Layered Security for Storage at the Edge: On Decentralized Multi-factor Access Control

Esiner, Ertem; Datta, Anwitaman

2016

Esiner, E., & Datta, A. (2016). Layered Security for Storage at the Edge: On Decentralized Multi-factor Access Control. Proceedings of the 17th International Conference on Distributed Computing and Networking, 9-.

https://hdl.handle.net/10356/81018

https://doi.org/10.1145/2833312.2833452

Layered Security for Storage at the Edge: On Decentralized Multi-factor Access Control

Ertem Esiner
School of Computer Engineering
Nanyang Technological University
Singapore
ertem001@ntu.edu.sg

Anwitaman Datta
School of Computer Engineering
Nanyang Technological University
Singapore
anwitaman@ntu.edu.sg

ABSTRACT

In this paper we propose a protocol that allows end-users in a decentralized setup (without requiring any trusted third party) to protect data shipped to remote servers using two factors - knowledge (passwords) and possession (a time based one time password generation for authentication) that is portable. The protocol also supports revocation and recreation of a new possession factor if the older possession factor is compromised, provided the legitimate owner still has a copy of the possession factor. Furthermore, akin to some other recent works, our approach naturally protects the outsourced data from the storage servers themselves, by application of encryption and dispersal of information across multiple servers. We also extend the basic protocol to demonstrate how collaboration can be supported even while the stored content is encrypted, and where each collaborator is still restrained from accessing the data through a multi-factor access mechanism. Such techniques achieving layered security is crucial to (opportunistically) harness storage resources from untrusted entities.

CCS Concepts

•**Security and privacy** → **Multi-factor authentication;** *Database and storage security;* **Access control;**

Keywords

Layered security, multi-factor access control, data out-sourcing, edge computing, user controlled encryption, erasure codes

1. Introduction

Since its early days, there has been a continuous flux in computing paradigm, moving back and forth between centralization and decentralization: from mainframes to PCs, peer-to-peer and federated grids to cloud to mention some prominent trends across decades. Fuelled by evolution of miniaturized hardware, there is a renewed swing back towards (partial) decentralisation, wherein edge elements in the system reassert a prominent role not only as consumer of resources, but also act as active contributors to the overall computing infrastructure. There is a growing cognizance from various perspectives and interpretations, and under various nomenclatures such as fog computing [14], edge computing [27], internet of everything (IoE) [22], that edge elements provide a natural way to scale and solve last mile problems, by augmenting and extending the 'core' cloud computing and service infrastructure. It is worth disambiguate at this juncture that though modern high-end mobile devices (phones/tablets/'things' from IoT) are capable of running process heavy applications, in this work we do not focus on such energy restricted devices as the storage edge servers (these may very well be the clients though), but refer by edge services more the kind of setups envisioned under fog computing [14].

Storage is a cardinal functionality, along with other elements such as computation and services. In order for edge/fog computing to gain adequate traction, security and reliability of such an edge resource based storage service would thus be of paramount importance. The focus of this work is primarily on the security, and in particular, on realising layered security (multi-factor access control) in a decentralised set-up, where the edge elements cannot be fully trusted, nor a trusted centralized entity may be desirable. At this juncture we will like to emphasize that the aim here is to go beyond the popular 'user controlled encryption' paradigm that is already used in several cloud service offeings, in that, typically the control factor the end user has is knowledge of a password. The objective of this work is to augment the knowledge factor with an orthogonal possession factor.

Given the emphasis on decentralization, there are some natural echoes with works done in the context of peer-to-peer (P2P) storage systems, and the proposed approach is naturally applicable in P2P and hybrid environments [20]. While our work also inherently supports fault-tolerance and availability, it is not the emphasis, and accordingly, the treatment of this work is more on layered security and not on issues like churn, which is a central theme of P2P literature. The works closest to our treatment in the P2P literature are those of [21, 40], but these works do not explicitly address issues of layered security, as is the case in this paper. We do use the user controlled encryption paradigm (the user holds the encryption key, and the storage servers can't decrypt it as such) as several recent commercial cloud service offerings [4, 6] do natively, and alternatively, users can separately carry out using tools like TrueCrypt or Boxcryptor [1, 5]. But in addition, we leverage on dispersing the content across multiple storage services to gain an additional layer of security, which makes [10, 25] the closest works in the cloud security literature. However, these prior works rely on a single control (knowledge factor) at the end-user's end, while the approach we propose augments the security of user controlled encryption by explicitly realizing an orthogonal possession factor.

It is often desirable to deploy multiple controls in order to realise a layered defense in protecting resources or data. A ready example of such layered defense is multi-factor authentication based access control. Typically, multi-factor authentication relies on information which can be divided across orthogonal factors such as knowledge (something one knows, e.g., password), possession (something the user has, e.g., a hardware token) or inherence (something the user is, e.g., biometric information such as fingerprint or retina scan).

Traditionally, the authentication service provider facilitating an additional factor authentication has the burden, but also control over the the infrastructure to support multi-factor authentication, and is implicitly trusted with the same. It can thus become a single point of vulnerability. For instance, breach of RSA's SecureID[1] is a well known incident.

Furthermore, in adopting a multi-factor access control solution provided by a storage service provider, the end user can prevent other entities from accessing the content, but do not achieve layered security from the storage service or access control service providers themselves. The obvious and popular solution of encrypting the content before shipping it to the outsourced storage service will only have a single factor security from the service or any entity which can access the content of the storage servers of the service provider directly, e.g., insiders, or someone who has compromised the service provider's institutional security.

A natural workaround to protect data from individual service provi- ders (in addition to applying user controlled encryption) is to distribute the data across multiple services [10, 26] using a threshold cryptography or erasure coding technique. Such redundant dispersion provide additionally the benefit of fault-tolerance [7, 10, 26], even when content from a small subset of used service providers become unavailable.

However, when multiple services are used, then all these services may not natively support multi-factor authentication, and moreover it may not be desirable to introduce a trusted third party to realise the functionality universally. To fully benefit from opportunistic usage of available resources at the edge, a decentralised mechanism is preferable instead.

Consequently, in this paper we explore a holistic technique to realize multi-factor access control which provides layered security with respect to both extrinsic parties, as well as the storage service providers, is robust against the collusion of a (small) fraction of service providers, and can be deployed and controlled by individual data owners in a decentralised manner without requiring any trusted third party, or collaboration among the storage services. We also demonstrate how the basic protocol can be extended to facilitate collaborative manipulation of the data (i.e., not only read operations, but also write operations) among multiple users approved by a data owner.

Though the overall objective looks challenging, in fact, existing works from cloud and P2P security literature realize different subsets of the objectives, and the main contribution of this paper is to put these techniques together in a seamless manner to achieve all the properties in harmony. The elimination of centralized entities to realise the solution means that the approach decouples the user's identity from the access-control mechanism, which then can furthermore provide the end user certain degree of anonymity. We next discuss related works, which provide the founding pillars for the proposed solution, and also differentiate our work with respect to these prior works.

2. Related Work

There are many different security aspects to data outsourcing, and a myriad of approaches to address those respective issues. Encrypting the content before outsourcing it to a storage service is the most obvious thing to do. There are several commercial solutions that natively do so [4, 6], but furthermore, users can use other solutions such as TrueCrypt [5] or Boxcryptor [1] on their own.

In addition to the basic encryption based security that is already available for use, the data outsourcing security literature is rich with approaches that go beyond mere encryption, to provide either further functionality or further security and privacy. Some prominent ones among these include works on search over encrypted content [12], determining the integrity and ensuring availability of the outsourced data [7, 10, 26, 36], masking the query (and results) information [31], etc. Depending on the specific treatment of the problems at hand, some of these related works are of peripheral relevance to the presented work, while others are more directly related. In the rest of this section, we will focus mainly on those works which have direct bearing to some of the techniques that are used in the current work.

2.1 Multi-cloud and multi-server storage

In the recent years, several approaches to leverage multiple cloud services have been proposed. [16] demonstrates that freemium service resources can be easily aggregated, and in doing so a better end user performance can be achieved. Erasure coded redundancy across cloud services to achieve fault tolerance has been explored in [7, 10, 26]. The dispersal of information across different (and presumed independent) services has also been used as a mechanism for security in [10, 25]. It is worth noting that the model used in [25] is somewhat ambiguous with respect to the relationship among the multiple servers. Parts of the processes require some cooperation among the servers, and yet part of the security is derived from the distribution of data across servers.

Our proposed approach shares the basic design choices of [10, 25], in that we also apply encryption at the end user (client) and dispersal as a two-pronged approach to derive security. However, the specific authentication mechanism proposed in our approach introduces a possession factor as an orthogonal control in addition to the knowledge factor of passwords, which allows us to generate time limited one time authentication tokens, and is the key novelty providing multi-factor access control in a decentralised manner.

It is also worth noting that the usage of erasure coding in our approach naturally provides some fault-tolerance, similar to [7, 10, 26], however, the erasure code itself does not add anything to the security. Specifically, the dispersal is desired, but the additional redundancy (while beneficial for fault-tolerance) in fact has slight detrimental effect from a confidentiality point of view. We elaborate this issue later in Section 6.

P2P storage systems literature extensively study the utility of erasure coding based information dispersion, mainly for reliability reasons. In [40], a threshold cryptography based technique is used to store a user's private keys over a p2p network, and the main emphasis is on choosing the servers based on a trust model in order to minimize the chance that an adversary can control adequate servers to gain access to the private key of a specific user. Follow-up works have explored variants to emulate password based login in a P2P network [21]. Our work does not make any of the several assumptions, e.g., of (partial) global information using a gossip algorithm, or an additional extrinsic directory infrastructure (e.g., DHT services) as these works from P2P domain assume, but also, in essence, this work addresses a rather different problem despite

[1] http://en.wikipedia.org/wiki/RSA_SecurID#March_2011_ system_compromise

the apparent similarity at a high level (all these works do address different kinds of access control).

2.2 Authentication factors

In addition to hardware based tokens for creating one time passwords, such as RSA's SecureID, there are many software based solutions. Popular software solutions like Google's one time password (OTP) [3] is based on Time-Based One-Time Password Algorithm (TOTP) defined in IETF's RFC 6238 [30]. We leverage the same principle in our approach to carry out time based authentication.

Public-key cryptography has long been used as a mechanism for authentication. In our context, we do not need to bind a public key to any specific identity, in that the storage server just needs to know that a current access request is coming from a party which possesses the private key corresponding to the public key, which the original data owner had provided when storing the object. This eliminates the burden of a public key infrastructure (PKI). Specifically, we use the private key to create a time-based one time authentication code that a storage server can verify based on the public key it holds. The basic usage of public key cryptography is thus similar to other schemes using public key cryptography for authentication [17]. However, in typical deployment scenarios the service provider/server generates the possession factor and the private/public keys as opposed to our approach. The main novelty of our deployment is however in explicitly leveraging on it to create an orthogonal possession factor separate from the knowledge factor.

2.3 Proxy Re-encryption over Cloud Storage

One of the natural challenges of data outsourcing over untrusted storages is to allow multiple parties collaborate. For confidentiality, the data is encrypted by the owner, therefore it is masked not only from the servers but everyone else that possibly needs to collaborate on the data. Using an encryption where the key is shared by every collaborator is a naive and inflexible option. Mambo and Okamoto first proposed a proxy re-encryption (PRE) scheme to delegate the ability to decrypt the ciphertexts [28], where a proxy server can transform data encrypted for Alice to the one encrypted for Bob, without learning any function out of the plaintext. Later, symmetric PRE schemes were prosed [18, 37], where security guarantees and proxy functions are defined. The schemes can be broadly divided into two categories: unidirectional and bidirectional, depending on if the re-encryption works only from Alice to Bob or works for both directions. Additionally, PRE scheme is said to be of multi-use if the transformation is transitive and can continue from Bob to someone else and so on. Blaze et al. proposed an El Gamal based scheme which is multi-use and bidirectional [11]. While the main idea stays, many later works focused on solving outstanding issues such as; Bob's colluding with the proxy resulting with revealing Alice's secret [19], for some settings bidirectional property is not required [19, 23], thus not proper and lastly, addressing vulnerabilities to chosen ciphertext attack (cca-security) [13, 23, 35].

Subsequently, proxy-re-encryption has been applied to secure storage systems by Ateniese et al. for collaboration and sharing [8] for the files stored on distributed replicas. The idea was then generalized and applied to the settings where data is dispersed using erasure coding instead [24]. The PRE scheme employed in [8] is unidirectional and the protocol itself is designed employing a centralized access control server. Our approach to faciliate collaboration is inspired by the fully distributed setting of [24], and uses bidirectional PRE to support write operations by multiple collaborators. In this aspect, the contribution or emphasis of our work (in section 5) in not the way PRE is used for collaboration, but on the extension of our basic multi-factor access enforcement (main contribution of this paper, the protocol presented next in section 3) mechanism to the collaborative setting.

3. Protocol

We next discuss our protocol (shown in Figure 1) which achieves layered security against both storage service providers as well as any other third party entities by the application of encryption and dispersal. While, at a high level, usage of these two approaches in conjunction is along the lines of prior works [10, 25], the novelty in our approach is derived from the usage of a orthogonal possession factor in addition to the usage of knowledge factor, as a mechanism to control access. We note however that the proposed protocol does not follow the strict interpretation of conventional centralized multi-factor authentication, in that the enforcement of these multiple factors is not carried out at the storage servers which uses only an authorisation information sent by the client, however the client does need to use both a knowledge and a possession factor together in order to generate the authorisation information to start with, and therefrom our protocol derives multi-factor control.

3.1 Access-control factors

The proposed design enables the enforcement of access control using two factors - knowledge and possession. Specifically, an encryption password *epp* is used, which acts as the primary knowledge factor. In our proposed solution, another password $pw = [pw_1|pw_2]$ is also used (which can be seen as a knowledge factor), but mainly for additional protection of the possession factor. We use '|' to indicate concatenation.

The possession factor is a secret package *SP* which comprises of a randomly (locally at client) generated secret token *sk* and a public/private key pair $priv_k/pub_k$. These randomised content of *SP* need to be cryptographically strong. Furthermore, *n* salt values $salt_i$ (alternatively, a seed to a random sequence generator may be stored for storage efficiency) and address information of *n* servers ip_i are also stored as part of *SP*. *n* is a design parameter of the protocol that is described below, and it (along with another parameter $m < n$) determines the number of servers across which information is dispersed, and the quorum needed to retrieve back the information. Note that we will not particularly leverage on the randomness of the $salt_i$ to derive security and thus we have not emphasised on it be cryptographically strong. Identity of the servers in *SP* is likewise not particularly vital to the security of the proposed protocol, however if kept confidential, it provides an additional layer of security by obfuscation. The content of *SP* acts as the possession factor. For further security, one may thus encrypt this content using pw_1 part of *pw*.

3.2 Setup

A client needs to input the passwords *epp* and *pw*, and initialise the remaining content of secret package *SP*. For that the user may need to explicitly determine the choice of the *n* servers $(S_0, S_1, ..., S_{n-1})$, or it may be automated through a background process - depending on whichever is suitable for a specific setting. The servers in turn may be dedicated services like Amazon web services (AWS), or edge resources in fog computing or peer-to-peer networks, or a mix. The choice of these servers generates the tuple *ips:* ip_0, ip_1, ..., ip_i, ..., ip_{n-1} stored in *SP*. Overall, the following secret package *SP* is created. Note that the public key pub_k need

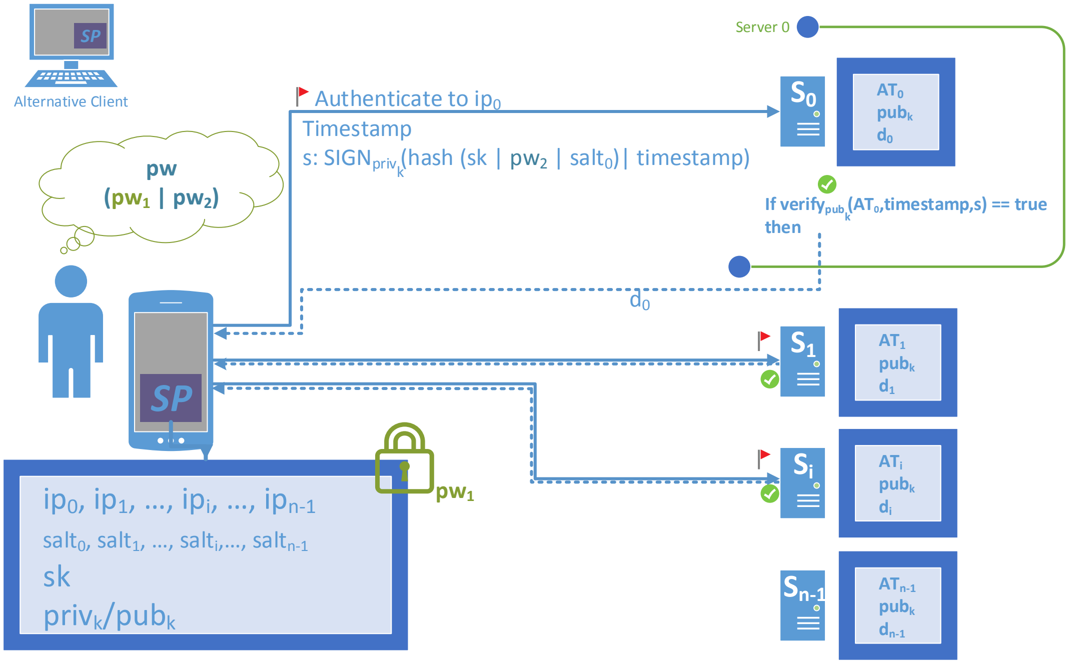

Figure 1: Client authentication and data retrieval.

not be kept confidential, but just for the sake of portability has been stored along with the remaining secret package.

$$SP = \begin{cases} \text{ips: } ip_0, ip_1, \ldots, ip_i, \ldots, ip_{n-1} \\ \text{salts: } salt_0, salt_1, \ldots, salt_i, \ldots, salt_{n-1} \\ \text{secret token: } sk \\ \text{private/public key: } priv_k/pub_k \end{cases} \tag{1}$$

The possession of *SP* acts akin to possession of a token, but because of its software implementation, it can be used in a portable manner across different devices if the user so wishes. Consequently, similar to the use of a secret PIN for hardware tokens, we propose to store the *SP* encrypted with pw_1.

The client generates distinct authentication tokens AT_i per server as follows, and uploads pub_k and AT_i to the respective servers S_i at the set-up phase.

$$\mathrm{AT}_i = hash(sk|pw_2|salt_i) \tag{2}$$

3.3 Storage outsourcing

When a user needs to store an object O, the client splits the object in m parts, and applies a maximum distance separable (MDS) erasure code such as Reed-Solomon code to generate n encoded parts. These n parts are then encrypted using distinct symmetric encryption keys sek_i (per server symmetric encryption key creation is described in 3) to generate encrypted data d_i which are shipped to respective servers S_i. "1" is concatenated to the salt value only to differentiate it from its use in AT.

$$sek_i = hash(sk|epp|salt_i|1) \tag{3}$$

3.4 Multi-factor data access

When the user wants to access the original data object, she first provides the password $pw = [pw_1|pw_2]$ to the client application, which then decrypts and retrieves *SP*. Any m (or more) of the *ips* are chosen, and corresponding authentication tokens AT_i' (computed as in Equation 2). The current system time is used to pick a *timestamp*, and finally an authentication package is created based on signature using the private key $priv_k$ retrieved from *SP* and sent to server S_i.

$$\textbf{Authentication package} = \begin{cases} timestamp \\ SIGN_{priv_k}(AT_i'|timestamp) \end{cases} \tag{4}$$

Upon receipt of a data retrieval request along with the authentication package, the server verifies the time-stamped authentication package (using AT_i and pub_k that have been stored at the server during the setup phase). The server sends d_i back to the client only if satisfied with the authentication package. We note that the timestamp used in our approach is similar to the TOTP [30] approach, and limits the time for which the specific authentication token is valid, thwarting replay attacks. Typically the client would store multiple objects in the storage system. One can furthermore include an object identifier before creating the signed authentication package, to further limit any illegitimate access. Finally, in a decentralized setup, the servers and the client's system clock may not be in perfect synch, however, the client can first query the server time in order to correct the necessary offset. For the sake of simplicity, we have excluded details of such variations.

Once the client receives at least m of the encrypted d_i pieces, it can decrypt them using the knowledge factor *epp*, and then carry out (erasure) decoding in order to reconstruct the original object O.

3.5 Updating possession factor

After successful authentication at each server, the client can replace the older authentication token and public key stored at the servers using new ones, generated using a new password pw', and updating the locally stored *SP* accordingly. This will render *SP*s stored (if any) at any other locations invalid. Note that pw_2 is never stored locally, hence even if an adversary had gained access and broken the encryption in order to retrieve the older *SP* in plain text, they may not easily revoke the legitimate user's access in a similar manner.

The data owner can always re-encrypt and request to replace d_is using a new *epp*, however such an operation may be less meaningful. An attacker controlling a compromised server can continue to be in possession of the older version of encrypted data. Hence we don't explore this option.

4. Protocol security discussion

The proposed mechanism provides layered security, where multiple controls need to be breached in order to gain access to the stored object. We next analyse the robustness of the proposed approach against compromise of a partial set of controls, and what all the adversary will have to subsequently do in order to gain illegitimate access. We note for the discussions below that situations where a storage server is compromised is somewhat outside the scope of multi-factor access control per say. However, our layered security mechanism naturally provides some security even under these circumstances, and thus compromised servers are discussed accordingly. Furthermore, we assume that a server that is not compromised follows the proposed protocol properly, and likewise, if a storage server deviates from the protocol in any manner, we consider that server compromised.

Adversary gets hold of a device containing *SP*: The adversary will first require to break the encryption to retrieve the content of *SP*. In the process, the adversary will be able to estimate pw_1. Even so, the adversary will be unable to authenticate with any server and obtain any d_i without the knowledge of pw_2. It will also not be able to prevent the legitimate owner to continue to gain access to the content, or revoke the original *SP*. With the identity of the servers, it may however try to compromise the servers themselves. If it manages to compromise a quorum of at least m servers, then it can obtain the corresponding encrypted pieces d_i. Each of these encryption is done with a distinct key which the attacker will have to break in order to gain access to the actual content.

Adversary learns both the passwords used by the user: The adversary will have to get access to the device storing *SP*, either by gaining physical possession or by compromising it over the network. Then the adversary will have access to the content. Note, if the adversary knew only *epp* but not *pw* then there would be further barrier for illegitimate access. Alternatively, if the adversary cannot get hold of *SP*, the adversary can monitor the network traffic in order to identify the servers at which the encrypted data is dispersed. Even then, without content of *SP* the adversary will not be able to authenticate. The adversary will thus have to compromise at least m of these servers before getting hold of the encrypted pieces. Knowing *epp*, but without knowing *sk* which is stored in *SP*, the adversary is forced to break the encryption of each of these pieces individually before gaining access to the content.

One or few servers are compromised: Compromising some servers do not provide information about the identity of the other servers, nor information necessary to authenticate with the other servers. Furthermore it does not provide any knowledge on how to decrypt the individual pieces stored at any of the servers. However if the adversary then goes on to break the encryption of some of the pieces, even if the number of such pieces obtained in less than m, the adversary may gain partial information regarding the original content. Application of threshold (secret sharing [9]) cryptography instead of erasure coding would alleviate this problem, but at the cost of greater overheads. The user may also want to use distinct public/private key pair per server, so that an adversary controlling multiple compromised servers cannot create an obvious link of content of an individual user (assuming that the server stores data from multiple users). However, server logs and access timing analysis can easily give away such association, and despite the overheads the value of deploying distinct public/private key per server is questionable. Hence, we have discarded this design option.

Attacker eavesdrop on the communication between client and servers: We first note that this scenario is somewhat beyond the scope of multi-factor access control. The attacker will gain access to the encrypted pieces being communicated back to the client in this case, and the adversary can then try to break the encryption. More crucially, the adversary can try to replay the authentication token to fetch encrypted pieces of other objects stored (but not queried in that particular session) by the user. The authentication token is however time limited, thus limiting the adversary's attack. In fact, it is extremely simple to modify our time limited token creation by also using the object identifier in addition when computing the hash, so that the token cannot be used to fetch any other object even within the limited window of opportunity.

The user loses the possession factor or forgets one of the passwords: Unlike in a centralised solution where the service provider may be able to establish the user's identity using other out-of-band mechanisms to issue new credentials, in our approach, because of its decentralized design, there is no mechanism for the data owner to gain access back to the data anymore (barring the options any attacker would have had, as enumerated above).

Using a compromised client: We so far assumed the client device is itself not compromised and congregates all trust in the client's device. Therefore a compromised client device during the setup and/or authentication phase (such as a key-logger or a cuckoo attack [32]) may defeat the security of the proposed scheme. When the user provides the passwords and the SP is decrypted, the attacker can obtain all the necessary information not only to authenticate to servers once but also to be used for later interactions as well. A totally compromised client during the authentication process already yields breach, but third party OTP mechanisms can still prevent the attacker from carrying out future transactions. This is not the case with the current presentation and implementation of our work. However, though it needs more extensive investigation deferred for future, we speculate that this vulnerability can in fact be mitigated by decoupling the private key used for the signature scheme to yet another device. This device that holds the private key and runs the signature scheme can be either an easily attainable device (such as a cellphone, smartwatch, etc.) or a specifically manufactured token that is initialized during the setup phase.

As a final note, we will like to add that a further layer of security based on inherence factor may be added to our proposal. This can be achieved by eliminating *sk*, and instead of storing it in *SP*, generating it on the fly based on a suitable biometric input. Note that if someone opts to compromise the servers and break the encryptions, then use of biometrics may not provide further security. But for the other attack vectors, creating *sk* on the fly using inherence would add to the overall security.

5. Collaboration

Let's say Alice has set up a secure distributed storage space using our proposed multi-factor access control protocol and then naturally, and subsequently she needs to collaborate with her colleague Bob, who needs to have read/write rights on the data. In this sec-

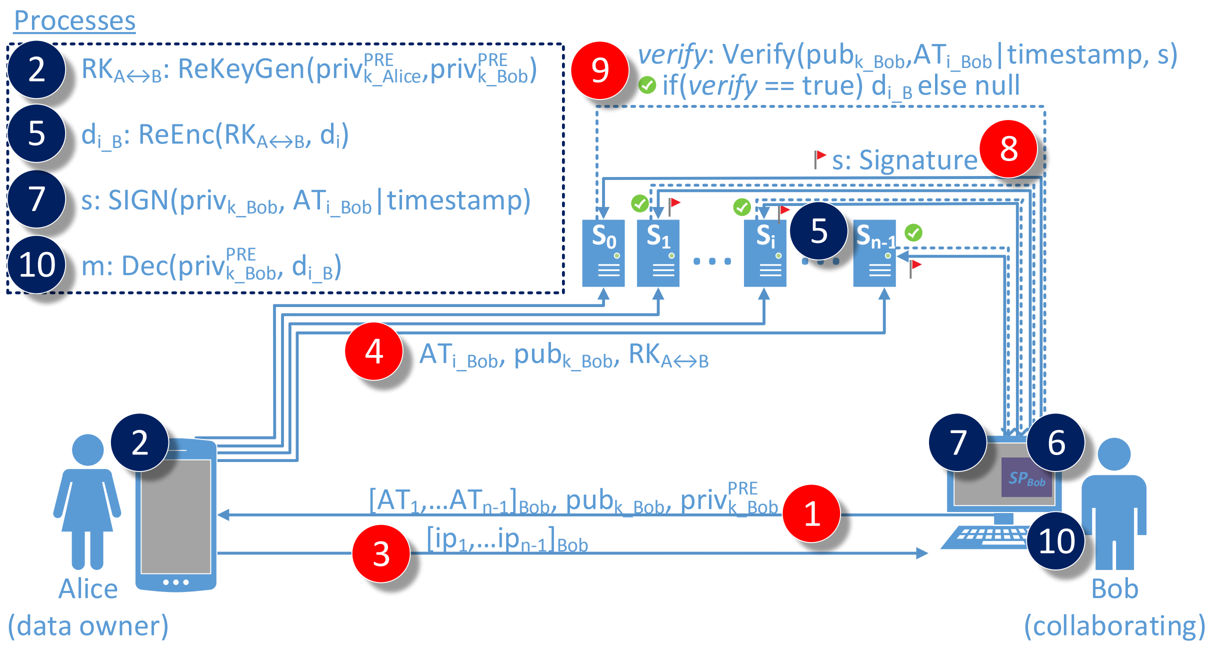
Processes
2 $RK_{A\leftrightarrow B}$: ReKeyGen($priv^{PRE}_{k\_Alice}$,$priv^{PRE}_{k\_Bob}$)
5 d_{i_B}: ReEnc($RK_{A\leftrightarrow B}$, d_i)
7 s: SIGN($priv_{k\_Bob}$, AT_{i_Bob}|timestamp)
10 m: Dec($priv^{PRE}_{k\_Bob}$, d_{i_B})
9 verify: Verify(pub_{k_Bob},AT_{i_Bob}|timestamp, s)
if(verify == true) d_{i_B} else null
s: Signature 8
S_0 S_1 S_i 5 S_{n-1}
4 AT_{i_Bob}, pub_{k_Bob}, $RK_{A\leftrightarrow B}$
2
7 6
SP_{Bob}
$[AT_1,...AT_{n-1}]_{Bob}$, pub_{k_Bob}, $priv^{PRE}_{k\_Bob}$ 1
3 $[ip_1,...ip_{n-1}]_{Bob}$
10
Alice
(data owner)
Bob
(collaborating)

Figure 2: Bob collaborating on data owned by Alice

tion we elaborate on how Alice can set Bob up to gain access to and decrypt the data owned by (and encrypted under the secret of) Alice, and update the data, while also enforcing a multi-factor access control on such third party.

Similar to related prior works [8, 24], in principle, the encrypted data can be collaborated on using a proxy re-encryption (PRE) scheme where the untrusted server can re-encrypt the ciphertext under a private key $priv^{PRE}_{k\_Alice}$ into another ciphertext of the same data under another private key $priv^{PRE}_{k\_Bob}$. The untrusted server cannot, however, access any part of the plaintext or get any function out of it. For Bob to be able to update the data, the PRE scheme employed should be *bidirectional*. Our proposed protocol varies mainly in how the multi-factor access control is set-up and enforced, rather than in its use of proxy re-encryption per say. We next summarize the main operational primitives of bi-directional proxy re-encryption, before elaborating our protocol, which leverages on these operations.

5.1 Bidirectional Proxy Re-encryption

For the next sections we employ a generic proxy re-encryption scheme. A bidirectional proxy re-encryption scheme between Alice and Bob consists of the following 5 algorithms;

- Key generation **Keygen**(sr) $\rightarrow (pub^{PRE}_k, priv^{PRE}_k)$, where it takes the security requirements (such as key length, protocol specifications, etc.) as input and returns a public and a private key.
- Re-encryption key generation **ReKeyGen**$(priv^{PRE}_{k\_A}, priv^{PRE}_{k\_B})$ $\rightarrow RK_{A\leftrightarrow B}$ that takes two secrets of two parties as input and returns a re-encryption key that works both ways between Alice and Bob.
- Encryption **Enc**$(priv^{PRE}_k, m) \rightarrow d$ that encrypts the data with public key and returns the ciphertext.
- Re-encryption **ReEnc**$(RK_{A\leftrightarrow B}, d_A) \rightarrow d_B$ that transforms ciphertext for Alice with the re-encryption key and outputs ciphertext for Bob.
- Decryption **Dec**$(priv^{PRE}_k, d') \rightarrow d$ that takes private key and encrypted data and returns the plaintext.

5.2 Two-factor Authentication for Bob

With the original protocol, while Alice securely distributes her data, she can not collaborate with another party (Bob). Bob can neither prove to the servers that he is the person who is authorized to read the data nor can he decrypt the data even if he receives it. Given that Alice herself needs multiple factors to access the stored data, Bob too should at least be challenged by multiple factors as well, thus the protocol should satisfy the requirement of multiple factors for him (Problem 1). Furthermore, the collaboration needs to be approved by Alice (Problem 2) in a way that without her permission, her data shouldn't be accessible to any other parties. And when authorized, Bob should be able to interact with the data not only by decrypting it, but also encrypting his modifications accordingly, so that Alice can decrypt it back as well (Problem 3). A problem that has not been covered by many previous schemes (if not all) is that if Bob removes or overwrites during an update, Alice may lose some of her data (Problem 4). Even if Bob may be trusted not to act in a malicious manner, nevertheless such precautions are important for inadvertent loss or corruption of the data. We address the last issue in Section 5.3 and 5.4. Last, but not least, some collaborative activities may need to support nonrepudiation, which in turn requires that neither Alice nor anyone else should be able to authenticate pretending to be Bob (Problem 5).

We design a triangular protocol where Bob registers to the system through Alice (solution to Problems 1,2). We integrate a PRE scheme into the protocol instead of using a symmetric encryption, where we add another public/private key pair into the Secret Package (*SP*) that Alice and Bob keeps as the possession factor (addressing Problem 3). Note that the encryption password *epp* from the original protocol is the password for the PRE private key. Lastly, by using a different public private key pair for PRE scheme than the signature scheme, we let Bob keep his secret for signature scheme to himself (solution to Problem 5).

In Figure 2 we show how Bob becomes a part of the protocol and claims the read/write privileges. In the following text, the numbers correspond to a sequence of steps in the protocol, and are likewise marked in the accompanying figure. We assume that Bob and Alice have a secure channel to communicate on. First, Bob creates his own authentication tokens (AT), public/private key pair (for the signature scheme) and then Bob registers these with Alice where Bob sends all authentication tokens, signature public key and a secret ($priv^{PRE}_{k\_Bob}$) for the PRE scheme to alice (1). Then, as input, a secret of her own, and Bob's secret, Alice runs the ReKeyGen of the PRE protocol to generate a re-encryption key $RK_{A\leftrightarrow B}$ (2). Alice provides the *ip* list of the servers to Bob (3). Alice also sends Bob's authentication token, Bob's signature public key and re-encryption key to the servers (4). Bob then creates and encrypts his own *SP* locally (6) as shown in Equation 5. Now that the registration is done and Bob is authorized by Alice, he can two-factor authenticate with the servers to download the re-encrypted data.

$$SP_{Bob} = \begin{cases} \text{ips: } ip_0, ip_1, ..., ip_i, ..., ip_{n-1} \\ \text{salts: } salt_0, salt_1, ..., salt_i, ..., salt_{n-1} \\ \text{secret token: } sk \\ \text{signature private/public key: } priv_{k\_Bob}/pub_{k\_Bob} \\ \text{PRE private/public key: } priv^{PRE}_{k\_Bob}/pub^{PRE}_{k\_Bob} \end{cases} \tag{5}$$

For the actual access at this stage, all that Bob needs to do is to choose necessary number of *ip*s from his *SP* to recreate the data and authenticate to them with the two-factor authentication protocol (steps 7,8,9 in Figure 2). Each server can either re-encrypt and store the data (redundant storage, but fast response time when Bob queries) or re-encrypt on the fly when the corresponding data download request arrives (5). Once the verification is done, server returns the re-encrypted data to Bob (9) who then can decrypt it (10) using the new private key $priv^{PRE}_{k\_Bob}$ that he had shared with Alice at the first instance.

5.3 Updates by Bob

A bidirectional encryption scheme allows the updates by Bob to be re-encrypted under the secret of Alice for her later use. On the other hand, she already knows the secret of Bob and can anytime decrypt his updates. We suggest a bidirectional PRE scheme to prevent any confusions and many re-encryptions when there are multiple parties collaborating. When an update by Bob is issued, it can be re-encrypted back and stored.

Another concern is that any data may be overwritten or removed by Bob. An easy way to keep Alice in the loop and let her choose which data to be updated, added or removed is to let her approve each and every update by other parties. But this would be cumbersome and undesirable if Alice is not online/present. An alternative is to apply a persistent versioning mechanism. One way to achieve persistent versioning is to apply a variant of proof of data possession technique that has been proposed recently [15]. With this scheme, both parties can keep track of the integrity of all their own versions.

5.4 Auditable Versioning

This subsection is not a contribution of this paper per say. It provides a summary of [15] and is being included for the sake of self-containedness, to elaborate how one can avoid both intentional or unintentional data corruption while allowing content mutation and collaboration in the proposed setting. We show the enabling data structure for the versioning scheme in Figure 3. Data, divided into parts, is kept at each leaf level node of the data structure where each part is encrypted. When Bob sends an update, the parts that belongs to Alice are not to be touched, yet making use of the persistence algorithms of the data structure, the new data is added to its place. Later if Alice doesn't like the updated data, she can anytime return to her own by simply following the root of the data structure that belongs to her. If the data structure is authenticated [15, 38] then she even can challenge each of the untrusted servers to check integrity of her data at any version. We note that, such an integrity guarantee is a must-have in any case when storing data over untrusted servers, and the storage and performance overheads of such an authenticated data structure are marginal.

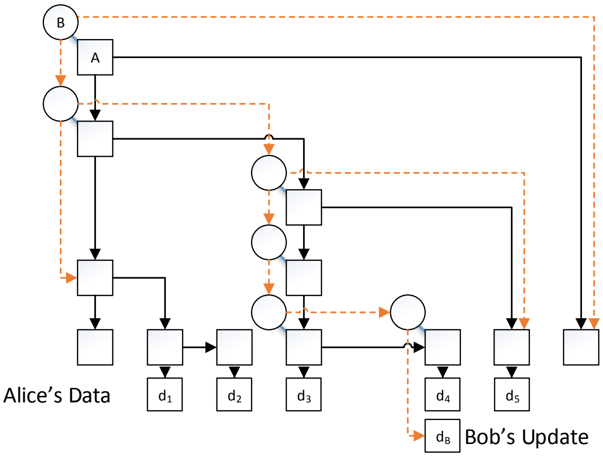

Figure 3: Bob's Update.

6. Performance implication of parameter choices

We have implemented the individual building blocks of the protocol, and they work in tandem. The testing and performance benchmarking have been done on a 64-bit computer with 4 Intel (R) Xeon (R) CPU E5-2640 @ 2.50GHz CPU, 16GB of memory and 16MB of L2 level cache, running Ubuntu 12.04 LTS. We employed openssl[39], cashlib[29], ceph[41] and crypto++[2] libraries and C++ as the programming language for our implementation. All results are the average of 10 runs. Our results include all environmental effects such as input/output times, serialization overheads of objects, random key generation times, etc. We employed RSA with modulus size of 2048 bits and SHA-1 as the hash function for our signature scheme. 256 bit AES (salt enabled) in CBC mode was used for encryption. We generated a 256 bit random string as the secret *sk*. SHA-1 was used also for authentication token hash calculation. For the experiments, we assume the client has a total of 1GB of data to store. We investigate the time and space requirements of the protocol and show the explicit overheads (other than the standard IO times for storage) in Table 1. The first row stands for the simple case where the data owner sends her data to a server without processing it anyhow. Descending in the table, with each row we present time and space requirements at each step of the

	Setup					Retrieve				Update		
	Comm. Cost	Client Process	Server Process	Client Storage	Server Storage (each)	Comm. Cost		Client Process	Server Process	Comm. Cost	Client Process	Server Process
	Client sends					Client sends	Server (each) sends					
Simple (Base case)	1* GB	0 sec	0	0	1* GB	~0 bytes	1* GB	0	0	20´ KB	0	0
Authentication Overhead (OH)	1* GB + 20ª bytes	0.05ª ms	0	0	1* GB + 20ª bytes	20ª bytes	1* GB	0.05ª ms	~0˚ ms	20´ KB + 20ª bytes	0.05ª ms	~0˚ ms
2 fact. Authentication OH	1* GB + 471ª¨ bytes	125^ª ms	0	~2^ KB	1* GB + 471¨ bytes	256º bytes	1* GB	4˜ ms + 12ªº ms	6˚ ms	20´ KB + 256º bytes	12ªº ms	6˚ ms
2 fact. Auth. + Encryption OH	1* GB + 471ª¨ bytes	3.75‘ sec + 125^ª ms	0	~2^ KB	1* GB + 471¨ bytes	256º bytes	1* GB	4˜ ms + 12ªº ms + 2’ sec	6˚ ms	~20´ KB + 256º bytes	~0.5‘ ms + 12ªº ms	6˚ ms
2 fact. Auth. + Encr. + Replication OH	(1* GB + 471ª¨ bytes) x n	3.75‘ sec + 125^ ms + (0.05ª ms x n)	0	~2^ KB	1* GB + 471¨ bytes	256º bytes	(1* GB)/n	4˜ ms + 12ªº ms x n + 2’ sec	6˚ ms	(~20´ KB + 256º bytes) x n	~0.5‘ ms + 12º ms x n	6˚ ms
2 fact. Auth. + Encr. + Erasure Coding OH	1* GB x (n/m) + 471ª¨ bytes x n	10“ sec + 3.75‘ sec + 125^ ms + (0.05ª ms x n)	0	~2^ KB	(1* GB)/m + 471¨ bytes	256º bytes	(1* GB)/m	4˜ ms + 12ªº ms x m + 2’ sec + 7” sec	6˚ ms	(~20´ KB + 256º bytes) x n	1“ ms + 12ªº ms x m	6˚ ms

* Data ª Authentication Token ¨ Public Key ^ Secret Package (SP) º Signature ˚ Verification ´ Update
‘ Data Encryption ’ Data Decryption “ Data Encoding ” Data Decoding ˜ Secret Package Decryption

Table 1: Protocol Overhead Comparison.

protocol for a set of additional security measures.

Setup: The preprocessing time at the client - to generate the possession factor was 125ms. Time to generate individual authentication token per server is essentially the time to compute one hash, and it was 0.044ms. Each server is initially sent a data package of 471 bytes comprising of the user's public key, and the authentication token. That is also the storage footprint at each server for carrying out subsequent authentication verification. The secret package (stored at the client side) which includes information for only one server is of 1926 bytes and it increases by 16 bytes for each additional server used. The public key of 451 bytes can also be stored along with the secret package for portability.

Authentication: When the client sends a request to receive her data, the authentication package generation which consists of the authentication token generation and the signature generation, takes 12ms per server. The size of the authentication package is 268 bytes. Each server takes 6ms for carrying out verification of the authentication package.

Data processing: In terms of computation, the most time consuming part of our protocol is to create the *n* pieces out of the data, to be dispersed across the *n* servers, and subsequently recreating the original data object out of *m* of these *n* dispersed pieces. Note that there will also be network usage and latency issues, which we have left out in the current discussions. We considered the application of both erasure coding as well as Shamir's secret sharing [34]. Specifically, for erasure coding, we applied Reed Solomon Vandermonde implementation from jerasure plugin of ceph library [41]. We used the default 4KB stripe size. The crypto++ [2] library was used to implement Shamir's secret sharing algorithm. The reported times include the actual computation time, but also the time to read/write to the buffer.

Before we discuss the computation time for these two approaches, it is worth discussing qualitatively the implications of the choice of parameters *m* and *n*. The two-factor aspect of approach will work even with $m = 1$, however the additional level of security achieved through dispersal will be missing in that case. For a given choice of *m*, a larger *n* will provide a potential attacker a larger choice of subsets of servers to compromise, and may have a negative effect on security from the perspective of content confidentiality. At the same time, though fault-tolerance was out of the direct scope of this paper, a larger *n* for any particular choice of *m* also provides the system to better mitigate server failures vis-a-vis data availability and integrity. The storage (and communication) overhead increases by a factor of n/m when erasure coding is used, while it increases by a factor of *n* when using Shamir's secret sharing scheme. This also means that for same choice of *m* and *n*, secret sharing uses *m*-folds more storage than erasure coding. Thus, an optimal choice of *m* and *n* is not obvious, and further systematic investigation to determine good choices, likely dependent on the deployment environment, will have to be carried out in future. For the time being, we instead report the performance for a few ad-hoc parameter choices, where we aimed for n/m within 2 (since that will also be the storage overhead if erasure coding is used), while still allowing for a margin of recovery from 1 or 2 faults, while exploring non-trivial *m* and yet keeping *n* small (since the storage overhead worst case scenario is determined by *n* when using secret sharing). Specifically, we chose the following n/m configurations: 3/2, 4/3 and 6/4 to carry out our experiments. Figure 4 shows the time for dispersal and recreation of 1GB of data. In order to also determine the fault-tolerance implications, we simulated one randomly missing piece among the dispersed data for each of our experiment iterations.

Since both Reed Solomon erasure coding and Shamir's secret sharing (indicated with RS and SS respectively in Figure 4) are well studied, the results are on expected lines, but nevertheless, there are several interesting things to note. Erasure coding/decoding is orders of magnitude faster. In fact, the dominant time for erasure coding is really the data read/write operations, and a larger n/m value leads to larger amount of data writes after coding. That's why coding operations for $n/m = 4/3$ is faster than for $n/m = 3/2$ and $n/m = 6/4$. In contrast, for secret sharing, while read/write operations take much more time than erasure coding, the dominant time is in the computation. Decryption however is significantly faster

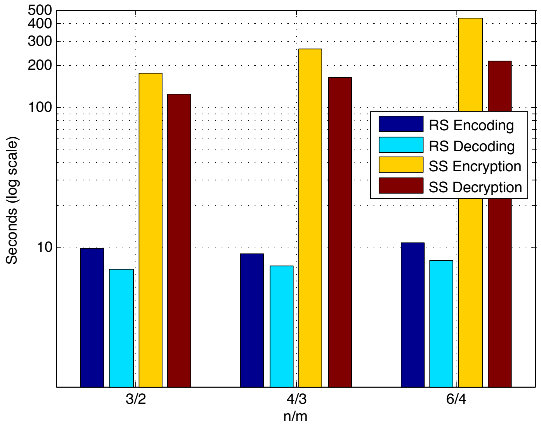

Figure 4: Times spent for Reed Solomon and Shamir's Secret Sharing algorithms.

than encryption. We further note that even though the performance results are reasonably acceptable when using erasure codes, by enabling SIMD parallelization [33] and increasing the stripe size (e.g. 1Mb) the coding can be made even faster. Overall, given the other layers of security in our approach, and the performance penalty that secret sharing imposes, we advocate the use of erasure coding as the preferred mechanism for data dispersal. AES-256 encryption and decryption took respectively 3.75 and 2.02 seconds per GB. Note that due to the redundancy added for the dispersal process, the volumes of data to be encrypted increase, and the time taken for encryption/decryption increase proportionally. The encryption/decryption time overheads are dominated by those of (de-)coding. Though we have not measured the network latency to ship in and out data in our experiments, all the computations cumulated together still is significantly small, and hence, the proposed approach for multi-factor access control should readily extend existing works like [10, 25] without perceptible performance deterioration. Exploring a completely integrated system and benchmarking other system aspects (network usage, energy implications for mobile devices, etc.) are outstanding aspects for future.

7. Conclusions

In this paper we propose a decentralised layered security mechanism for data outsourcing, which protects confidentiality of content from third parties by enforcing a multi-factor access control, which requires both knowledge and possession factors. Security from storage servers is achieved using user controlled encryptions and dispersal (a technique which already exists in the literature). The current protocol is also flexible, in that it allows the user to revoke and create a new possession factor, if the user believes that the possession factor has been compromised. We furthermore demonstrate how the basic protocol can be extended with the use of proxy re-encryption to facilitate collaboration across multiple parties, where each party needs to go through multi-factor access control of the data. A minor limitation of the proposed approach is that one specific user has to act as the owner of the data, and has to approve/facilitate the registration of the other collaborators. A limitation of the general protocol is that a user can revoke older possession factor and reestablish a new possession factor only if he is still also in possession (of a copy of) the possession factor. Because of the decentralised nature of the approach, there is also no mechanism of recovery or changing of either the possession factor or the knowledge factor, if the user loses any of these. Note that the last drawback may be inherent to the degree of security and decentralisation (no trusted third party) aimed at.

Acknowledgments

This research has been partly supported by MoE Tier-2 grant MOE2013-T2-1-068 'eCODE: Erasure Codes for Datacenter Environments', Xvid grant 'Nimbus: A Secure and Reliable Decentralised Storage Platform' and by the Singapore International Graduate Award (SINGA) grant TU-2013-R13145800000000.

References

[1] Boxcryptor https://www.boxcryptor.com/, 2015.

[2] Crypto++ http://www.cryptopp.com/, 2015.

[3] Google. About 2-Step Verification https://support.google.com/accounts/answer/180744?hl=en&ref_topic=1099588, 2015.

[4] The spideroak https://spideroak.com/, 2015.

[5] Truecrypt https://truecrypt.ch/, 2015.

[6] Wuala by lacie https://www.wuala.com/, 2015.

[7] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon. Racs: A case for cloud storage diversity. In *ACM Symposium on Cloud Computing (SOCC)*, 2010.

[8] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9(1):1–30, Feb. 2006.

[9] A. Beimel. Secret-sharing schemes: A survey. *Coding and Cryptography*, 2011.

[10] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. Depsky: Dependable and secure storage in a cloud-of-clouds. *ACM Transactions on Storage (TOS)*, 9(4), 2013.

[11] M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. In *Advances in CryptologyâĂŤEUROCRYPT'98*, pages 127–144. Springer, 1998.

[12] C. Bösch, P. Hartel, W. Jonker, and A. Peter. A survey of provably secure searchable encryption. *ACM Comput. Surv.*, 47(2):18:1–18:51, Aug. 2014.

[13] R. Canetti and S. Hohenberger. Chosen-ciphertext secure proxy re-encryption. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 185–194. ACM, 2007.

[14] CISCO. Fog computing https://techradar.cisco.com/trends/Fog-Computing.

[15] E. Esiner and A. Datta. Auditable versioned data storage outsourcing. *To appear in Future Generation Computer Systems (FGCS)* (Available in http://arxiv.org/abs/1507.08838), 2015.

[16] R. Gracia-Tinedo, M. S. Artigas, and P. G. Lopez. Cloud-as-a-gift: Effectively exploiting personal cloud free accounts via rest apis. In *International Conference on Cloud Computing (IEEE Cloud)*, 2013.

[17] X. Huang, Y. Xiang, A. Chonka, J. Zhou, and R. H. Deng. A generic framework for three-factor authentication: preserving security and privacy in distributed systems. *Parallel and Dis-*

tributed Systems, IEEE Transactions on, 22(8):1390–1397, 2011.

[18] A.-A. Ivan and Y. Dodis. Proxy cryptography revisited. In *NDSS*, 2003.

[19] M. Jakobsson. On quorum controlled asymmetric proxy re-encryption. In *Public Key Cryptography*, pages 112–121. Springer, 1999.

[20] H. Kavalionak and A. Montresor. P2p and cloud: A marriage of convenience for replica management. In *Self-Organizing Systems*, 2012.

[21] G. Kreitz, O. Bodriagov, B. Greschbach, G. Rodriguez-Cano, and S. Buchegger. Passwords in peer-to-peer. In *IEEE P2P*, 2012.

[22] E. A. Lee, J. Rabaey, D. Blaauw, K. Fu, C. Guestrin, B. Hartman, R. Jafari, D. Jones, J. Kubiatowicz, V. Kumar, R. Mangharam, R. Murray, G. Pappas, K. Pister, A. Rowe, A. Sangiovanni-Vincentelli, S. A. Seshia, T. S. Rosing, B. Taskar, J. Wawrzynek, and D. Wessel. The swarm at the edge of the cloud. *IEEE Design & Test*, 31(3), 2014.

[23] B. Libert and D. Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. In *Public Key Cryptography–PKC 2008*, pages 360–379. Springer, 2008.

[24] H.-Y. Lin and W.-G. Tzeng. A secure erasure code-based cloud storage system with secure data forwarding. *Parallel and Distributed Systems, IEEE Transactions on*, 23(6):995–1003, 2012.

[25] H.-Y. Lin and T. W.-G. A secure erasure code-based cloud storage system with secure data forwarding. In *Parallel and Distributed Systems, IEEE Transactions on*, volume 23, 2012.

[26] C. W. Ling and A. Datta. Intercloud raider: A do-it-yourself multi-cloud private data backup system. In *Distributed Computing and Networking (ICDCN)*, 2014.

[27] P. G. Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere. Edge-centric computing: Vision and challenges, 2015.

[28] M. Mambo and E. Okamoto. Proxy cryptosystems: Delegation of the power to decrypt ciphertexts. *IEICE transactions on fundamentals of electronics, Communications and computer sciences*, 80(1):54–63, 1997.

[29] S. Meiklejohn, C. C. Erway, A. Küpçü, T. Hinkle, and A. Lysyanskaya. Zkpdl: A language-based system for efficient zero-knowledge proofs and electronic cash. In *USENIX Security Symposium*, volume 10, pages 193–206, 2010.

[30] D. M'Raihi, S. Machani, M. Pei, and J. Rydell. Totp: Time-based one-time password algorithm. *Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC*, 6238, 2011.

[31] R. Ostrovsky and W. E. Skeith. A survey of single-database private information retrieval: Techniques and applications. *Public Key Cryptography*, 2007.

[32] B. Parno. Bootstrapping trust in a" trusted" platform. In *Hot-Sec*, 2008.

[33] D. A. Patterson and J. L. Hennessy. *Computer organization and design: the hardware/software interface*. Newnes, 2013.

[34] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[35] J. Shao and Z. Cao. Cca-secure proxy re-encryption without pairings. In *Public Key Cryptography–PKC 2009*, pages 357–376. Springer, 2009.

[36] M. Sookhak, H. Talebian, E. Ahmed, A. Gania, and M. K. Khan. A review on remote data auditing in single cloud server: Taxonomy and open issues. *Journal of Network and Computer Applications*, 43, 2014.

[37] M. B. M. Strauss. Atomic proxy cryptography. 1998.

[38] R. Tamassia. Authenticated data structures. In *Algorithms-ESA 2003*, pages 2–5. Springer, 2003.

[39] J. Viega, M. Messier, and P. Chandra. *Network Security with OpenSSL: Cryptography for Secure Communications*. " O'Reilly Media, Inc.", 2002.

[40] L.-H. Vu, K. Aberer, S. Buchegger, and A. Datta. Enabling secure secret sharing in distributed online social networks. In *IEEE Computer Security Applications Conference*, 2009.

[41] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320. USENIX Association, 2006.