ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/329061198

Enforcing Privacy and Security in Public Cloud Storage

Conference Paper · August 2018

DOI: 10.1109/PST.2018.8514195

CITATION
1

READS
305

3 authors:

João Resende
Universidade NOVA de Lisboa
19 PUBLICATIONS **30** CITATIONS
SEE PROFILE

Rolando Silva Martins
University of Porto
43 PUBLICATIONS **266** CITATIONS
SEE PROFILE

Luís Antunes
University of Porto
119 PUBLICATIONS **1,147** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Hyrax - Crowdsourcing Mobile Devices to Develop Edge Clouds View project

Project TRONE: Trustworthy and Resilient Operations in a Network Environment View project

All content following this page was uploaded by João Resende on 13 January 2019.

The user has requested enhancement of the downloaded file.

Enforcing Privacy and Security in Public Cloud Storage

1st João S. Resende
DCC-FCUP
CRACS-INESC TEC
Porto, Portugal
jresende@dcc.fc.up.pt

2nd Rolando Martins
DCC-FCUP
CRACS-INESC TEC
Porto, Portugal
rmartins@dcc.fc.up.pt

3rd Luís Antunes
DCC-FCUP
CRACS-INESC TEC
Porto, Portugal
lfa@dcc.fc.up.pt

***Abstract*—Cloud storage allows users to remotely store their data, giving access anywhere and to anyone with an Internet connection. The accessibility, lack of local data maintenance and absence of local storage hardware are the main advantages of this type of storage. The adoption of this type of storage is being driven by its accessibility. However, one of the main barriers to its widespread adoption is the sovereignty issues originated by lack of trust in storing private and sensitive information in such a medium. Recent attacks to cloud-based storage show that current solutions do not provide adequate levels of security and subsequently fail to protect users' privacy. Usually, users rely solely on the security supplied by the storage providers, which in the presence of a security breach will ultimate lead to data leakage. In this paper, we propose and implement a broker (ARGUS) that acts as a proxy to the existing public cloud infrastructures by performing all the necessary authentication, cryptography and erasure coding. ARGUS uses erasure code as a way to provide efficient redundancy (opposite to standard replication) while adding an extra layer to data protection in which data is broken into fragments, expanded and encoded with redundant data pieces that are stored across a set of different storage providers (public or private). The key characteristics of ARGUS are confidentiality, integrity and availability of data stored in public cloud systems.**

***Index Terms*—Privacy, Security, Cloud Storage, Erasure Code, Broker, IoT**

I. Introduction

Individuals use the cloud to store documents, photographs, and files, so that they are available everywhere. For example, if we are at the office, we can save a file and then access it from home. The cloud acts as a data storage device placed on the Internet, free or paid, accessible from any platform. This type of technology has several advantages, namely: availability on any platform or device; cost savings (users do not need to buy hard drives with large storage capacities); and ease of use in making backups and restoring [2] [3]. Public cloud-based storage services, namely Dropbox, OneDrive and Google Drive, are an inexpensive and scalable way for companies and users alike to create and maintain their files, with a high availability and redundancy against data loss [12]. However, these providers do not offer guarantees on privacy and security over the hosted data besides the trust based on their reputations. Recent attacks [29], [30], show that user data is increasingly seen as a valuable commodity, and subsequently, a desired target for criminals. Given these issues, new mechanisms have to be in place in order to ensure data integrity and confidentiality while using these public storage services.

Our approach addresses a real-world problem related to user trust and empowerment. Nowadays, Cloud Service Providers (CSPs) control all the data in their servers, and the users do not have ownership of the data. Hence, CSPs should take responsibility for the data security [21].

Given the aforementioned limitations and challenges, the following question arises: *Can we have a reliable system that features anonymization by default which enables users to effectively control their personal data?*

In this paper, we will present the ARGUS, a novel privacy brokerage system aiming to enhance confidentiality and availability by partitioning encrypted data over multiple public storage providers, solving the problem of trust in the public cloud providers [4]. Simultaneously, ARGUS offloads computation from the clients in order to make it viable for resource constrained devices, e.g. mobile devices. ARGUS minimises the possibility of data leakage by combining erasure coding and cryptography. In addition, it also adds an extra layer of protection to the user credentials by ciphering them with Intel SGX.

The paper is organised as follows: Section II presents the architecture of the system where ARGUS is tested, and the different components that compose this architecture. In addition, it presents the different modes of operation that are used. Section III describes the implementation of the system. Section IV presents a performance and cost evaluation of the system. Lastly, Section V presents the conclusions of this work.

A. Contributions

Throughout this paper we expand on the following novel contributions:

Privacy Broker: We implement a broker that acts as a proxy to the public cloud infrastructures by performing all the necessary authentication, cryptography and erasure coding. By doing so, it offloads the computational workloads from clients. Our approach ensures confidentiality, integrity and availability of the data in the public cloud systems.

Dual Option File Encryption: In order to ensure the security and privacy of user data, the user can opt to encrypt everything locally and be responsible for the key management. In this way, the user does not need to rely on a third party. In addition, we have the option of delegating the encryption to a third party, which has the benefit of reducing the cost of execution on a limited device.

Confidentiality, Integrity and High-Availability: ARGUS maintains the integrity of the data as it stores an HMAC of all files. The confidentiality of the data is ensured as the data are encrypted and the user can save their private key locally. ARGUS provides high-availability through the redundancy that is assigned in the different cloud providers, that is, information is redundant on the three public clouds.

Intel SGX: The system uses Intel's CPU SGX extensions to cipher user credentials (access tokens that give access to the user's public cloud storage). This is an improvement over current implementations in systems that use the Google Drive API because the credentials are stored locally in the file system.

B. Assumptions

In order to develop the system, it needs the following assumptions:

Availability: ARGUS must be always online to allow the interactions between the clients and the public cloud providers solving the problem of sharing files.

Trusted Third Party

In one of the encryption modes everything is encrypted on the server side. In this case, we have to assume that the server is a trusted third party.

Authentication: is not the main focus of this article. For this reason, and analysing what already exists in the state-of-the-art, we assume an authenticated channel.

C. Related Work

Current cloud systems try to overcome issues regarding the trust in the provider, the limitation of sharing files, or the bottleneck of having a local hybrid cloud. However, there are also new limitations. Users nowadays use edge devices such as mobile or IoT devices to send information to the cloud. The traditional cloud systems do not address the limitations of this system.

	Resilio	*AeroFs*	*Safecloud Photos*	*Boxcryptor*	*Whisp*	*NetApp*	*ARGUS*
Share Files	✓	✓	x	✓	✓	✓	✓
Dropbox API	x	x	✓	✓	✓	✓	✓
OneCloud API	x	x	✓	✓	✓	✓	✓
Google drive API	x	x	✓	✓	✓	X	✓
Private cloud support	x	✓	x	✓	x	✓	✓
Encrypted Public	x	x	x	✓	✓	X	✓
Erasure Coding	x	x	x	x	x	x	✓
ZK encryption	x	x	x	✓	x	x	✓
Collaborative	✓	✓	x	✓	x	✓	✓
Computation offload	x	x	x	x	x	✓	✓
Data Partitioning	x	x	x	x	x	x	✓

TABLE I: Secure Cloud Systems

An outline of the current state-of-the-art systems is depicted in table I. For this comparison we have included *Resilio* [8], *AeroFs* [9] [10], *Safecloud Photos* [5], *Boxcryptor* [6] [24], *Whisp* [7] and *NetApp* .

The literature also describes data transfer some examples that use the NFS (Network File System), or other technologies that allow the users to communicate with the cloud [25][26][27][13][28]. An example can be the implementation of a Shared Cloud-backed File System (*SCFS* [13]), where the user needs to interact with a coordination service. The computation is completed locally with an extension of Depsky [15]. The use of coordination's systems is not the ideal for IoT devices because the computation is intensive and all the tasks are performed by the user device, the coordination service just grants permission to write in the cloud and solves duplication's and detection of collisions. An example of this systems is SCSS [1] and SCFS [13]. Systems such as the FUSE-based file system, backed by Amazon S3 (*S3FS* [19]), typically not solve the limitation of sharing a file with a third party.

In ARGUS, we want to address all the limitations of current implementations, joining efforts of multiple projects in one. Where the users could choose from public or private cloud providers along with the computation offload to guarantee data partitioning and security in the storage providers.

II. Architecture

This section presents our system composed by: a broker system (ARGUS); the public/private cloud providers; the client Android application. An overview of the linkage of the components is shown in Figure 1.

The system takes the main advantages of the broker, using them to give the user control, through an Android application, over the entire content of the data. This system has two confidence modes decided by the user: 1) User Mode: The content is encrypted entirely on the Android device, meaning that the user will have full control of the data and is responsible for the management of the keys; 2) Server Mode: All the content is encrypted on the broker system. However, the keys are shared with ARGUS, so the location and access to ARGUS must be limited.

A. ARGUS

ARGUS can be seen as a broker for a hybrid cloud architecture (B in Figure 1) and is the core of our system. For the client side (A in Figure 1), the broker acts as a proxy to the public cloud infrastructures by performing all the necessary authentication, cryptography and erasure coding. By doing this, it offloads the computational workloads from clients.

Our approach ensures confidentiality, integrity and availability of the data in the public cloud systems. Unlike other approaches, we use multiple cloud providers to ensure both confidentially and availability. For example, an encrypted file can be split and sent to multiple cloud providers (C in Figure 1) which prevents a single provider gaining access to all the necessary data, even if they are somehow able to get the encryption keys. To guarantee the confidentially of the user regarding the access control to the cloud provider, Intel SGX was used to encrypt the access tokens through the use of enclaves, which are protected execution areas.

In order to prevent cloud providers unavailable, we use erasure coding that partitions the data and simultaneously creates parity blocks for redundancy. This ensures that if the necessary parity is present, e.g. any two providers in Figure 1, then we are able to reconstruct the entire data, without user awareness. By using erasure coding, we are making the implicit trade-off between storage space and redundancy/availability.

For data integrity, we store a Hash-based Message Authentication Code (HMAC) [11] that allows the broker to ensure integrity of the stored or transmitted information over an unreliable channel. A new HMAC is produced for each file, before splitting the data, to ensure that the data was not tampered with while being hosted in the public clouds. For complete security against lost data, at least three cloud providers are required, so there is at least one parity block for every two data blocks.

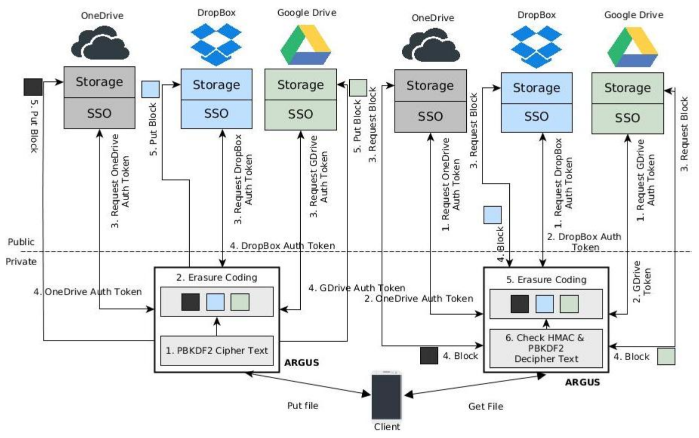

Fig. 1: Upload/Download File

In the left of the Figure 1 depicts the instant when a user wants to upload a file to the cloud, assuming that it has already been authenticated. When the broker receives a request to upload a file, the system must perform two different tasks: understand if the user is sending the file in clear text to cipher or already ciphered (1), and collect the authorisations from the public clouds (3, 4), i.e., the system must get the authentication of the user using OAuth2 or request credentials from Intel SGX. When ARGUS is responsible for encrypting the information, it uses a symmetric key to encrypt the document. The next step is to apply the erasure code algorithm (2) in order to produce the file split into three parts (two data plus one parity chunk). Finally, the system uploads the chunks to the cloud providers (5).

In the right Figure 1 shows the request of a file by a user. First, the system needs permissions to search in the cloud providers for the part of the file that is requested by the user (1, 2). This search can be performed by requested in the list of tokens stored locally (ciphered on Intel SGX) or by performing a request to the cloud provider. Then, the system needs to reconstruct the original encrypted file, and so it applies the erasure code (5) before the system fetches the different chunks of the file (3, 4). In order to recover the original ciphered file, at least two out of three chunks are required. When the system gets two chunks, it will be ready to return to the user the original file if it has the decryption key to decipher, calculate and compare the HMAC (6). If all the previous steps are concluded, the system is ready to send the file to the user. The same happens if ARGUS is not responsible for ciphering the file, in this situation the system compares the HMAC with the original ciphered file.

B. *Client Android Application*

The client (shown as A in Figure 1) is represented by a mobile application that allows the user navigation through the sub-tree that the user defines, similar to public cloud applications such as Dropbox. However, in terms of storage, ARGUS does not resemble Dropbox as it distributes the information across multiple cloud providers.

III. Implementation

In order to implement a prototype that focuses on the security aspects, we decided to skip the implementations with three cloud providers and to only use one, the Google Drive environment, simulating the split of the files in three different folders. For this implementation, we used a Jetty server running in Java that allowed us to implement a direct connection with the Google Drive API.

In order to implement the upload of the file and the split, we decide to use the erasure code provided by the library *liberasurecode* [14]. We used the Jerasure algorithm that gives the possibility of transforming an entire file into two chunks of data and a third parity block that replicates any of the two previous blocks. Regarding the system file manipulation, we have a relational database that stores all HMAC and the owner of the file. When the file is shared with another user, the relational database also stores the encryption key of the file that is ciphered with a public key of each user.

In this first proof of concept, have used Google Drive as a cloud provider rather than using the three different cloud providers needed to run the system respecting our design assumptions. Therefore, we use erasure code to split the information into different files split between different folders to replicate the three cloud providers, because we decided to skip for the implementation of the connection with three different cloud providers. In this scenario, we do not address the upload and download speed between public/private cloud providers.

A. *File Encryption*

Regarding the ciphering process, the user can choose whether to have their data exposed to a third party or not. This has consequences for the upload and download time take longer, as Android devices take longer to process the information either due to resource constraints in the disk access capacity or due to the lower capacity of their processors. To mitigate this problem, we started by implementing the cipher process in a third party, with the following flow: 1) The user generates a public and private key pair on the device;

2) The device sends to ARGUS both keys, to perform the encryption/decryption process of a document.

This encryption process is performed using a 128-bit AES where a random key is generated, this key than is readily encrypted with the public key and stored on the server. This means that the server has access to information; If the user requires the information to be kept confidential, up to ARGUS level, the encryption method will be carried out on the Android device. To do this, the same public and private key pair are used, but only the public key is sent to the server. Here, the encryption is performed through a random key generated to encrypt the documents with AES, and all information (the keys) is stored locally. This means that the user is responsible for all key management, so if the user loses the session keys, the files will be encrypted remotely and will not be retrievable (the user will not be able to edit or read their original format). However, the ability to delete is always available.

B. File Sharing

On public cloud systems, the user is able to share information with others. In order to solve this problem, we use a broker to perform the file sharing between two clients A and B. The private keys of the clients can be stored either locally or remotely in ARGUS, depending of the operation mode decided by the user. Independently of the operation mode, the public keys are always stored in ARGUS.

In the first step, depending on the operation mode, the sender or ARGUS has to gather the original data that the sender wants to share (using erasure code to reconstruct the file). Then, they need to re-encrypt the original data with a new random generated key. This guarantee that the file is ciphered with only this key, and so, there is no need for duplication of files in the cloud - sharing the random generated key allow them to use the same space. In this final step, the sender or ARGUS (depending on the use mode) must encrypt the new random generated key with both public keys. In brief, we have two modes for the file sharing: 1) A user wants to share a file with another user and the sender has to cipher the file being shared. In this case, the sender needs the public key of the receiver that can be obtained from ARGUS without the receiver being online; 2) A user wants to share a file with another user and the ARGUS performs all the ciphering on the file. In this case, the sender does not need the public key of the receiver.

C. Cloud Access Tokens Management

In order to implement the security and privacy features, essential for the design of the model, we also intend to look for the behaviour of the API from Google Drive [16]. With this analysis it was found that the file *StoredCredential* containing the serialised class *StoredCredential* [23] is responsible for the management of the authentication tokens locally. These tokens provide ARGUS the certainly that the authorisation has been granted for a period of time to use the cloud, without the need of additional authentications. An attacker that gets access to this information can launch an attack to steal the information from the cloud.

To solve these issues and taking advantage of the trusted Enclave, we convert the file to incorporate as a persistent storage the authentication tokens on the Intel SGX. The information sealed in Intel SGX guarantees that an attacker that have access to the file will not get the information unsealed without compromising the Intel SGX. This solutions becomes a low-cost and scalable implementation, as well as offered security similar to that of a hardware security module (HSM) [22]. These connections were performed using JNI to interconnect the Google Drive API in Java with C implementations to securely cipher the information inside the Intel SGX. The output is the encrypted data that is stored in a local file.

IV. Experimental Setup and Methodology

The goal for our evaluation is to experimentally demonstrate the overheads associated with cryptographic and erasure coding operations needed to support our privacy preserving solution. Our experimental setup was composed of one Nexus 5X featuring Android 8.0, an Ubiquiti Uap-Ac-Pro Access Point (AP) and an Intel NUC6i7KYK for supporting our backend server. The smartphone was connected to the AP using 802.11n AC with a standard 20 MHz channel width in the 5 GHz wireless frequency range. Each benchmark was evaluated across five runs of different size files: 100 MB, 200 MB and 400 MB. The files were created using random data with the following command: *head -c 100MB* $< /dev/urandom > SIZE$

	File Size		
Services	*100MB*	*200MB*	*400MB*
Upload Google Drive App (s)	107 ± 22	690 ± 75	1020 ± 209
Upload Client Mode (s)	22 ± 5	106 ± 5	232 ± 9
Upload Server Mode (s)	16 ± 1	50 ± 2	70 ± 6
Download Google Drive App (s)	5 ± 1	12 ± 1	21 ± 4
Download Client Mode (s)	22 ± 5	62 ± 11	367 ± 25
Download Server Mode (s)	13 ± 1	27 ± 2	169 ± 27

TABLE II: Download and upload times

We measured both upload and download times across three distinct scenarios, namely, ”Google Drive App”, ”Client mode” and ”Server mode”. The results are shown in Tables II. The Google Drive evaluations, dubbed ”Google Drive App”, were performed using *androidquickstart*[17] that implements the connection with Google Drive using the official API [18]. Both ”Client mode” and ”Server mode” denote the two modes of operation supported by ARGUS. The first uses solely the resources available in the smartphone to perform the cryptographic and erasure coding operations, while the latter offloads those same operations to our backend server.

As expected, performing encryption in the local device is time consuming, and this can be seen in Table II. Increasing the file size from 100 MB to 400 MB leads to an increase in the upload times of roughly one order of magnitude across the three scenarios. Our solutions achieve a 5-fold speedup when using the ”Client Mode”, and 14-fold when using ”Server Mode”, with the difference between these two being easily explained by the difference in computational power between the smartphone and the server. However, the significant improvements over the ”Google Drive App” are explained with our multi-channel approach which mimics mechanisms used in modern computer architectures, namely in memory and storage subsystems, where multiple channels are aggregated in order to improve throughput.

In terms of downloading, both in ”Client mode” and ”Server mode”, our solution exhibits slower download times compared to ”Google Drive App”. Besides the lack of optimisation in the current prototype, our solution incurs the overhead associated with the cryptographic and FEC operations. After some debugging, we found that our use of very large blocks for the cryptographic and FEC operations led to a significant overhead on the memory subsystem, namely, due to page faults. In the worst case, our system is roughly 17 times slower while downloading when compared to ”Google Drive App”.

V. Conclusion

In this work, we presented a novel architecture that enhances the current state-of-the-art by improving data confidentiality and availability over small footprint devices such as IoT systems. We make use of the erasure code and cryptography techniques over multiple storage providers in order to provide a trustworthy and privacy-preserving system. Moreover, it increases the difficulty for attackers to access the entire ciphered file by partitioning it across multiple cloud providers. Lastly, ARGUS offloads computation from the client as a way to support resource constrained devices, such as mobile devices.

In our system, the user can control all their data. For this, the option of locally storing the encryption keys allows, at any time, the possibility for the user to render the stored data obsolete just by uninstalling the application or removing the private keys because the files are encrypted with the encryption keys, and without them, the files are just random data. We compared the results of our system with Google Drive, and although we did not count on optimisation issues, our system provides better results when compared to Google Drive in terms of the upload time.

Acknowledgements

This work is partially funded by the ERDF through the COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, and by National Funds through the FCT as part of project UID/EEA/50014/2013. The work of João S. Resende was supported by a scholarship from the Fundação para a Ciência e Tecnologia (FCT), Portugal (scholarship number PD/BD/128149/2016). The work of Luís Antunes was supported by Project ”NanoSTIMA and Analytics/NORTE-01-0145-FEDER-000016”, financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF).

References

[1] Wang, Wenfeng, et al. ”An enhanced erasure code-based security mechanism for cloud storage.” Mathematical Problems in Engineering 2014 (2014).
[2] Ma, Sugang. ”A Review on Cloud Computing Development.” JNW 7.2 (2012): 305-310.
[3] Misra, Sudip, et al. ”Learning automata-based QoS framework for cloud IaaS.” IEEE Transactions on Network and Service Management 11.1 (2014): 15-24.
[4] Kang, Baoyuan, Jiaqiang Wang, and Dongyang Shao. ”Attack on Privacy-Preserving Public Auditing Schemes for Cloud Storage.” Mathematical Problems in Engineering 2017 (2017).
[5] Safe Cloud photos, https://yoursafecloud.com/ [visited 02/22/2017]
[6] Boxcryptor, Encryption software to secure cloud files , [Online; Accessed https://www.boxcryptor.com [Online; Accessed 02/22/2017]
[7] Whisp, Secure and easy file transfer [Online; Accessed https://whisp.ly [Online; Accessed 1 May 2018]
[8] Resilio, All Your Data, Across All Your Devices, [Online; Accessed https://www.resilio.com/individuals/ [Online; Accessed 1 May 2018]
[9] Aerofs, File Sync & Share on Your Infrastructure, [Online; Accessed https//www.aerofs.com/ [Online; Accessed 1 May 2018]
[10] Durao, Frederico, et al. ”Usto. re: A private cloud storage software system.” International Conference on Web Engineering. Springer, Berlin, Heidelberg, 2013.
[11] Krawczyk, Hugo, et al. ”HMAC: Keyed-hashing for message authentication.”, 1997.
[12] Armbrust, Michael, et al. ”A view of cloud computing.” Communications of the ACM 53.4 (2010): 50-58.
[13] Bessani, Alysson Neves, et al. ”SCFS: A Shared Cloud-backed File System.” USENIX Annual Technical Conference. 2014.
[14] OpenStack, liberasurecode - Erasure Code API , [Online; Accessed https://github.com/openstack/liberasurecode [Online; Accessed 1 May 2018]
[15] Bessani, Alysson, et al. ”DepSky: dependable and secure storage in a cloud-of-clouds.” ACM Transactions on Storage (TOS) 9.4 (2013): 12.
[16] Google, google-api-java-client-samples, [Online; Accessed https://github.com/google/google-api-java-client-samples/ [Online; Accessed 1 May 2018]
[17] Google Drive Android Quickstart, [Online; Accessed https://github.com/googledrive/android-quickstart [Online; Accessed 1 May 2018]
[18] Google Drive APIs [Online; Accessed https://developers.google.com/drive [Online; Accessed 15 February 2018]
[19] s3fs-fuse [Online; Accessed https://github.com/s3fs-fuse/s3fs-fuse [Online; Accessed 1 May 2018]
[20] NetApp Cloud Central [Online; Accessed https://cloud.netapp.com [Online; Accessed 1 May 2018]
[21] Liu, Keyang, et al. ”A Cloud-User Protocol Based on Ciphertext Watermarking Technology.” Security and Communication Networks 2017 (2017).
[22] Chakrabarti, Somnath, et al. ”Intel SGX Enabled Key Manager Service with OpenStack Barbican.” arXiv preprint arXiv:1712.07694 (2017).
[23] Class StoredCredential (2015) - [Online; Accessed https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/StoredCredential [Online; Accessed 6 February 2018]
[24] Latha, S., K. Raju, and S. Santhi. ”Overview of dropbox encryption in cloud computing.” Transactions on Engineering and Sciences 2.3 (2014): 27-32.
[25] Vrable, Michael, Stefan Savage, and Geoffrey M. Voelker. ”BlueSky: A cloud-backed file system for the enterprise.” Proceedings of the 10th USENIX conference on File and Storage Technologies. USENIX Association, 2012.
[26] Han, Seungyeop, et al. ”MetaSync: File Synchronization Across Multiple Untrusted Storage Services.” USENIX Annual Technical Conference. 2015.
[27] Kotla, Ramakrishna, Lorenzo Alvisi, and Mike Dahlin. ”SafeStore: A durable and practical storage system.” USENIX Annual Technical Conference. 2007.
[28] Shen, Zhiming, et al. ”Supercloud: A Library Cloud for Exploiting Cloud Diversity.” ACM Transactions on Computer Systems (TOCS) 35.2 (2017): 6.
[29] Grobauer, Bernd, et al. ”Understanding cloud computing vulnerabilities.” IEEE Security & Privacy 9.2 (2011):50-57.
[30] Subashini, S., and Veeraruna Kavitha. ”A survey on security issues in service delivery models of cloud computing.” Journal of network and computer applications 34.1 (2011): 1-11.

View publication stats