Secure Cloud Storage Sharing Through Proxy Re-Encryption

Morten Salte (Author)
Department of Electrical and Computer Engineering
University of Stavanger
Stavanger, Norway
msalte86@gmail.com

***Abstract*—This paper proposes a scheme in which it is possible to provide access control on files stored in the cloud in a secure manner. Typically, in cloud environments, when end users want to share their files with other users, they must do so at the cost of privacy and security.**

Sacrifices like this is far from optimal and there are many examples of subsequent issues caused by such a deficiency. For instance, there have been several incidents featured in the media in the recent years where private files have ended up in the wrong hands.

Sharing files in the cloud in a secure manner is perfectly possible, but often includes a tedious process. The scheme proposed in this paper introduces an algorithm/procedure that allows secure sharing in a relatively straightforward approach by using so-called proxy re-encryption while also taking advantage of the BitTorrent protocols strengths.

***Index Terms*—cryptography; security; sharing; cloud storage; proxy re-encryption; BitTorrent**

I. Introduction

The scheme proposed in this paper uses an encryption scheme called proxy re-encryption in addition to taking advantage of peer-to-peer functionality present in the popular BitTorrent protocol. In practice, the proposed scheme will let file owners encrypt the location of their files with their own private encryption keys, before uploading the location to a proxy instance application. The files' location is in this case their respective static metadata files in accordance to the BitTorrent protocol, commonly known as torrent files.

When the file owners wants to share a file with someone, they do so through the proxy instance application, which is capable of re-encrypting the torrent files, enabling decryption capability by the recipient. The proxy instance application will not learn anything about the file's contents during this process and neither the proxy instance nor the recipient will learn anything about the file owner's private encryption key. When the recipient downloads the encrypted torrent file from the proxy instance application, it is able to decrypt the file with its own private decryption key.

II. Background

The following two sub sections discusses the background of the two most central elements in the proposed scheme, namely proxy re-encryption and the BitTorrent protocol.

A. A Proxy Re-Encryption Scheme

Proxy re-encryption is a technique for re-encrypting an already encrypted message (i.e. ciphertext) so that another secret key can decrypt it [5]. This technique is very useful in cases where a primary party wishes to let a secondary party read an encrypted message without revealing the secret decryption key. A proxy re-encryption scheme has many similarities to both symmetric- as well as asymmetric encryption schemes.

To explain the significance and strength of such a scheme, the following scenario is often used [5] [6] [2]. Let us assume that Alice and Bob works in the same company and that Bob is Alice's subordinate. Many people send e-mails to Alice containing important company information. When they do this, they of course encrypt the e-mails with Alice's public key so that only she can decrypt- and read them.

However, if Alice is out sick or away on vacation, she may allow Bob access to her e-mails during this period. To achieve such functionality without Alice having to reveal her private key, proxy re-encryption is a perfect fit. All Alice has to do is to supply Bob with a re-encryption key generated using her own private key and Bob's public key. Bob can then re-encrypt Alice's incoming e-mails so that he can decrypt them with his own private key.

1) Design: There are several possible approaches to designing a proxy re-encryption scheme for use in the aforementioned e-mail scenario. In theory, it is possible for Alice simply to store her private key on the e-mail server. This way the e-mail server will be able to decrypt Alice's incoming e-mails and immediately encrypt them using Bob's public key. There is a big problem with this approach though; there must be a completely trusted party present, i.e. the e-mail server in this case. Relying on trusted parties is never a good idea in cryptography, so this approach is naïve and consequently relatively useless [1].

Another possibility would be for Alice to remain online at all times, and manually decrypt all of her incoming e-mails with her private decryption key before encrypting them with Bob's public encryption key. Subsequently, Alice would forward the newly encrypted e-mail to Bob. The problem with this approach is obviously the fact that it is very impractical for Alice to remain online and perform the decryption/encryption task.

Blaze, Bleumer and Strauss (BSS) introduced the concept of proxy re-encryption in 1998 when they published their paper about their scheme Divertible Protocols and Atomic Proxy Cryptography [2]. Their scheme is perfectly capable of providing the desired re-encryption functionality. However, their scheme contains some weaknesses, specifically in regards to delegation transitivity and collusion [5].

Firstly, the scheme's delegation function is bidirectional, meaning that a single re-encryption key generated for re-encrypting messages intended for Alice so that Bob can decrypt them, also allows re-encrypting messages intended for Bob so that Alice can decrypt them. In other words, if Alice wants to let Bob be able to decrypt her incoming e-mails, Bob must agree to let Alice be able to decrypt his incoming e-mails as well.

Secondly, if the proxy instance is fraudulent, a collusion issue lets it cooperate with either Alice or Bob to learn the other party's secret key.

The scheme proposed in this paper does not discriminate between proxy re-encryption implementations. However, it is obvious that an implementation providing a unidirectional delegation function while also solving the collusion problem is preferred. There are many such implementations available, but their respective details are not within the scope of this paper. The proposed scheme is only concerned with the existence and capability of a proxy re-encryption scheme, and intend to take advantage it.

B. The BitTorrent protocol

The BitTorrent protocol is a widely used protocol in file sharing systems over the Internet today. Its conservative use of bandwidth and hardware resources is arguably one of its best properties. In traditional file sharing systems, one would establish a client/server architecture by placing a host machine with exceptional hardware specs on a location with state of the art Internet connectivity. This host machine would contain all the files of interest to its clients, and users would need to download the files directly from the host over the FTP or HTTP protocol. This approach would apply a huge traffic load to both the host's network infrastructure as well as its hardware. If many enough users were downloading simultaneously, traffic would consequently end up being slow or even cause the host machine or network to kneel. It is reasonable to argue that regardless of how much money one invests into a single host machine approach; its performance and reliability will likely never be able to compete with a peer-to-peer approach such as in the BitTorrent protocol.

In the BitTorrent protocol, the idea of a single host machine containing all the files is abandoned. The BitTorrent protocol uses a peer-to-peer approach that allows all users interested in a particular file to act as equals in distributing the file among themselves over multiple TCP connections. This takes away the load from a single host and spreads it among an arbitrary number of users instead.

To achieve this behavior, a file in the BitTorrent protocol is split into an array of pieces. A single piece is typically 256KB

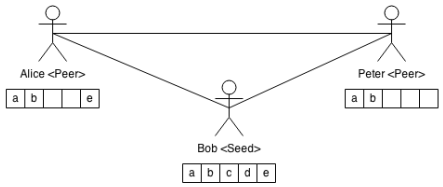

Fig. 1. Communication Between Seeds and Peers

large, and the protocol enables piece by piece downloading. This means that users can download parts of a file in a single session, and then come back later to download more parts in a later session. The downloading process lets users download a file's pieces from any of the other users participating in a file's distribution, given that they are in possession of them.

Initially, when adding a new file to the system, a file owner/source must necessarily have the complete array containing all of the file's pieces at its disposal. The term *seed* is used to classify someone in possession of a file's entire array of pieces. Initially, the file owner is the only seed present, but throughout the file distribution process, other users who obtain the complete file become seeds as well. Other participating users in possession of few or no pieces are simply *peers*.

Figure 1 is a simplified illustration showing the distribution of a file in the BitTorrent protocol. The lines between the participating users represents TCP connections. As we can see, Bob is the seed and is in possession of the entire file. Peter and Alice are peers and are only in possession of some of the file so far in the process. In reality, the figure is missing an essential part of the BitTorrent protocol, namely the tracker.

The tracker is an application running on a web server and is responsible for coordinating the communication between the seeds and the peers. The main role of the tracker is to allow peers and seeds to find each other. Participants connect to web server hosting the tracker to join in a file's distribution [8]. Consequently, the tracker contains a list of all the peers in the system, state information per peer and knows which peer has which of the file's pieces. If we look back at Figure 1 illustrating the communication between the seeds and peers in a file distribution, we would need to add two links between each of the participants and the tracker to complete the illustration; one TCP and one HTTP link. In Figure 2, this modification is present and shows a more complete illustration of how a file distribution works in the BitTorrent protocol. The dotted line between a participant and the tracker represents a HTTP connection, while the regular line remains a TCP connection as before.

To find information about a file's tracker, one would need to look in the file's corresponding .torrent file. The .torrent file is a static file containing metadata about the file it is representing. It is a dictionary of keys and values encoded in the Bencode format [8]. It contains the URL of the tracker, a hash value for every piece of the file (for validation purposes), piece length, file name and file length/size. The .torrent file is an essential part of the downloading process, as it is required for new users

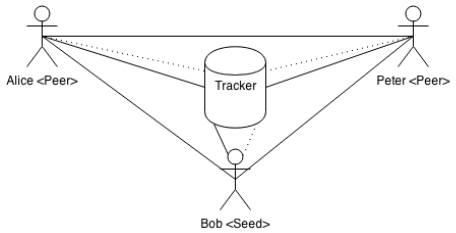

Fig. 2. The BitTorrent Tracker's Communication With Seeds and Peers

```
announce: 'http://some.tracker.url?token=abc'
info {
    name: 'my_music'
    length: 2131515
    piece length: 256000
    pieces: abb213b  231jjeau  23kkejai
    path: { 'song1.mp3', 'song2.mp3' }
}
```

Fig. 3. A Simplified Torrent File

to join in the distribution of a file.

The Bencode format is a data language that serves for storing data in a platform independent way. It has many similarities with more familiar formats such as JSON and XML.

All strings in the Bencode format is length prefixed, meaning that to store the string `hello`, it would need to be stored as `5:hello`, indicating that it is five characters long. To store integers, one must prepend a prefix `i` as well as appending a suffix `e`. To store the number 6, it would need to be stored as `i6e`.

Additionally, to encode a list it would look like `l<contents>e` while a dictionary's encoding would look like `d<contents>e`. The contents of the respective data structures must be bencoded as well. The contents of a list must appear in order. The contents of a dictionary must appear as alternations between keys and their corresponding value. Table I shows some examples of bencoding.

TABLE I
BENCODING EXAMPLES

Data	Type	Bencode
hello	String	5:hello
6	Integer	i6e
{”one”, 2, ”three”}	List	l3:onei2e5:threee
{”year”, 2014}	Dictionary	d4:yeari2014ee

Figure 3 shows an example of a simplified .torrent file. Note that it is not bencoded, for simplicity's sake.

We can see that the announce field points to the tracker URL and that the info field contains a dictionary of further information about the file. The file name is set to ”my_music” indicating that this is a file containing music files. The length of the file is 2131515 bytes and has a piece length of 256KB. The pieces field indicates that the file consists of three pieces, all of which is stored as hash values. Finally, the path field contains a list specifying the file's subdirectory names, or in this case the names of the individual mp3 files. Again, please note that Figure 3 is a simplified illustration of a .torrent file, and is not accurate in any way.

1) The Process of Downloading a File: The following describes the process of downloading a file using the BitTorrent protocol.

1) Obtain the .torrent file containing the metadata about the desired file and open it in any BitTorrent client.
2) The BitTorrent client will examine the .torrent file and locate the tracker to which it will make a connection attempt.
3) If the connection was successful, the tracker will tell the BitTorrent client about which peers and seeders are currently actively participating in the distribution of the file, to which the client will subsequently make connections.
4) The client will begin downloading the pieces of the file from its connections.
5) Eventually the client will have downloaded all the pieces and the entire file is at the user's disposal. At this time, the client can safely be closed, or remain open to continue contributing in the distribution of the file.

III. SCHEME AND ALGORITHM

Up until now, most of the paper has discussed the details of the proxy re-encryption scheme as well as the BitTorrent protocol. The following will take a dive into the combination of these technologies to end up with the proposed scheme capable of providing access control on files that are stored in cloud environments.

A. General Idea, Purpose and Goal

As was touched on in the introduction, the scheme proposed in this paper tries to provide access control on files that are stored in cloud environments. This lets users share their files stored in the cloud with other users, without sacrificing security in the process. The first couple of months of this project make up a survey period. During this period, an exploration of what exists in regards to secure cloud sharing took place. It quickly became clear that most of the popular cloud storage services today are in fact lacking when it comes to sharing. They all appear to invest heavily in making sure that their users' files are stored in a secure manner, but they also openly admit that they are in fact capable of decrypting their users' files. This is because they typically use a symmetric encryption scheme for encrypting their users' files, and they themselves are the key holders. This means they have complete control over their users' files. Additionally, many cloud storage services admit that they will grant access to their users' files to government officials, if they are required to do so by law.

The idea behind this paper's proposed scheme is not to come up with some new encryption scheme, but rather to come up with a procedure taking advantage of already existing technologies that will let users achieve better security when sharing their files in the cloud.

The idea of using the BitTorrent protocol simply origins from the fact that it will let users share files in an efficient way. When sharing files, letting other people access them directly in the cloud causes security issues right off the bat. Services that only offer server-side encryption and are consequently in possession of the decryption keys will simply decrypt the file owner's file and let the recipient access it as well. Services that claim to offer client-side encryption appear to have their own security issues as well. While the security appears to remain intact as long as the data is not shared with other users, issues surface once the sharing takes place [4].

To try to compensate for the deficiencies present in a more traditional cloud sharing approach, the proposed scheme will continue to allow the file owner to store his files in his private cloud storage. However, when sharing a file, he must grant the receiver access through the BitTorrent protocol's file distribution process. The proposed scheme suggests to share files' respective torrent file among the participants. The file owner initially encrypts the torrent file, and a proxy instance may grant others access to said torrent file through proxy re-encryption.

When the file owner shares the file, it does so by letting a proxy instance re-encrypt the file so that the recipient can decrypt it with its own private key. This will allow the file owner to store his file in an arbitrary manner. When the sharing takes place, the file owner generates a re-encryption key by using his private key and the recipient's public key. After the file owner supplies a proxy instance with the re-encryption key, the recipient is in practice granted access to the file. All the recipient must do to complete the process is to let the proxy instance re-encrypt the file. After re-encryption, the recipient itself is capable of decrypting the file with its private key.

Because the files shared in the proposed scheme are .torrent files, the recipient can join the torrent swarm after decryption. At this time, the recipient learns the BitTorrent tracker's location, and eventually obtains the complete file.

B. Algorithm

The following sub sections will describe the entire algorithm from start to finish, i.e. from sharing the file with someone to the receiver obtaining the file. The algorithm can be divided into three stages; *upload*, *share* and *download*. In addition to these three stages, the algorithm relies on some prerequisites.

Again, note that the scheme does not specify which proxy re-encryption scheme to use. However, as mentioned previously in the paper, some conditions should preferably be satisfied, namely the unidirectional delegation function as well as an avoidance of the collusion issue.

1) Prerequisites: Since the proposed scheme is relying on the BitTorrent protocol, there must exist web server hosting a BitTorrent tracker for the scheme to work. The location of the BitTorrent tracker is arbitrary, and the scheme's participants need not know anything about it. The web server hosting the tracker should preferably contain a mechanism excluding incoming connections that are not part of the proposed scheme. More on this later.

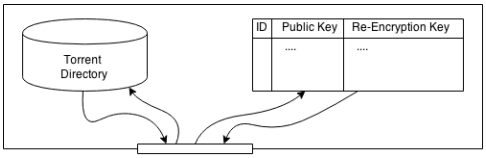

Fig. 4. The Proxy Instance Application

Additionally, the scheme's proxy instance is an application in itself. This means that users must access an application representing the proxy instance to participate in the scheme. In other words, when the file owner wants to share one of his files, he must do so through the proxy instance application. Similarly, this lets the recipient access the file by inquiring the proxy instance application.

The proxy instance application's interface consists of some very basic functions. As illustrated in Figure 4, the proxy instance application primarily consists of a torrent directory and an access control table of file ids, recipient's public keys and corresponding re-encryption keys. The torrent directory contains all the .torrent files that are part of the scheme at any given time. The access control table contains entries for all shared torrents and the corresponding public- and re-encryption keys.

When a file owner uploads .torrent files to the proxy instance, they will be stored in the torrent directory. When a file owner shares a file with someone, the proxy instance makes a corresponding entry in the access control table.

The pseudo code in Listing 1 intends to illustrate the proxy instance application's interface in a simplified manner. With this in mind, by looking at the pseudo code it still becomes apparent that its purpose makes a good fit for a web service or RMI oriented application. However, this paper does not contain any specific specifications on this topic.

Listing 1. The Proxy Instance API

```
// Upload a File to the Proxy Instance Application
int upload(file);

// Share a File With Given Public Key Holder
void share(file_id, re-encryption_key, public_key);

// Download the File Shared With Given Public Key
Object download(file_id, public_key);
```

As we can see, the *upload* function lets file owners upload their initial files to the proxy instance. The file argument supplied to this function should be a .torrent file. Similarly, the *share* function lets file owners share a file with someone. The arguments supplied to this function must represent the file's id, a re-encryption key and a public key, respectively. Finally, the *download* function lets a recipient download a torrent file. In it, by supplying its public key as an argument, the recipient identifies itself to the proxy instance. The proxy instance looks at the access control table entry for the given file id and if the given public key is found, lets the recipient download the re-encrypted torrent.

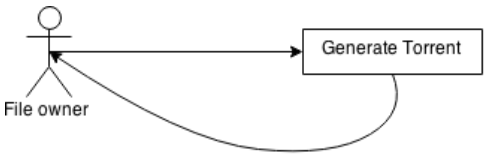

Fig. 5. [UPLOAD] File Owner Torrent Generation

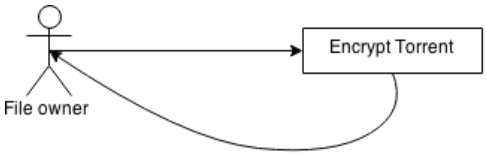

Fig. 6. [UPLOAD] File Owner Torrent Encryption

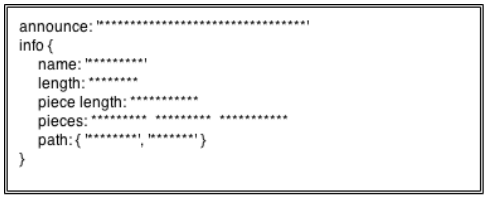

Fig. 7. [UPLOAD] Encrypted Torrent File

2) Upload: Initiating the upload stage, the file owner generates a .torrent file corresponding to the file he wants to share. As discussed previously in this paper, the .torrent file contains a lot of metadata about a file. The URL of the BitTorrent tracker is arguably one of the most important properties of the torrent file. As such, the torrent file generation process taking place at this stage must specify a tracker URL corresponding to the tracker that was set up in the prerequisites chapter. The tracker URL should have a type of security token appended to it. The web server that is hosting the tracker can intercept this token and determine whether the incoming connection request is valid, i.e. part of the proposed scheme.

From Figure 5 we see an illustration of the file owner generating the torrent file. Figure 3 shows an example output file. Note that the tracker URL in the announce field is appended by a security token, in this case simply ”abc”.

After the .torrent file generation, the file owner encrypts the file as illustrated in Figure 6. Figure 7 shows the result of the encrypted torrent file. In practice, the encryption will obviously entail the entire torrent file, not just the keys' values such as in the illustration.

After encrypting the torrent file, the file owner uploads it to the proxy instance application as shown in Figure 8. As a result, the proxy instance application returns the newly uploaded file's id to the file owner. The file owner now knows the id of the uploaded file, which the proxy instance application generated. The file owner must supply this id to the share function in the next stage of the algorithm.

After uploading the file, the upload stage concludes. The next stage is the sharing stage.

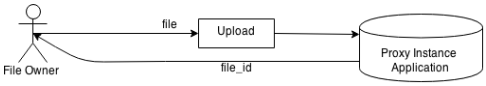

Fig. 8. [UPLOAD] Torrent File Upload

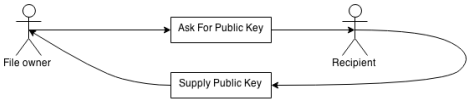

Fig. 9. [SHARE] Public Key Inquiry

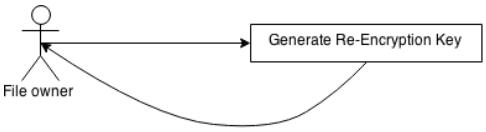

Fig. 10. [SHARE] Re-Encryption Key Generation

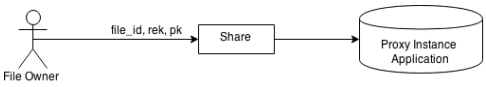

Fig. 11. [SHARE] Share File

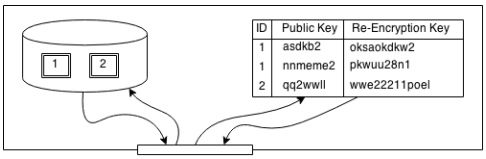

Fig. 12. Example Proxy Instance Application State

3) Share: When the file owner wants to share access to a torrent file residing in the proxy instance application's torrent directory, he initiates the process by inquiring the recipient about its public encryption key. After receiving the recipient's public key, the file owner can continue to generate a re-encryption key. Figure 9 shows a simple illustration of the key inquiry. After the file owner has the recipient's public key in his possession, he generates the re-encryption key as illustrated in Figure 10.

To complete the sharing stage, the file owner informs the proxy instance application about the sharing by initiating the share function. Figure 11 shows a simple illustration of this.

As we can see, the file owner supplies the share function with three arguments, namely the id of the torrent file, the generated re-encryption key and the recipient's public encryption key. Followed by this function call, the proxy instance application makes a corresponding entry in its access control table.

Figure 12 shows an example of the proxy instance application's state when two torrent files are present in its torrent directory. We see that the file with id 1 is shared with two different users while the file with id 2 is shared with a single user.

As a side note, note that it would be entirely possible for the

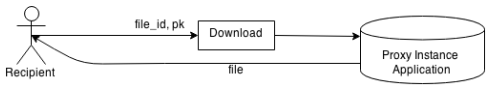

Fig. 13. [DOWNLOAD] Download File

Fig. 14. [DOWNLOAD] Token Validation

file owner to re-encrypt the respective torrent file himself and then to supply it directly to the recipient, without involving a proxy instance. However, this would require the file owner to remain online during the entire sharing process. By introducing a proxy instance application, the file owner supplies a third party instance with the re-encryption key, and the recipient can inquire that instance about re-encryption at an arbitrary time. In other words, one of the primary reasons for the proxy instance application to exist in this scheme is to remove the necessity for the file owner to remain online when the recipient wants to access a shared file.

To conclude and summarize the sharing stage, at this point the encrypted torrent is still residing in the proxy instance application's torrent directory, which is now also in possession of a re-encryption key. The final download stage can now begin, in which the recipient will end up downloading the torrent file.

4) Download: When the recipient is ready to download a file that the file owner shared with it, it initiates the proxy instance application's download function. Figure 13 illustrates this.

As we can see, the recipient supplies the download function with two arguments, namely the id of the file it wants to download as well as its public encryption key. By examining the supplied public encryption key, the proxy instance application verifies whether the inquiring entity has access to the given file id. If the file exists in the access control table with a corresponding public encryption key matching the supplied argument, the proxy instance application re-encrypts the file with the matching re-encryption key before returning the re-encrypted file to the inquiring user. If there is no match for the supplied public key in the access control table, the proxy instance application ignores the inquiry. Access denied.

If access is granted and the inquiring user, i.e. the recipient receives the re-encrypted torrent, all it has to do is to decrypt it with its own private decryption key. Subsequently, upon accessing the file in clear text, the recipient can connect to the tracker and join the torrent swarm, eventually obtaining the file. Figure 14 illustrates this.

C. Further Details

An obvious challenge with the proposed scheme is that there is no way for the file owner to deprive a recipient the ability to share the file with its friends. However, this is something that is common for most if not all security schemes out there. As a workaround for this weakness, it is common to inscribe a type of watermark in the files that it will be able to track back to the original recipient.

For instance, if Bob the file owner were to apply a watermark unique to Alice the recipient on the file before sharing it with her, Bob would be able to reduce Alice's temptation to share the file by informing her that he will be able to track the betrayal back to her. If Bob tells Alice that he will be able to prove it if Alice shares the file with someone else, the probability that Alice will do so declines significantly. However, it is still true that for Bob to be able to prove that Alice shared the file, he must first stumble upon the file containing Alice's unique watermark somewhere else. If Alice and her friends are able to keep the file private, Bob will never know.

Another obvious weakness with the proposed scheme is that there is no way for Bob to revoke access to the file. After Alice has the file at her disposal, Bob will not be able to change his mind about Alice's access to it.

While the proposed scheme does not remedy these challenges, they are obvious candidates for further work.

IV. Conclusion

First off, the proposed scheme is so far only an idea and as such primarily works on paper. It still needs to be implemented, almost certainly resulting in the surfacing of even more challenges than is discussed in this paper.

The proposed scheme introduces the so-called proxy instance application. This application will need to be introduced into current systems to be able to take advantage of the proposed scheme. While it would be nice to achieve the same capabilities without introducing a new application instance, that was not achieved here. It would undoubtedly be more attractive if a scheme with the same capabilities as in the resulting proposed scheme could be implemented in an already existing system without introducing customized application instances.

Even though the aforementioned was a slight letdown, the proposed scheme is still capable of providing secure sharing in the cloud, which was the goal and purpose of the project. To that end, the resulting proposal is satisfying.

Acknowledgment

I would like to thank Professor Chunming Rong at the University of Stavanger. His outstanding enthusiasm about the project and his invaluable feedback throughout the semester proved essential in the resulting paper.

References

[1] Z. Scott, "Special topics in theoretical cryptography, lecture 17: Re-encryption," 2007.
[2] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," 1998.
[3] E. M. van Eenennaam, "Performance factors in peertopeer file distribution networks," 2007.
[4] D. C. Wilson and G. Ateniese, "To share or not to share in client-side encrypted clouds," 2014.
[5] S. S. Chow, J. Weng, Y. Yang, and R. H. Deng, "Efficient unidirectional proxy re-encryption," 2009.

[6] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, “Improved proxy re-encryption schemes with applications to secure distributed storage,” 2005.
[7] Joe, “The bittorrent protocol,” April 2008. [Online]. Available: http://www.morehawes.co.uk/the-bittorrent-protocol
[8] B. Cohen, “The bittorrent protocol specification,” January 2009. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html