A Private Walk in the Clouds: Using End-to-End Encryption between Cloud Applications in a Personal Domain

Youngbae Song[1], Hyoungshick Kim[1], and Aziz Mohaisen[2]

[1] Department of Computer Science and Engineering,
Sungkyunkwan University, Republic of Korea
{youngbae, hyoung}@skku.edu
[2] Verisign Labs, USA
amohaisen@verisign.com

Abstract. This paper presents Encrypted Cloud (EnCloud), a system designed for providing end-to-end encryption between cloud applications to facilitate their operation and enable users trust in providers. EnCloud relieves end-users' privacy concerns about the data stored in cloud services so that the private data are securely stored on the cloud server in an encrypted form while the data owner's EnCloud applications are only allowed to decrypt the encrypted data. To show the feasibility of EnCloud, we implemented a prototype for Dropbox. The experimental results of the prototype demonstrate that the additional time delay incurred by EnCloud operations is acceptable (within 11.5% of the total execution-time).

Keywords: Cloud; Domain Management; Privacy; End-to-End Encryption

1 Introduction

Cloud computing services offer many benefits (e.g., data storage and computing infrastructure). However, they also raise serious privacy concerns [14]. These concerns are not only limited to the prying eyes of providers but also include government programs violating their citizens' basic rights.

The Snowden's leaks exposed that US and UK government agencies have collected online users' activities from cloud service providers. Reportedly, these agencies can directly access data on central servers of several companies (e.g., Microsoft, Apple, Facebook, Yahoo, Google, PalTalk, and AOL) for their surveillance efforts [6]. Also, providers like Google regularly get requests from governments and courts around the world to hand over users' data. In 2012, Google received 21,389 requests for information affecting 33,634 user accounts, where Google provided at least some data in response (about 66% of the time) [11]. Even worse, such cooperation is legal – the US Patriot Act, which was designed to give the US government access to information that may help prevent terrorist

attacks, provides the legal platform for US law enforcement agencies to access corporate and users data when necessary. To that end, users are starting to distrust cloud providers, and many users would prefer to store their data on their own devices at home, when possible [5, 7].

To contain such a powerful adversary, we propose *EnCloud*, a new security application that prevents an attacker from accessing cloud-based data without the owner's knowledge or consent. EnCloud is designed to provide end-to-end encryption between multiple cloud applications in the data owner's personal domain. The proposed system is quite different from existing commercial products in the key management (cf. §6): All encryption and decryption keys in EnCloud are located at the client side rather than the server side. From a privacy perspective, users can then truly control the use of keys and manage their personal data. To this end, our key contributions can be summarized as follows:

- We introduce EnCloud, a framework to address end-users' privacy concerns in cloud storage settings. We propose a secure domain management framework so that the user's data can only be accessed by cloud applications registered for her personal domain (cf. §3).
- We show that EnCloud can achieve data confidentiality against powerful adversaries who can access not only the data stored in the cloud storage but also any network communication at home by analyzing the security properties of the EnCloud system (cf. §4).
- We demonstrate the deployability of EnCloud by implementing a prototype to support end-to-end encryption between cloud applications for Dropbox. With this prototype, we analyze the overhead of EnCloud and demonstrate that the additional time overhead incurred by the encryption and decryption operations is rather marginal (within 11.5% of the total execution-time) compared with the overall execution-time (cf. §5).

The rest of this paper is organized as follows. In §2 we introduce the threat model. In §3 we outline the design of EnCloud. In §4 we discuss a security analysis of EnCloud. A prototype implementation and results are introduced in §5. The related work is reviewed in §6, followed by concluding remarks in §7.

2 Threat Model

We consider a powerful adversary who acts as a government agency. The adversary can access the data stored in the cloud storage and can monitor the traffic between the end-user and cloud provider. We assume a computationally bounded adversary running in a polynomial time, and is not capable of breaking the encryption algorithm without knowing the key(s) used for encryption; this

assumption is reasonable since breaking advanced encryption algorithms (e.g., AES [3]) is computationally infeasible for the most powerful supercomputers.

We assume end-hosts are trusted and not under the control of the adversary. However, the adversary is able to guess a user-chosen password using a low-cost offline password attack – in offline password attacks, an adversary holds any password-related messages or data, and then iteratively guesses the user's password and verifies whether his guess is correct or not in an offline manner.

Our goal is to protect the user's private data stored in the cloud so that the adversary only knows the presence of the encrypted data and their characteristics (e.g., creation time, size, etc.) but not the contents or intended use.

3 EnCloud System

To achieve the security goal described in §2, a user U's data should always be stored on the cloud server in an encrypted form. If U uses a single machine, end-to-end encryption is simple enough to be implemented – the encryption and decryption keys can easily be managed by an application on that machine.

However, nowadays a cloud service is not anymore accessed by a client application on one single machine. Users can use the cloud service for their PCs, Smartphones, tablets, smart TVs or any device equipped with the client applications. Thus a major challenge is how to share keys securely between these devices. EnCloud is designed to do this by creating a personal domain of authorized applications (or devices) that can decrypt the data on the cloud storage, in turn, ensuring that no unauthorized application can decrypt the data.

EnCloud has two functional components: Domain Manager (DM) and Domain Client (DC). To use a cloud server S in a private manner, the user U installs a DM application on one of her devices (e.g., PC or smartphone) – in principle DM should always be available for the communication with DC's; U also installs a DC application on her other devices to create her personal domain. We briefly explain the roles of DM and DC as follows (shown in Figure 1):

– **Domain Manager (**DM**)** is an application which is responsible for managing its domain members (i.e., Domain Clients) by registering and revoking them. The user U has to install this application on a network-enabled device (e.g., PC or Smartphone). This device would have processing, storage and display capabilities. The DM application creates a domain key and distributes the key to the domain clients in a secure manner. Here we assume that DM should always be available for the communication with its domain members although DC's can often be turned off and turned back on.

Table 1. The notations used in the EnCloud system.

Notation	Description	Notation	Description
U	User of EnCloud	S	Cloud Server
DM	Domain Manager	DC_i	Domain Client with the name of i
d	Data being stored in the cloud	k	Domain key
dek	Data encryption key	id_d	Unique identifier of data d
c_i	PIN code for DC_i	sk_i	Session key for DC_i
puk_i	Public key for DC_i	prk_i	Private key for DC_i

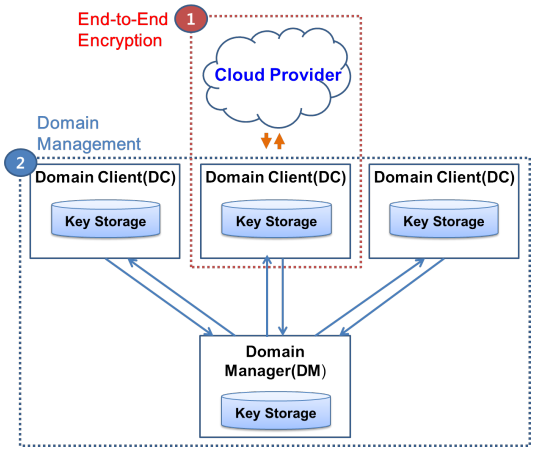

Fig. 1. The proposed EnCloud framework. Domain Manager (DM) is responsible for managing its domain members (i.e., Domain Clients) by registering and revoking them; a Domain Client (DC) can securely access the data created by one of DC's in the same domain.

– **Domain Client** (DC) is an application which is responsible for encrypting user data when the data are exported from originating domain and decrypting the encrypted data when encrypted data are imported into the domain. An existing cloud application interacts with DC to encrypt user data before uploading the data to the cloud server directly. A user U has to install the DC application on all the user's devices which will use the cloud server S.

With these two components, EnCloud securely protects U's private data within her personal domain devices through (1) end-to-end encryption and (2) domain management. In the following subsections, we will present how EnCloud works for protecting user data on the cloud server S. The notations used in outlining the operation of EnCloud are summarized in Table 1.

3.1 End-to-End Encryption

In EnCloud, encryption and decryption operations are processed at each DC application for end-to-end encryption between cloud applications. When up-

loading the data to the cloud server S, the data should be encrypted while the encrypted data should be decrypted only after downloading the data so that the user's data can be accessed by authorized applications only in her personal domain. Here we assume that each DC application holds the domain key k. In §3.2, we will discuss how to manage k for DC applications.

Uploading Data. When a user U uploads her personal data d from a cloud application to the cloud server S, the cloud application asks for the encryption of the data d by interacting with DC_i. DC_i creates a *data encryption key* dek to encrypt the data d. After encrypting the data d with dek, dek is also encrypted to be securely stored in the cloud storage. A unique identification string id_d is used to associate dek with d. For example, the *file name* of d can be used to implement id_d. After creating these objects, DC_i returns them to the cloud application and then the cloud application sequentially uploads them instead of the original data d. This process can be represented in the following manner:

$$U \longrightarrow S : (E_{dek}(d), id_d)$$
$$U \longrightarrow S : (E_k(dek), id_d)$$

Downloading Data. When a user U wishes to download the data d from the cloud server S via its associated cloud application, the cloud application returns $(E_{dek}(d),\ id_d)$ and $(E_k(dek), id_d)$. The identifier id_d is used as an index to obtain them. This process can be represented in the following manner:

$$S \longrightarrow U : (E_{dek}(d), id_d)$$
$$S \longrightarrow U : (E_k(dek), id_d)$$

DC_i decrypts $E_k(dek)$ with the key k then decrypts $E_{dek}(d)$ with dek. After decrypting both objects, DC_i returns the plain data d to the cloud application.

3.2 Domain Management

When a user U uses several applications and devices for the cloud server S, it is necessary to securely share keys between them for an application to freely exchange the encrypted data with other applications installed on other devices.

Creating Domain. After choosing a device which is proper for DM, U installs the DM application on the device. The DM application runs with default configuration settings and generates a random key k as domain key. The domain key k is securely stored.

Registering Device. When U registers a device with the name of i into her domain, U installs the DC application on the device. The DC application (i.e., DC_i) then searches DM via a network interface (e.g., WiFi). When the proper DM application is found, U requests the DM application to join DC_i into the domain managed by the DM application by sending the `JOIN` request message. This process can be represented in the following manner:

$$\text{DC}_i \longrightarrow \text{DM} : \texttt{JOIN}, \text{DC}_i, \text{DM}$$

When DM receives the `JOIN` request message, the unique identifier i for DC_i and a randomly generated PIN code c_i are displayed on DM. The displayed information is used to prevent man-in-the-middle attacks. U has to input the code c_i on DC_i for establishing a secure communication channel between DM and DC_i. When U successfully types the code c_i on DC_i, both DM and DC_i generate a session key sk_i derived from the common secret code c_i (i.e., $sk_i \leftarrow \mathcal{G}(c_i)$ where $\mathcal{G}$ is a randomized algorithm that takes c_i as input and returns sk_i).

In addition, DC_i generates its own public/private key pair puk_i and prk_i to securely exchange messages with the DM application. To register puk_i to DM, DC_i first encrypts (puk_i, DC_i, DM) with sk_i and sends it to DM. After receiving this message, DM decrypts it with sk_i and checks whether DC_i and DM are correctly obtained. If they are valid, DM stores the information about DC_i including its public key puk_i and sends the domain key k to DC_i encrypted with puk_i. This process can be represented in the following manner:

$$\text{DC}_i \longrightarrow \text{DM} : E_{sk_i}(puk_i, \text{DC}_i, \text{DM})$$
$$\text{DM} \longrightarrow \text{DC}_i : E_{puk_i}(k, \text{DC}_i, \text{DM})$$

After receiving the above message, DC_i decrypts E_{puk_i}(k, DC_i, DM) with its private key prk_i and then securely stores k for later end-to-end encryption. In this step, DC_i also checks whether DC_i and DM are correctly obtained by the decryption to prevent modification of $E_{puk_i}(k, \text{DC}_i, \text{DM})$ by an adversary.

Removing Device. When U removes a device i from her domain, U uninstalls the DC application (i.e., DC_i) from the device. While DC_i is uninstalled, it securely deletes the domain key k, its own public/private key pair and then sends the `LEAVE` request message. This process can be represented as follows:

$$\text{DC}_i \longrightarrow \text{DM} :\texttt{LEAVE}, \text{DC}_i, \text{DM}$$

After receiving the `LEAVE` request message, DM displays DC_i's identifier i and asks U to remove DC_i from her domain. When U agrees to remove DC_i, DM deletes all the data related to DC_i (i.e., puk_i and i for DC_i).

Updating Domain Key. When a domain device is stolen or lost, U needs to update the domain key k since she does not want to allow the stolen (or lost) device to still access her personal data. To accomplish this task, U manually selects to remove the stolen (or lost) device from the domain members – DM deletes all the data related to the DC application to be removed.

When U tries to update the domain key, DM generates a new domain key $\hat{k}$ and then searches actively running DC applications via a network interface. If there exist multiple running DC applications, DM (randomly) chooses an application as key updater; DM sends the new domain key $\hat{k}$ with the `UPDATE` message to the chosen DC application (without loss of generality, we assume that DC_i is chosen). This process can be represented in the following manner:

$$\text{DM} \longrightarrow \text{DC}_i : \texttt{UPDATE}, \text{DC}_i, \text{DM}, E_{puk_i}(\hat{k})$$

After receiving the above message, DC_i displays the DM application's identifier and asks U to update the domain key. When U agrees to replace the old domain key k with $\hat{k}$, DC_i decrypts $E_{puk_i}(\hat{k})$ with prk_i to obtain the new domain key $\hat{k}$ and starts downloading all encrypted *data encryption keys* from the cloud server S. After downloading all *data encryption keys* encrypted with the old domain key k, DC_i decrypts them with k, and re-encrypts the *data encryption keys* with the new domain key $\hat{k}$. Finally, DC_i uploads all the *data encryption keys* encrypted with $\hat{k}$ to S and then sends the `UPDATED` message to DM. This process can be represented in the following manner:

$$\begin{aligned} S &\longrightarrow \text{DC}_i : (E_k(dek), id) \\ \text{DC}_i &\longrightarrow S : (E_{\hat{k}}(dek), id) \\ \text{DC}_i &\longrightarrow \text{DM} : \texttt{UPDATED}, \text{DC}_i, \text{DM} \end{aligned}$$

After receiving the `UPDATED` message, DM periodically sends the new domain key $\hat{k}$ to the remaining domain members over secure and authenticated channels created using their public keys until no more DC applications with the old domain key k are found.

Replacing Domain Manger. We also need to consider replacing the domain manager DM with a new one. This task can be implemented by a sequential combination of 'creating domain', 'registering device' followed by 'updating domain key'. After creating a new domain and registering all the current DC applications to the new domain, domain key should be updated with the new domain manager. To support this feature, each DC application keeps the last domain key when it was registered again.

4 Security Analysis

In EnCloud, encryption provides confidentiality of user data – the encrypted data are protected with the data encryption key dek which is randomly generated by an individual DC application; the dek is encrypted again with the domain key k so that only a DC application with the domain key k can obtain the data encryption key dek.

With the information about the user data stored in the cloud storage, an adversary cannot obtain any information about the data encryption key dek and the domain key k except for their associated identifiers if the adversary cannot break the encryption algorithms used for $E_k(dek)$.

Furthermore, even if the adversary can monitor all the communications between the DM and DC applications, the message including the domain key k is encrypted with the DC_i's public key puk_i. Thus, it is infeasible to obtain k for the adversary since the DC_i's private key prk_i securely stays in DC_i.

A major challenge which we address in EnCloud is to prevent an adversary who performs an offline brute force attack on the PIN code c_i. With the captured messages between DM and DC applications, an adversary might still try to guess c_i and check his guesses by attempting decryption of $E_{sk_i}(puk_i, \mathrm{DC}_i, \mathrm{DM})$. However, the session key sk_i is only used to provide the integrity of puk_i. With puk_i alone, an adversary cannot obtain any information about k.

EnCloud engages users in actions by showing on-screen messages with the requested device's identifier to continue performing tasks like registration into a domain, removal of a domain, and update of a domain key. These interactions can help rule out unauthorized commands and man-in-the-middle attacks.

5 Prototype Implementation

In this section, we demonstrate a prototype implementation of EnCloud for Dropbox. The purpose of this implementation shows that the EnCloud system can practically be implemented without incurring significant overhead. We implemented an app on the Android platform for Dropbox. Dropbox APIs (sdk-1.6) were used to upload and download files. We simplified the implementation of EnCloud by assuming that the domain client application already holds a domain key. We particularly focused on the feasibility of end-to-end encryption rather than domain management. For encryption, we used AES-256 (with CBC and PKCS5Padding) in the `javax.crypto` package. When a user uploads a file d, this app internally creates two files where one is for d and the other one is for its *data encryption key* dek. The file d is encrypted with dek; dek is encrypted with a domain key k internally stored in the EnCloud app.

Table 2. The execution-time measurements (SD: Standard Deviation) for encryption/decryption operations and the total processing with varying file sizes. The time units are in milliseconds.

File Size		Encryption	Decryption	Total
20MB	Mean	1545.73 (1.38%)	4710.73(4.19%)	112366.16
	Max	1919.00 (0.74%)	7459.00 (2.88%)	259086.00
	SD	207.96 (0.63%)	914.07 (2.77%)	32981.14
40MB	Mean	2796.26 (1.53%)	8032.70 (4.40%)	182738.23
	Max	3537.00 (1.40%)	9798.00 (3.88%)	252612.00
	SD	294.50 (0.76%)	984.22 (2.54%)	38808.89
60MB	Mean	4394.83 (1.44%)	13855.57 (4.54%)	304931.63
	Max	5327.00 (1.19%)	19079.00 (4.25%)	448645.00
	SD	599.61 (0.70%)	2585.83 (3.02%)	85367.29
80MB	Mean	7251.43 (1.40%)	32864.33 (6.33%)	519346.70
	Max	92648.00 (7.22%)	57218.00 (4.46%)	1281747.00
	SD	674.48 (0.29%)	8799.67 (3.85%)	228689.78
100MB	Mean	8389.26 (1.22%)	69905.77 (10.16%)	687818.37
	Max	9779.00 (0.39%)	79668.00 (3.16%)	2523523.00
	SD	1257.80 (0.31%)	5598.06 (1.36%)	410183.42

When a file is uploaded and downloaded, we measured the execution-time incurred by encryption and decryption operations compared with the total execution-time. To decrease the bias associated with the performance realized from the testing samples, we repeated the test procedure 30 times with varying file sizes from 20MB to 100MB. We used a Samsung Galaxy Note 2 (with a 1.6 GHz Quad-core CPU, 533MHz GPU and 2GB RAM) running the Android 4.3 version, and equipped with a non-congested 100 Mbit/s WiFi connection to a LAN that was connected to the Internet via a Gigabit-speed link; the execution-time overhead was measured using the method `System.currentTimeMillis()`. The experimental results are shown in Table 2.

The test results show that the execution-time overhead incurred by encryption and decryption operations is marginal compared to the overall overhead. For example, when the file size was 20MB, the total execution-time was 112,366 milliseconds on average while the execution-time measurements for two encryption and two decryption operations were only about 1,545 and 4,710 milliseconds, respectively, on average (for the 30 trials; about 1.37% and 4.19% of the total execution-time). Although the average encryption and decryption time was greatly affected by the file size, the additional overheads of encryption and decryption operations were still manageable: the average additional delay incurred by encryption and decryption was less than 11.5% of the total execution time in the worst case. This is because file transfer may overwhelm other operations such as encryption and decryption. Interestingly, we can see the significant difference in execution time between encryption and decryption. We surmise that the underlying physical characteristics of NAND flash mem-

ory used in the prototype implementation may explain this characteristic – read and write operations are needed only once for encryption whereas one read and two write operations are needed, respectively, for decryption. If we consider the fact that read is typically at least twice faster than write for flash memory, the performance difference between encryption and decryption seems natural.

We now discuss the execution-time for updating domain key. When a domain device is stolen or lost, the domain key should be updated. We tested the key update procedure 30 times with a 100MB file under the same conditions to measure the execution-time to process this task. The average execution-time was 2,258 milliseconds, concluding that the proposed key update procedure is efficient compared with, for instance, the case of the time it takes to download (or upload), encrypt, and decrypt the same file with a domain key k, which yields the total average execution-time of 364,393 milliseconds.

The space overhead can generally be computed with the number of stored files as follows: For a file d to be stored in the cloud storage, the EnCloud system stores the following files: *encrypted data* = $(E_{dek}(d), id_d)$ and *data encryption key* = $(E_k(dek), id_d)$ where dek is a randomly generated key and id_d is a unique identification string for d. If we use a m-bits block cipher for the encryption E, $|E_{dek}(d)| \leq |d| + m$ since the maximum length of the padding is less than m. To store $E_k(dek)$ and two id_d strings, the additional overhead of m bits and $2 \cdot |id_d|$ is also needed, respectively. Therefore the worst case space overhead is $n \cdot (2 \cdot m + 2 \cdot |id_d|)$ where n is the number of files to be securely stored in the cloud storage. For example, if 30,000 private files are stored and the EnCloud system uses AES-256 for encryption E with 256 bits for id_d, the total storage overhead is about 3.66MB ($\approx 30,000$kb). In the EnCloud system, the space overhead is proportional to the number of files to be stored and is rather marginal.

6 Related Work

The cloud computing [12] promises many opportunities while posing a unique security and privacy challenges. Takabi et al [14] argued that privacy is a core issue in all the challenges facing cloud computing – many organizations and users are not comfortable storing their data on off-premise data centers or machines.

End-users do not trust cloud services to store their personal data, and would prefer to store the data on their devices at home. Ion et al. [7] showed that many users believe that the data stored in cloud services can be exposed or stolen. Similarly, a survey by the Fujitsu Research Institute showed that about 88% of cloud customers were concerned with unauthorized access to their data [5].

In practice, the user data in cloud services are often exposed to the risk of unauthorized access. For example, Dropbox recently suffered an authentication bug that made it possible to log into some users' accounts without a password

for about 4 hours [10]. In addition, the Snowden's leaks [1] explain why privacy concerns on the cloud are not far fetched – some intelligence agencies (e.g., NSA and GCHQ) have collected online users' data, even from cloud providers. These cases show how cloud computing services could be vulnerable in real-world situations, not only by external but also internal adversaries.

Kamara and Lauter [8] proposed several architectures for cryptographic cloud storage based on cryptographic primitives such as *searchable encryption* and *attribute-based encryption* to mitigate privacy risks in cloud services. They are particularly interested in sharing a secure cloud storage between users. We extend their work for a different scenario where a user wants to share her personal data between her cloud applications. Our focus is to design a simple, efficient, and general framework that provides end-to-end encryption between cloud applications in the data owner's personal domain.

Slamanig [13] demonstrated how to use side channels (CPU time or storage space) in cloud to infer the behavior of co-located users. Khan and Hamlen [9] proposed a framework called AnonymousCloud based on Tor [4] (which is designed to resist traffic analysis) in order to conceal ownership of cloud data.

There are several solutions providing end-to-end encryption for off-premise user data. To that end, Voltage Security (`www.voltage.com`) introduced a commercial security product based on identity-based encryption (IBE) [2] to protect files and documents used by individuals and groups. While usable in many applications, like email, an obvious shortcoming of the technique is that the provider can also decrypt its users' data for having access to users' private keys. In other words, in order for the system to work, users have to trust the provider, a requirement we set to avoid in this work. Encryption services such as Boxcryptor (`https://www.boxcryptor.com`) and Cloudfogger (`http://www.cloudfogger.com`) encrypt user data locally and then the encrypted data sync with the user's cloud storage. However, the security of their solutions relies on the difficulty of guessing passwords since the decryption keys in their services are protected with a password typed by the user at login. Unlike these products, EnCloud is designed to provide end-to-end encryption to defeat offline dictionary attacks; all encryption and decryption keys in EnCloud are located at the client side rather than the server side.

7 Conclusions

We proposed a system named EnCloud against powerful adversaries (e.g., an intelligence agency) who can access cloud-based data. EnCloud is designed to provide end-to-end encryption between cloud applications in the data owner's personal domain so that the private data are securely stored on the cloud server in

an encrypted form while the data owner's EnCloud applications are only allowed to decrypt the encrypted data.

We also demonstrated EnCloud's feasibility by analyzing the security and performance on a prototype implementation for Dropbox. We highlighted that the additional execution-time overhead incurred by EnCloud is not significant compared with the overall execution-time. This shows that EnCloud can be implemented without a significant overhead while providing an effective end-to-end encryption between cloud applications. However, in this prototype, the adversary can learn some meta attributes about files (e.g., creation time, size, etc.). In future work, we will consider how to hide such information.

8 Acknowledgements

This research was supported by the MSIP (Ministry of Science, ICT & Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2014-H0301-14-1010) supervised by the NIPA (National IT Industry Promotion Agency).

References

1. Ball, J., Borger, J., Greenwald, G.: Revealed: how US and UK spy agencies defeat internet privacy and security (2013)
2. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. SIAM J. of Computing **32**(3) (2003) 586–615 extended abstract in Crypto'01.
3. Daemen, J., Rijmen, V.: The Design of Rijndael. Springer-Verlag New York, Inc. (2002)
4. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: Proceedings of the 13th Conference on USENIX Security Symposium. (2004)
5. Fujitsu Research Institute: Personal data in the cloud: a global survey of customer attitudes (2010)
6. Gellman, B., Poitras, L.: U.S., British intelligence mining data from nine U.S. Internet companies in broad secret program (2013)
7. Ion, I., Sachdeva, N., Kumaraguru, P., Čapkun, S.: Home is safer than the cloud!: Privacy concerns for consumer cloud storage. In: Proceedings of the Seventh Symposium on Usable Privacy and Security, ACM (2011) 13:1–13:20
8. Kamara, S., Lauter, K.: Cryptographic Cloud Storage. In: Proceedings of the 14th International Conference on Financial Cryptograpy and Data Security. (2010) 136–149
9. Khan, S., Hamlen, K.: Anonymouscloud: A data ownership privacy provider framework in cloud computing. In: Proceedings of the 11th International Conference on Trust, Security and Privacy in Computing and Communications. (2012) 170–176
10. Kincaid, J.: Dropbox security bug made passwords optional for four hours (2012)
11. Mearian, L.: No, your data isn't secure in the cloud (2013)
12. Mell, P., Grance, T.: The NIST definition of cloud computing. `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf` (2011)
13. Slamanig, D.: More privacy for cloud users: Privacy-preserving resource usage in the cloud. In: 4th Hot Topics in Privacy Enhancing Technologies, HotPETs. (2011)
14. Takabi, H., Joshi, J.B.D., Ahn, G.J.: Security and privacy challenges in cloud computing environments. IEEE Security and Privacy **8**(6) (2010) 24–31