Chunming Rong, Martin Gilje Jaatun, and Frode Sandnes. Reference deployment models for eliminating user concerns on cloud security. *The Journal of Supercomputing*, pages 1–16, 2010. 10.1007/s11227-010-0460-9.