TOWARDS SECURE CLOUD STORAGE SERVICES

A Dissertation
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Mazhar Ali

In Partial Fulfillment of the Requirements
for the Degree of
DOCTOR OF PHILOSOPHY

Major Department:
Electrical and Computer Engineering

January 2015

Fargo, North Dakota

North Dakota State University
Graduate School

Title

Towards Secure Cloud Storage Services

By

Mazhar Ali

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

DOCTOR OF PHILOSOPHY

SUPERVISORY COMMITTEE:

Samee U. Khan
Chair

Jacob S. Glower

Sudarshan K. Srinivasan

Ying Huang

Approved:

01/26/2015	Scott C. Smith
Date	Department Chair

ABSTRACT

Cloud computing is anticipated to revolutionize the Information and Communication Technology sector and has been a mainstream of research over the last decade. The cloud computing, upsurges the capabilities of the hardware resources by optimal and shared utilization. The above mentioned features encourage the organizations and individual users to shift their data, applications and services to the cloud. However, the services provided by third-party cloud service providers entail additional security threats.

Data being one of the prime assets of the organizations must be protected from all sorts of security threats. The data in the cloud is much more vulnerable to risks in terms of confidentiality, integrity, and availability in comparison to the conventional computing model. The ever increasing number of users and applications leads to enhanced security risks. Violation of integrity may also result from multi-tenant nature of the cloud. Employee of SaaS providers, having access to information may also act as a potential risk.

Considering the paramount importance of data security in the cloud environment, we propose methodologies towards the secure cloud storage services. We propose methodologies to secure: (a) Single user data, (b) Group shared data, and (c) approach security and performance collectively. We propose Data Security for Cloud Environment with Semi-trusted third party (DaSCE) protocol, a cloud storage security system that provide key management, access control, and file assured deletion. Parts of keys are stored at semi-trusted servers called key managers. The key management is accomplished using (k, n) threshold secret sharing mechanism. Finally, we present the DROPS methodology that collectively deals with the security and performance in terms of retrieval time. The data file is fragmented and the fragments are dispersed over multiple nodes. The nodes are separated by means of T-coloring. The fragmentation and dispersal ensures that no significant information is obtainable by an adversary in case of a successful attack.

ACKNOWLEDGEMENTS

First and foremost thanks to ALLMIGHTY ALLAH Who has helped me throughout the course of my studies. All of my knowledge, strength, courage, health, and abilities are His blessings upon me and there is no way to fulfill His right to thank Him.

Special thanks to Dr. Samee U. Khan, my advisor, for his help, guidance and innovative ideas. I offer my sincere and deep hearted gratitude to my advisor who always encouraged me, and persistently conveyed the spirit and guidance required for the research. Without his kind guidance and continuous efforts, this disquisition would not have been possible.

Special thanks to my committee members, Dr. Jacob S. Glower, Dr. Sudarshan K. Srinivasan, and Dr. Ying Huang for their support, guidance and helpful recommendations. Thanks to the Electrical and Computer Engineering staff members Jeffrey Erickson, Laura D. Dallman, and Priscilla Schlenker for all the unconditional help and favor.

I would like to thank my family. Their continuous support is always a source of motivation and encouragement for me. I especially like to thank my father and mother, who are the only and every reason for whatever I am today and whatever I achieved in my life. I also would like to thank my loving wife and my son Abdullah, for their patience, time, and support. They both have shown extreme patience and cooperation that resulted in successful completion of this disquisition.

Finally, I owe my heartiest thanks to all my friends and colleagues here in the US and Pakistan, who always helped me in the time of need.

DEDICATION

I would like to dedicate this thesis to my family, especially to my parents, my wife, and my son for all the inexplicable love, support, and motivation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1. Cloud Computing and Data Security

Cloud computing has emerged as a promising computing paradigm and has shown tremendous potential in managing the hardware and software resources located at third-party service providers. On-demand access to the computing resources in a pay-as-you-go manner relieves the customers from building and maintaining complex infrastructures [1.1, 1.2]. Cloud computing presents every computing component as a utility, such as software, platform, and infrastructure. The economy of infrastructure, maintenance, and flexibility makes cloud computing attractive for organizations and individual customers [1.3]. Despite benefits, cloud computing faces certain challenges and issues that hinder widespread adoption of cloud. For instance, security, performance, and quality are a few to mention [1.4, 1.5].

The development and operation of data storage sites is ongoing process in organizations. Off-site data storage is a cloud application that liberates the customer from focusing on data storage systems [1.4]. Representing system characteristics and capabilities as utility, causes the user to focus on aspects directly related to data (security, transmission, processing) [1.6, 1.7]. However, moving data to the cloud, administered and operated by certain vendor requires high level of trust and security. Multiple users, separated through logical barriers of virtual machines, share resources including storage space. Multi-tenancy and virtualization generate risks and underpins the confidence of users to adopt the cloud model [1.8, 1.9]. Armbrust *et al.* [1.1] ranked data confidentiality and auditing at number three in the list of top ten obstacles impeding widespread cloud adoption. Data can be used by the cloud service providers without authorization [1.8, 1.10, 1.11] and can be accessed by other machines in cloud [1.10, 1.9].

Moreover, the loss of control over data and computation raises many security concerns for the organizations that thwart the wide adoptability of the public cloud.

Data being the principal asset for organizations needs to be secured especially, when data must enter a public cloud. To avoid unauthorized access to the cloud data, access control mechanism must be enforced [1.12, 1.13]. Moreover, data leakage and data privacy strategies must be employed so that only authorized users can access and utilize data. Refraining cloud service providers from utilizing the customer data requires high preventive measures [1.9]. Encryption techniques provide a solution to ensure privacy and confidentiality of stored data. However, key management becomes a prime issue in the case of encryption [1.14, 1.15]. Cryptographic keys need to be stored and protected. Compromise or failure of a key storage facility may lead to the loss of data. Therefore, cryptographic keys must be stored in a robust manner and a single point of failure should not affect the availability of data [1.15].

The data is usually encrypted before storing on to the cloud. The access control, key management, encryption, and decryption processes are handled by the customers to ensure the data security [1.16]. However, when data is to be shared among the group, the cryptographic services need to be flexible enough to handle different users, exercise the access control, and manage the keys in an effective manner to safeguard the data confidentiality [1.17]. The data handling among the group has certain additional characteristics as opposed to a two-party communication or the data handling belonging to the single user. The existing, departing, and newly joining group members can prove to be an insider threat violating the data confidentiality and privacy [1.17]. The insider threats can prove to be more devastating due to the fact that they are generally launched by the trusted entities. Due to the fact that people trust the insider entities, the research community focuses more on outsider attackers. Nevertheless, multiple security issues can arise due to different users in the group. We discuss some of the issues in the following discussion.

A single key shared between entire group members will result in the access of the past data to the newly joining member. The aforesaid situation violates the confidentiality and the principle of least privilege [1.18]. Likewise, a departing member can access the future communication. Therefore, in a group shared data the inside members might generate the issue of backward access control (new user accessing past data) and forward access control (departing user access future data) [1.18]. The simple solution of rekeying (generating new key, decrypting all the data and re-encrypting with the new key) does not prove to be scalable for frequent changes in group membership [1.17].

A separate key for every user is a cumbersome solution. The data must be encrypted separately for every user in such a scenario. The changes in the data require the decryption of all of the copies of the users and encryption again with the modified contents [1.17]. Therefore, such a methodology is required that encrypts the data only once while granting access to all of the legitimate users.

The existing and legitimate group members might show illegitimate behavior to manipulate the data [1.2]. The presence of the entire symmetric key with the user allows malicious user to turn into an insider threat [1.2]. The data can be decrypted, modified, and re-encrypted by the malicious insider within the group. Consequently, the legitimate user in the group may access certain unauthorized files within the group [1.19]. On the other hand, it is necessary for the user to possess a key to conduct various operations on the data. The possession of the key also implicitly proves the legitimacy of the user to operate on the data [1.19].

Nevertheless, simultaneously dealing with both the issues related to the key is an important issue that needs to be addressed effectively.

A cloud consists of numerous entities. For a cloud to be secure, all of the participating entities must be secure. In any given system with multiple units, the highest level of the system's security is equal to the security level of the weakest entity [1.12]. The aforementioned fact can bring the security level of other entities down to the level of the victim entity. The weakened security of the victim entity becomes the gateway for an attacker to enter the system that, in turn, puts the whole system and resources at risk. Therefore, in a cloud, the security of the assets does not solely depend on an individual's security measures [1.20]. The neighboring entities may provide an opportunity to an attacker to bypass the user's defenses.

The off-site data storage requires users to move data in cloud's virtualized and shared environment that may result in various security concerns. Pooling and elasticity of a cloud, allows the physical resources to be shared among many users [1.21]. Moreover, the shared resources may be reassigned to other users at some instance of time that may result in data compromise through data recovery methodologies [1.21]. Furthermore, a multi-tenant virtualized environment may result in a VM to escape the bounds of virtual machine monitor (VMM). The escaped VM can interfere with other VMs to have access to unauthorized data [1.22]. Similarly, cross-tenant virtualized network access may also compromise data privacy and integrity. Improper media sanitization can also leak customer's private data [1.20].

A cloud must ensure throughput, reliability, and security [1.23]. A key factor determining the throughput of a cloud that stores data is the data retrieval time [1.24]. The cloud provider ensures such a performance through Service Level Agreements (SLA). In large-scale systems, the problems of data reliability, data availability, and response time are dealt with data replication strategies [1.24]. However, placing replicas data over a number of nodes increases the attack surface for that particular data. For instance, storing m replicas of a file in a cloud instead of one replica increases the probability of a node holding file to be chosen as attack victim, from $\frac{1}{n}$ to $\frac{m}{n}$, where n is the total number of nodes. The higher the value of m, the higher the probability that the node holding the data file will come under attack. Therefore, both security and performance are critical for the next generation large-scale systems, such as clouds.

1.2. Motivation

Cloud is anticipated to be the next major paradigm shift in the ICT sector (ICT). Today, contemporary society relies more than ever on the Internet and cloud computing. According to a Gartner report published in January 2013, overall public cloud services are anticipated to grow by 18.4 percent in 2014 into a $155 billion market [1.25]. Moreover, the total market is expected to grow from $93 billion in 2011 to $210 billion in 2017. Fig. 1.1 depicts the public cloud market size since 2009 and prediction for year 2017. We've seen cloud computing adopted and used in almost every domain of human life, such as business, research, scientific applications, healthcare, and e-commerce [1.26].

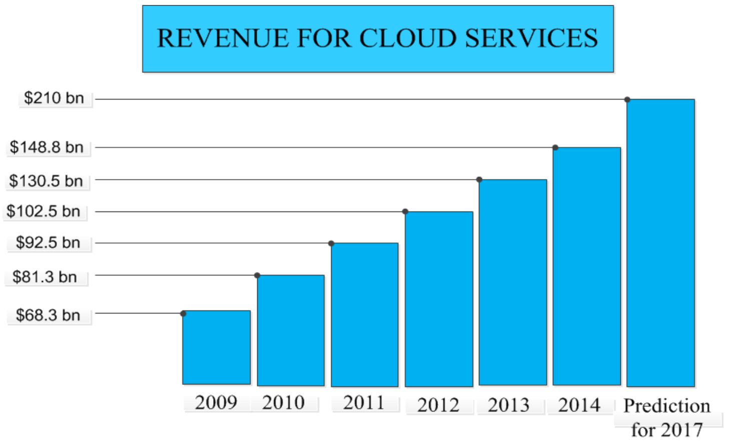


Fig. 1.1. Market size of public clouds.

The advent and rapid adoption of the cloud paradigm has brought about numerous challenges to the research community and cloud providers, however. One of the major issues posed by the cloud computing is security in general and data security in particular. According to a survey conducted by the Microsoft Corporation 58% of general population, 86% of senior business leaders are excited about cloud computing potential. However, 90% of the same are concerned about the data security. The Cloud Security Alliance (CSA) in one of its survey states that 80% of the users of cloud computing are concerned about the data security aspect. Moreover, the ref. [1.1] ranks the data security amongst the top ten obstacles for cloud computing.

The aforementioned discussion ratifies the apparent need and impetus for the data security in cloud computing. Thus, the research that targets the development of data security methods specialized for cloud computing paradigm is crucial for the future sustainability of large, medium, and small cloud users across the globe. As a matter of fact, this kind of research must continuously push the limits of what is possible in computing, and indeed in this case, is leading edge in the development of data security methodologies.

1.3. Research Goals and Objectives

The objective of our research is to develop methodologies for securing data in the cloud computing domain. Compared to the traditional ICT infrastructure, the cloud computing paradigm exhibits different characteristics and baseline technologies, such as multi-tenancy, virtualization, elasticity, resource pooling. The aforesaid characteristics demand for such methodologies that are able to cope with security requirement originated due to the core technologies used to provide aforesaid characteristics. Moreover, the data residing in the cloud is mainly one of the two types, (a) single user data and (b) group shared data. Both of the aforementioned data categories need different security methodologies due to different operational requirements. Furthermore, as discussed in Section 1.1, security and performance are critical for the next generation large-scale system, such as cloud. Consequently, the methodology must approach security and performance collectively. Based on the aforesaid discussion, following are the specific objectives of our research work.

- To propose a data security scheme for a single user data in cloud.
- To propose a data security methodology for group shared data in cloud.
- To propose a technique that collectively approaches the security and performance.

1.4. References

[1.1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Ktaz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoics, and M. Zaharia, "A View of Cloud Computing," Communications of the ACM, Vol. 53, No. 4, 2010, pp. 50-58.

[1.2] A. N. Khan, M. L. Mat Kiah, S. U. Khan, and S. A. Madani, "Towards secure mobile cloud computing: a survey," Future Generation Computer Systems, Vol. 29, No. 5, 2013, pp. 1278-1299.

[1.3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," Future Generation computer systems, Vol. 25, No. 6, 2009, pp. 599-616.

[1.4] T. Dillon, C. Wu, and E. Chang, "Cloud Computing: Issues and Challenges," In proceedings of 24th International Conference on Advanced Information Networking and Applications, pp. 27-33, 2010.

[1.5] D. Sun, G. Chang, L. Sun, and X. Wang, “Surveying and Analyzing Security, Privacy and Trust Issues in Cloud Computing Environments,” Procedia Engineering, Vol. 15, 2011, pp. 2852 – 2856.

[1.6] Cloud Security Alliance, https://downloads.cloudsecurityalliance.org/initiatives/cdg/CSA_CCAQIS_Survey.pdf (accessed March 24, 2013).

[1.7] A. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, “A survey of mobile cloud computing application models,” IEEE Communications Surveys and Tutorials, 2013, 1-21.

[1.8] M. S. Blumenthal, “Is Security Lost in the Clouds?” Communications and Strategies, No. 81, 2011, pp. 69-86.

[1.9] C.Cachinand M.Schunter, "A cloud you can trust," IEEE Spectrum, Vol. 48, No. 12, 2011,pp. 28-51.

[1.10] M. Mowbray, and S. Pearson, "A client-based privacy manager for cloud computing," In Proceedings of the Fourth International (ICST) Conference on COMmunication System softWAre and middleware, ACM, p. 5, 2009.

[1.11] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, "Controlling data in the cloud: outsourcing computation without outsourcing control," In Proceedings of the ACM workshop on Cloud computing security,pp. 85-90, 2009.

[1.12] M. Kaufman,”Data security in the world of cloud computing,” IEEE Security and Privacy, Vol. 7, No. 4, 2009, pp. 61-64.

[1.13] K. M. Khan, and Q. Malluhi, "Establishing trust in cloud computing," IT professional, Vol. 12, No. 5, 2010,pp. 20-27.

[1.14] H. Takabi, J. B. D. Joshi, and G. J. Ahn, "Security and privacy challenges in cloud computing environments," IEEE Security and Privacy, Vol. 8, No. 6, 2010, pp. 24-31.

[1.15] W. Jansen and T. Grance, "Guidelines on security and privacy in public cloud computing," NIST special publication, 800-144, 2011.

[1.16] D. Chen, X. Li, L. Wang, S. U. Khan, J. Wang, K. Zeng, C. Cai, "Fast and Scalable Multi-way Analysis of Massive Neural Data", IEEE Transactions on Computers, Vol. 63, DOI:10.1109/TC.2013.2295806, 2014.

[1.17] A. N. Khan, M. M. Kiah, S. A. Madani, M. Ali, and S. Shamshirband, "Incremental proxy re-encryption scheme for mobile cloud computing environment," The Journal of Supercomputing, Vol. 68, No. 2, 2014, pp. 624-651.

[1.18] Y. Chen and W. Tzeng, "Efficient and Provably-Secure Group Key Management Scheme Using Key Derivation," In IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2012, pp. 295-302.

[1.19] L. Xu, X. Wu, and X. Zhang, "CL-PRE: a certificateless proxy re-encryption scheme for secure data sharing with public cloud," In Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, 2012, pp. 87-88.

[1.20] B. Grobauer, T.Walloschek, and E. Stocker, "Understanding cloud computing vulnerabilities," IEEE Security and Privacy, Vol. 9, No. 2, 2011, pp. 50-57.

[1.21] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," Future Generation Computer Systems, Vol. 28, No. 3, 2012, pp. 583-592.

[1.22] W. A. Jansen, "Cloud hooks: Security and privacy issues in cloud computing," In 44th Hawaii IEEE International Conference on System Sciences (HICSS), 2011, pp. 1-10.

[1.23] A. N. Khan, M.L. M. Kiah, S. A. Madani, and M. Ali, “Enhanced dynamic credential generation scheme for protection of user identity in mobile-cloud computing, The Journal of Supercomputing, Vol. 66, No. 3, 2013, pp. 1687-1706 .

[1.24] M. Tu, P. Li, Q. Ma, I-L. Yen, and F. B. Bastani, “On the optimal placement of secure data objects over Internet,” In Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, pp. 14-14, 2005.

[1.25] Gartner, “Forecast Overview: Public Cloud Services, Worldwide,” 2011–2016, 4Q12 Update, 2013.

[1.26] K. Bilal, S. U. R. Malik, O. Khalid, A. Hameed, E. Alvarez, V. Wijaysekara, R. Irfan *et al*. "A taxonomy and survey on green data center networks." Future Generation Computer Systems, Vol. 36, 2013, pp. 189-208.

2. RELATED WORK

In this chapter we discuss some of the work that is related to the research we have performed during Ph.D.

2.1. Securing Single User Data

Juels *et al.* [2.1] presented a technique to secure the cloud data that provides a number of services, such as integrity, freshness, and availability. The authors employed a gateway application in the enterprise to manage the integrity and freshness checks for the data. The Iris file system is designed to migrate organization's internal file system to the cloud. Moreover, a Merkle tree is used by gateway, which ensures freshness and integrity of data by inserting file blocks, MAC codes, and file version numbers at different levels of the tree. The gateway application also manages the cryptographic keys for confidentiality requirements. Moreover, Ref. [2.1] proposed an auditing framework that audits the cloud environment for ensuring the freshness of the data, data retrievability, and resilience against disk failures. However, the technique heavily depends on the user's employed scheme for data confidentiality. Moreover, data cannot be protected against service provider wholesale.

In [2.2], the authors presented a cryptographic file system that provides confidentiality and integrity services to the outsourced data. The authors used hash based MAC tree for providing the aforesaid services. Block-wise encryption is used for the construction of a MAC tree. The file system at the client side interacts with the file system of the server and outsources the encrypted blocks. Encrypted file blocks and cryptographic metadata are stored separately. Nevertheless, the presence of cryptographic metadata on the storage side can be a potential threat.

The authors in [2.3] proposed a virtual private cryptographic storage service to provide confidentiality and integrity to user data within the cloud. The client application in the proposed method has three modules: (a) data processor, (b) data verifier, and (c) token generator. The client application generates a master key to be used for subsequent operations. The data processor encrypts the file to be uploaded with keys generated from the master key and uploads to the cloud. The data download involves the use of token generator that generates a token for the user to download data. Token also contains identity of files to be downloaded. The data verifier checks for the integrity of the data once the data is downloaded from the cloud. Attribute Based Encryption (ABE) is used for encryption. However, the key in [2.3] resides at client side and may be subject to a single point of failure.

The use of a trusted third party for providing security services in the cloud is advocated in [2.4]. The authors used the public key infrastructure (PKI) to enhance the level of trust in the authentication, integrity, and confidentiality of data and the communication between the involved parties. The keys are generated and managed by the certification authorities. At the user level, the use of temper proof devices, such as smart cards was proposed for the storage of the keys. Similarly, Tang *et. al*. have utilized the public key cryptography and trusted third party for providing data security in cloud environments [2.5]. However, the authors in [2.5] have not used the PKI infrastructure to reduce the overheads. The trusted third party is responsible for the generation and management of public/private keys. The trusted third party may be a single server or multiple servers. The symmetric keys are protected by combining the public key cryptography and the (k, n) threshold secret sharing schemes. Nevertheless, such schemes do not protect the data files against tempering and loss due to issues arising from virtualization and multi-tenancy.

The authors in [2.6] approached the virtualized and multi-tenancy related issues in the cloud storage by utilizing the consolidated storage and native access control. The Dike authorization architecture is proposed that combines the native access control and the tenant name space isolation. The proposed system is designed and works for object based file systems. However, the leakage of critical information in case of improper sanitization and malicious VM is not handled.

Wei *et al*. [2.7] presented SecCloud, a storage security protocol that not only secures the user data uploaded into the cloud but also secures the computations performed on the user data. The SecCloud uses encryption for achieving the storage security. The bilinear pairing (with cyclic additive and multiplicative groups) is used to generate keys for the user, cloud, and a trusted third party. The user gets the storage space from the CSP to store data. The data (divided into m number of messages) is signed by the trusted third party (called the verification agency). The data along with the verifiable signatures is sent to the cloud by encrypting with the session key. The session key is calculated through Bilinear Deffie-Hellman both by the user and the cloud. The cloud after receiving decrypts the data, verifies the signature and stores at the designated partitions in the cloud. The computational security is ensured against partial computation and use of invalid data to save computational cost. It also verifies that data is stored at the correct partitions in the cloud. For the computation security the SecCloud utilizes Merkle hash tree. The computational results are verified by the verifying agency by rebuilding the Merkle tree. To reduce the computational redundancy, the verifier does not build the whole tree but uses probabilistic sampling.

The author in [2.8] used a combination of established and specialized procedures besides additional proposed steps to secure the data in the cloud. The proposed scheme allows the user to rate the requirement of confidentiality, availability, and integrity between values of one to ten (1 - 10). The values are used to determine Sensitivity Rating (SR) of the user data. Based on the SR value, the data is allotted space in one of the three proposed partitions in the cloud. The proposed partitions are public, private, and limited access partitions. The SR value above eight assigns data to limited access partition and below value three to public partition. The data is encrypted with 128-bit SSL encryption and MAC is appended afterwards. An index is also prepared and encrypted to employ searching capabilities over encrypted data. The data and index are sent to the cloud where they are stored depending on the SR value. The download is allowed based on user authentication that is carried out cooperatively by data owner and the cloud. The data in the public partition needs no authentication. The data is transmitted over SSL in both the directions.

2.2. Securing Group Shared Data

The encryption of data before outsourcing to the cloud ensures the privacy of the data but poses certain restriction. The restrictions are specific to the situations where data is to be shared among the group and/or requires forwarding. Such an environment is accompanied with frequent user revocations that require the re-encryption of data with changed keys for avoiding data leakage to the revoked user. Liu *et al*. [2.9] proposed a time based proxy re-encryption combined with Attribute Based Encryption (ABE) to support secure data sharing in group along with the fine grained access control. The proposed scheme (TimePRE) ensures that data is securely forwarded to the group users and deals with the user revocation. Unlike other proxy re-encryption schemes, the TimePRE does not require the data owner to be online for user revocation and generation of new re-encryption keys. The TimePRE associates the time period with every user and upon expiration of the time period the user is automatically revoked by the Cloud Service Provider (CSP). A pre-shared master key between the data owner and the CSP allows the CSP to generate the re-encryption keys. The access control is ensured by use of ABE that identifies user by set of attributes rather than identity. The ABE in TimePRE uses eligible time periods for a user along with other attributes to identify a user. The proposed scheme ensures privacy and availability of the data within the group. However, it does not focus on the data integrity.

A cloud storage system based on secure erasure code is presented in [2.10]. The system uses threshold key servers for storing a user’s key generated by a system manager. User encrypts the data divided into blocks and stores every block on randomly selected multiple servers. The system also provides the functionality of data forwarding by allowing any of the users to forward the data to any other users without downloading. The authors used proxy re-encryption method for forwarding the encrypted data. A similar scheme is presented by the same authors in [2.11] with the difference that the later does not provide data forwarding. However, aforesaid schemes require heavy implementation level changes on the cloud side.

To ensure the quality of the cloud storage, integrity and availability of data in the cloud, authors in [2.12] proposed effectual methodology that supports on-demand data correctness verification. The proposed methodology conducts the verification of the cloud data correctness without explicit knowledge of the whole data. The erasure correcting code and homomorphic tokens are used for the aforesaid purpose. The homomorphic token are pre-computed by the user and data is fragmented and stored redundantly across the cloud servers. To verify data correctness, a challenge containing random data blocks indices is transmitted to the cloud. The cloud computes the response and sends back to the user where decision is made based on the comparison of received result with the pre-computed tokens. Additionally, the proposed scheme performs error localization by detecting the misbehaving server. Moreover, insertion, deletion, modification, and appending of data blocks are supported in the proposed scheme. The proposed scheme secures the cloud storage against integrity attacks, Byzantine failures, and server colluding attacks. The authors in [2.13] utilized the concept of proxy re-encryption in addition to erasure correcting codes to provide resident and forwarding data security.

2.3. Security and Performance

A secure and optimal placement of data objects in a distributed system is presented in [2.14]. An encryption key is divided into n shares and distributed on different sites within the network. The division of a key into n shares is carried out through the (k, n) threshold secret sharing scheme. The network is divided into clusters. The number of replicas and their placement is determined through heuristics. A primary site is selected in each of the clusters that allocate the replicas within the cluster. The scheme presented in [2.14] combines the replication problem with security and access time improvement. Nevertheless, the scheme focuses only on the security of the encryption key. The data files are not fragmented and are handled as a single file.

Ref. [2.15] also approached the security and optimal data placement through fragmentation and replication of the data objects. A data file is fragmented into small fragments, encrypted, and placed in distributed fashion over the network. The replication is performed for increased data availability and is done in a random manner by the aforesaid scheme. However, the scheme does not deal with the replica placement and associated performance issues.

Similarly, the authors of [2.16] studied the problem of reasoning about the engineering trade-offs inherent in data distribution scheme selection. The choice of an encoding algorithm and its parameters positions a system at a particular point in a complex trade-off space between performance, availability, and security. The authors are of the view that no single data distribution scheme is right for all systems. Instead, the right choice for any particular system depends on an array of factors, including expected workload, system component characteristics, and desired levels of availability and security. Therefore, the authors of [2.16] proposed an approach to selecting a better data distribution scheme to create a balance between security, availability, and performance. At a high level, this new approach consists of three steps: enumerating possible data distribution schemes (<algorithm, parameters>pairs), modeling the consequences of each scheme, and identifying the best-performing scheme for any given set of availability and security requirements. The aforesaid steps selects the best algorithms for data distribution that optimally create a balance between security, availability, and performance.

2.4. References

[2.1] A. Juels and A. Opera, “New approaches to security and availability for cloud data,” Communications of the ACM, Vol. 56, No. 2, 2013, pp. 64-73.

[2.2] A. Yun, C. Shi, and Y. Kim, “On protecting integrity and confidentiality of cryptographic file system for outscored storage,” *Proceedings of 2009 ACM workshop on cloud computing security CCSA’09,* pp. 67-76, 2009.

[2.3] S. Kamara and K. Lauter, “Cryptographic cloud storage,” *Financial Cryptography and Data Security,* Springer Berlin Heidelberg, 2010, pp. 136-149.

[2.4] D. Zissis and D. Lekkas, “Addressing cloud computing security issues,” Future Generation Computer Systems, Vol. 28, No. 3, 2012, pp. 583-592.

[2.5] Y. Tang, P. P. Lee, J. C. S. Lui, and R. Perlman, “Secure overlay cloud storage with access control and assured deletion,” IEEE Transactions on Dependable and Secure Computing, Vol. 9, No. 6, Nov. 2012, pp. 903-916.

[2.6] G. Kappes, A. Hatzieleftheriou, and S. V. Anastasiadis, “Dike: Virtualization-aware Access Control for Multitenant Filesystems,” University of Ioannina, Greece, Technical Report No. DCS2013-1, 2013.

[2.7] L. Wei, H. Zhu, Z. Cao, X. Dong, W. Jia, Y. Chen, and A. V. Vasilakos, “Security and privacy for storage and computation in cloud computing,” Information Sciences, Vol. 258, 2014, pp. 371-386.

[2.8] S. K. Sood, “A combined approach to ensure data security in cloud computing,” Journal of Network and Computer Applications, Vol. 35, No. 6, 2012, pp. 1831-1838.

[2.9] Q. Liu, G. Wang, and J. Wu, “Time-based proxy re-encryption scheme for secure data sharing in a cloud environment,” Information Sciences, Vol. 258, 2014, pp. 355-370.

[2.10] H. Lin and W. Tzeng, “A secure erasure code-based cloud storage system with secure data forwarding,” IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 6, June 2012, pp. 995-1003.

[2.11] H. Lin and W. Tzeng, “A secure decentralized erasure code for distributed network storage,” IEEE Transactions on Parallel and Distributed Systems, vol. 21, no. 11, Nov. 2010, pp. 1586-1594.

[2.12] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, “Toward secure and dependable storage services in cloud computing,” IEEE Transactions on Services Computing, Vol. 5, No. 2, 2012, pp. 220-232.

[2.13] H.Y. Lin and W. G. Tzeng, “A secure erasure code-based cloud storage system with secure data forwarding,” IEEE Transactions on Parallel and Distributed Systems, Vol. 23, No. 6, 2012, pp. 995-1003.

[2.14] M. Tu, P. Li, Q. Ma, I-L. Yen, and F. B. Bastani, “On the optimal placement of secure data objects over Internet,” In Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, pp. 14-14, 2005.

[2.15] A. Mei, L. V. Mancini, and S. Jajodia, “Secure dynamic fragment and replica allocation in large-scale distributed file systems,” IEEE Transactions on Parallel and Distributed Systems, Vol. 14, No. 9, 2003, pp. 885-896.

[2.16] J. J. Wylie, M. Bakkaloglu, V. Pandurangan, M. W. Bigrigg, S. Oguz, K. Tew, C. Williams, G. R. Ganger, and P. K. Khosla, “Selecting the right data distribution scheme for a survivable storage system,” Carnegie Mellon University, Technical Report CMU-CS-01-120, May 2001.

3. DASCE: DATA SECURITY FOR CLOUD ENVIRONMENT WITH SEMI-TRUSTED THIRD PARTY

This paper is submitted to *IEEE Transactions on Cloud Computing (TCC)* and is in the second round of review. The authors of the paper are Mazhar Ali, Saif U. R. Malik, and Samee U. Khan.

3.1. Introduction

Cloud computing has emerged as a promising computing paradigm and has shown tremendous potential in managing the hardware and software resources located at third-party service providers. On-demand access to the computing resources in a pay-as-you-go manner relieves the customers from building and maintaining complex infrastructures [3.1, 3.13]. Cloud computing presents every computing component as a utility, such as software, platform, and infrastructure. The economy of infrastructure, maintenance, and flexibility makes cloud computing attractive for organizations and individual customers [3.32]. Despite benefits, cloud computing faces certain challenges and issues that hinder widespread adoption of cloud. For instance, security, performance, and quality are a few to mention [3.10, 3.27].

The development and operation of data storage sites is ongoing process in organizations. Off-site data storage is a cloud application that liberates the customer from focusing on data storage systems [3.10]. Representing system characteristics and capabilities as utility, causes the user to focus on aspects directly related to data (security, transmission, processing) [3.6, 3.33]. However, moving data to the cloud, administered and operated by certain vendor requires high level of trust and security. Multiple users, separated through logical barriers of virtual machines, share resources including storage space. Multi-tenancy and virtualization generate risks and underpins the confidence of users to adopt the cloud model [3.2, 3.3]. Armbrust *et al.* [3.1] ranked data confidentiality and auditing at number three in the list of top ten obstacles impeding widespread cloud adoption. Data can be used by the cloud service providers without authorization [3.2, 3.23, 3.4] and can be accessed by other machines in cloud [3.23, 3.3].

Data being the principal asset for organizations needs to be secured especially, when data must enter a public cloud. To avoid unauthorized access to the cloud data, access control mechanism must be enforced [3.16, 3.17]. Moreover, data leakage and data privacy strategies must be employed so that only authorized users can access and utilize data. Refraining cloud service providers from utilizing the customer data requires high preventive measures [3.3]. Encryption techniques provide a solution to ensure privacy and confidentiality of stored data. However, key management becomes a prime issue in the case of encryption [3.28, 3.31]. Cryptographic keys need to be stored and protected. Compromise or failure of a key storage facility may lead to the loss of data. Therefore, cryptographic keys must be stored in a robust manner and a single point of failure should not affect the availability of data [3.31].

The security concerns of outsourcing data to public clouds, serves as our motivation to work for the development of data security technique. We aim for a technique capable of addressing the aforementioned critical issues. We propose a data security scheme that uses key manager servers for the management of cryptographic keys. Shamir's (k, n) threshold scheme [3.26] is used for the management of keys that uses k shares out of n to rebuild the key. Access to key and data is ensured through a policy file that states policies under which access is granted to the keys. The client generates random symmetric keys for encryption and integrity functions. Symmetric keys are protected by the public key generated by the key manager(s) (Fig. 3.1). All of the symmetric keys are deleted from the client afterwards. Encrypted data and keys are uploaded to the cloud. For downloading the data, client presents a policy file to cloud and downloads the encrypted data and keys. Keys are decrypted by key manager(s). Thereafter, the client decrypts the data.

We review the scheme presented in [3.29], called File Assured Deletion (FADE). The FADE is a light-weight and scalable technique that assures deletion of files from cloud when user asks for deletion. However, during our analysis, FADE fell short on issues of security of keys and authentication of participating parties. Based on our analysis and issues identified with FADE, we propose enhancements to the scheme and name it as Data Security for Cloud Environment with Semi-Trusted Third Party (DaSCE) that enhances the security of keys and authentication process. Moreover, to mitigate the man-in-the-middle-attack, we included supplementary steps for the session key establishment process. The aforesaid steps augment the security level and prohibit the malicious user to carry out the attack at slight performance overheads. However, the results from our verification analysis revealed that DaSCE is more secure than FADE when man-in-the-middle attack was introduced. Our major contributions include:

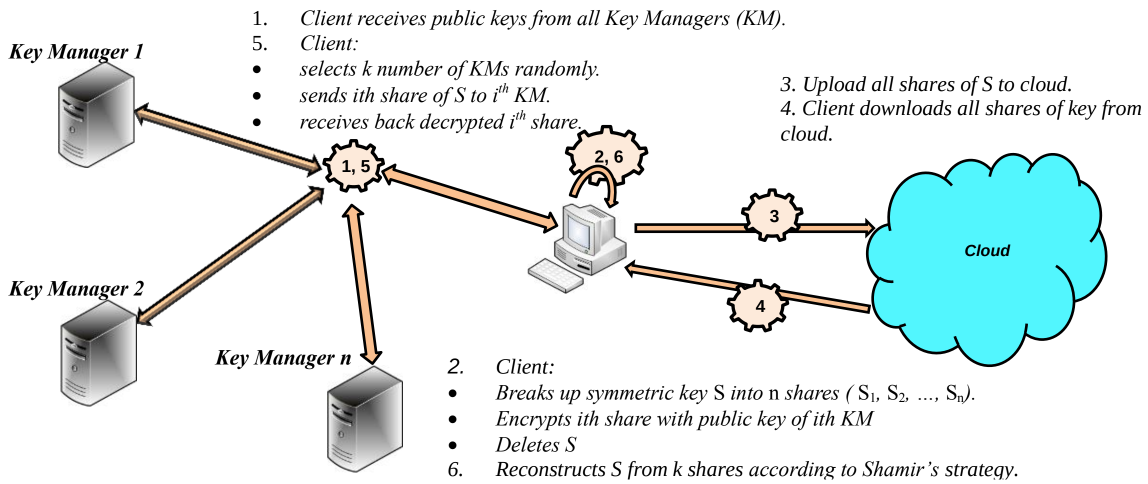

1. Client receives public keys from all Key Managers (KM).
5. Client:
• selects k number of KMs randomly.
• sends ith share of S to i^{th} KM.
• receives back decrypted i^{th} share.
Key Manager 1
3. Upload all shares of S to cloud.
4. Client downloads all shares of key from cloud.
2, 6
1, 5
3
Cloud
4
Key Manager 2
Key Manager n
2. Client:
• Breaks up symmetric key S into n shares ($S_1, S_2, \ldots, S_n$).
• Encrypts ith share with public key of ith KM
• Deletes S
6. Reconstructs S from k shares according to Shamir's strategy.

Fig. 3.1. Shamir's (*k, n*) threshold scheme in DaSCE.

- Development of a security scheme (DaSCE) for outsourced data to cloud that uses a combination of symmetric and asymmetric encryption. The DaSCE ensures data confidentiality at a cloud infrastructure, as long as it is in use by the client. It also assures that data gets deleted and becomes unrecoverable after the user deletes it from the cloud.
- Enforcing access control to both data and key through validity of policies and mutual authentication between client and key managers, and client and cloud. Digital signatures and variation of Diffie-Hellman is used for mutual authentication of parties. Successful authentication and session key establishment results in access to asymmetric keys that are used in subsequent cryptographic operations.
- Ensuring the integrity of data by use of symmetric key and message authentication code and securing symmetric keys with asymmetric keys generated by third party key managers.
- Formal modeling and verification of FADE and DaSCE by using High Level Petri Nets (HLPN), SMT-Lib, Z3 solver, and Scyther.
- We implemented a prototype of DaSCE and evaluated the performance of DaSCE based on time consumption parameters (file upload time, file download time, cryptographic operations time).

3.2. File Assured Deletion (FADE)

The FADE protocol provides privacy, integrity, access control, and assured deletion to outscored data. The FADE uses both symmetric and asymmetric keys. Symmetric keys are protected by using Shamir's (k, n) scheme to ample the trust level in the key. The FADE works with a group of key managers (KM). Following keys are used by FADE protocol. The variable K

is termed as data key and is used to encrypt file *F* of the client and *S* as secret key that is used to encrypt *K*. The public/private key pair generated by *KMs* is represented by (e_i, d_i) and is used to encrypt *S*. The *K* and *S* are symmetric keys. The operations supported by FADE are: **(a)** File upload, **(b)** File download, **(c)** Policy Revocation, and **(d)** Policy Renewal. The aforementioned operations are explained below. The notations used in the paper are presented in Table 3.1.

Table 3.1. Notations and their meanings.

Notation	**Meanings**
KM	Key manager
F	File
K	A symmetric key
S	A symmetric key.
e_i	Public key parameter.
n_i	Public key parameter.
d_i	Private key parameter.
e_j	Modified/New public key parameter.
n_j	Modified/New public key parameter.
d_j	Modified/New private key parameter.
$\{F\}_K$	File encrypted with key *K*.
$\{K\}_S$	*K* encrypted with key *S*.
S^e	*S* encrypted with public key *e*.
MAC	Message Authentication code
HMAC	Hash-based MAC
P_i	Original policy file of client
P_j	Modified policy file
HLPN	High Level Petri Net
IK	Integrity key for MAC calculation
ABE	Attribute based encryption
AES	Advance Encryption Standard

3.2.1. File Upload

When data must be uploaded to the cloud, the client requests the *KM* to generate a public/private key pair. The said is done by sending a policy file, P_i, to the *KM*. The *KM* generates the key pair, associates that with the P_i, and sends the public part of the key (e_i, n_i) to the client. After receiving public key for P_i, the client performs the following cryptographic operations. The client encrypts *F* with *K* to generate $\{F\}_K$ (*F* encrypted with *K*). The *K* is then encrypted with S_i to get $\{K\}_{Si}$. Subsequently, S_i is encrypted with the public key generated by the *KM* with P_i. The S_i is encrypted using asymmetric encryption (S_i^{ei} *mod n*). The P_i, $\{F\}_K$, $\{K\}_{Si}$, and (S_i^{e} *mod n*) are uploaded to the cloud afterwards. The hashed MAC (HMAC) of data file is also uploaded with the encrypted file. The client deletes all of the symmetric keys through secure overwriting. The process of file upload is shown in the Figure 3.2(a).

When FADE works with full quorum of *KMs*, S_i is divided into *n* shares and each share is encrypted with a public key generated by one of the *KMs*. The key is divided based on Shamir's

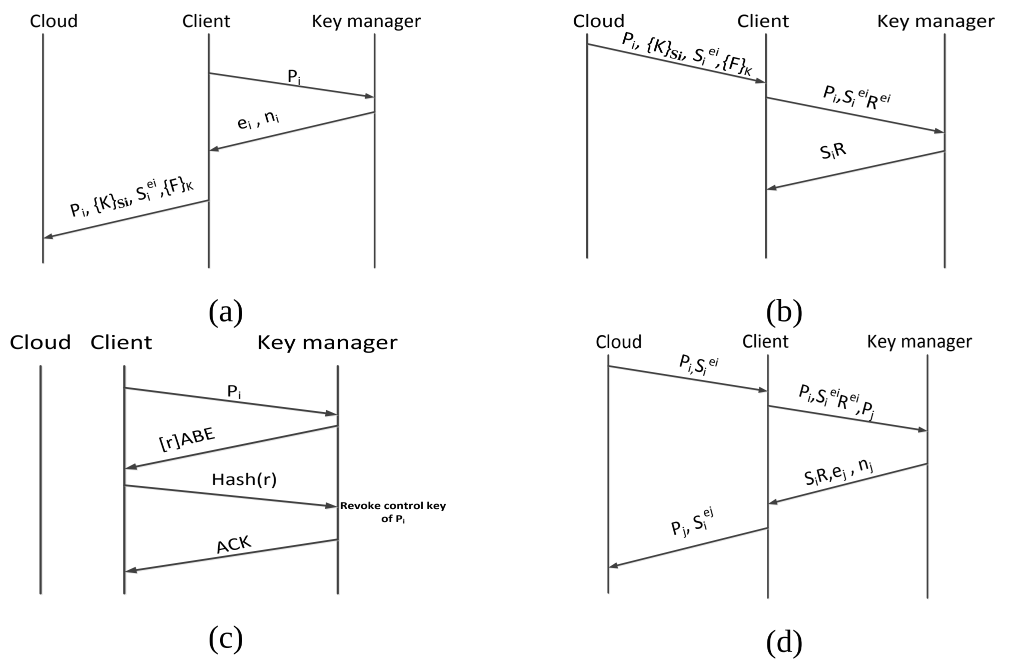


Fig. 3.2. FADE (a) File upload, (b) File download, (c) Policy revocation, and (d) Policy renewal (single key manager) [3.29].

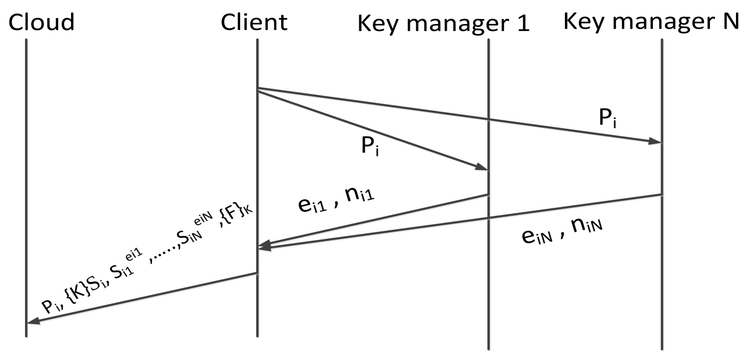


Fig. 3.3. FADE file upload with multiple key managers [3.29].

(k, n) threshold scheme. To get back the S_i, k shares are needed. The FADE protocol does not authenticate the client for the upload process. The process with multiple *KMs* is shown in Fig. 3.3. When data must be uploaded to the cloud, the client requests the *KM* to generate a public/private key pair. The said is done by sending a policy file, P_i, to the *KM*. The *KM* generates the key pair, associates that with the P_i, and sends the public part of the key (e_i, n_i) to the client. After receiving public key for P_i, the client performs the following cryptographic operations. The client encrypts F with K to generate $\{F\}_K$ (F encrypted with K). The K is then encrypted with S_i to get $\{K\}_{Si}$. Subsequently, S_i is encrypted with the public key generated by the *KM* with P_i. The S_i is encrypted using asymmetric encryption (S_i^{ei} *mod n*). The P_i, $\{F\}_K$, $\{K\}_{Si}$, and (S_i^e *mod n*) are uploaded to the cloud afterwards. The hashed MAC (HMAC) of data file is also uploaded with the encrypted file. The client deletes all of the symmetric keys through secure overwriting. The process of file upload is shown in the Figure 3.2(a).

When FADE works with full quorum of *KMs*, S_i is divided into n shares and each share is encrypted with a public key generated by one of the *KMs*. The key is divided based on Shamir's (k, n) threshold scheme. To get back the S_i, k shares are needed. The FADE protocol does not authenticate the client for the upload process. The process with multiple *KMs* is shown in Fig. 3.3.

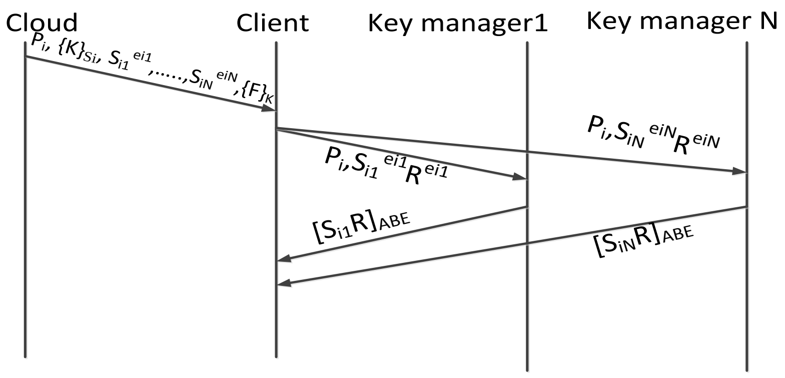

Cloud
Client
Key manager1
Key manager N
P_i, $\{K\}_{Si}$, S_{i1}^{ei1},…..,S_{iN}^{eiN},$\{F\}_K$
P_i,$S_{iN}^{eiN}R^{eiN}$
P_i,$S_{i1}^{ei1}R^{ei1}$
$[S_{i1}R]_{ABE}$
$[S_{iN}R]_{ABE}$

Fig. 3.4. File download using ABE with multiple key managers [3.29].

3.2.2. File Download

The client requests the cloud for file and encrypted keys to download. The client checks for the integrity of the file through the HMAC. Afterwards, the client generates a secret number *R* and calculates R^{ei} and then generates $S_i^e R^{ei} = (S_iR)^{ei}$. The $(S_iR)^{ei}$ is then sent to KM for decryption. The *KM* decrypts $(S_iR)^{ei}$ with corresponding d_i and sends back S_iR. At this point, ABE comes into the play. The *KM* sends S_iR with ABE, where the attributes used for ABE are based on P_i. The client extracts S_i from the received message and decrypts *K* that is used to decrypt *F*. The process is highlighted in Fig. 3.2(b). Similarly, the file download with multiple *KMs* takes place according to the flow of messages shown in Fig. 3.4.

3.2.3. Policy Revocation

If P_i needs to be revoked, the client requests the *KM* by sending the P_i. The *KM* generates a random number *r* and sends *r* to the client after encryption with ABE. The authentic client decrypts *r*, calculates the hash value, and sends back to the *KM*. After verification the *KM* revokes *Pi* and acknowledges the client as depicted in Fig. 3.2 (c).

3.2.4. Policy Renewal

If P_i needs to be renewed as P_j, the client downloads all of the keys and sends P_i and encrypted S_i to the *KM* along with P_j. The *KM* decrypts S_i. Moreover, the *KM* sends new public key parameters (e_j, n_j) to the client as outlined in Fig. 3.2 (d). We will now formally analyze FADE in the following section.

3.3. Analysis of FADE

The FADE is a light weight protocol that does not require heavy modifications in cloud architecture. The analysis of FADE identified the following issues.

3.3.1. File Upload

In file upload process of FADE we assume that there is a man-in-the-middle (intruder) between client and *KM*. The intruder can intercept P_i and send P_j (modified P_i) to *KM*. In the second step, the *KM* sends (e_i, n_i). The intruder intercepts (e_i, n_i) and sends the client modified parameters (e_j, n_j). The client encrypts the keys with (e_j, n_j) and uploads to the cloud. The client cannot verify that the received (e_j, n_j) is from *KM* or any other entity. The aforesaid issue is highlighted in Fig. 3.5 (a).

In the original file upload process of FADE, independence of Step 1 and Step 2 allows the intruder to carry out the attack. The policies received by the *KM* are neither from the client nor does the client receive keys from the *KM*. However, both assume a valid data exchange with each other. As a result, the client encrypts the S_i with the (e_j, n_j). The encryption of data with the intruder's generated keys may result in any of the following scenarios:

3.3.1.1. Intruder Fetches the Data during Download Process

During the download process, the intruder can intercept the data. As S_i is encrypted with (e_j, n_j) that is generated by the intruder; therefore, after reviving S_i^{ej} intruder can recover S_i by

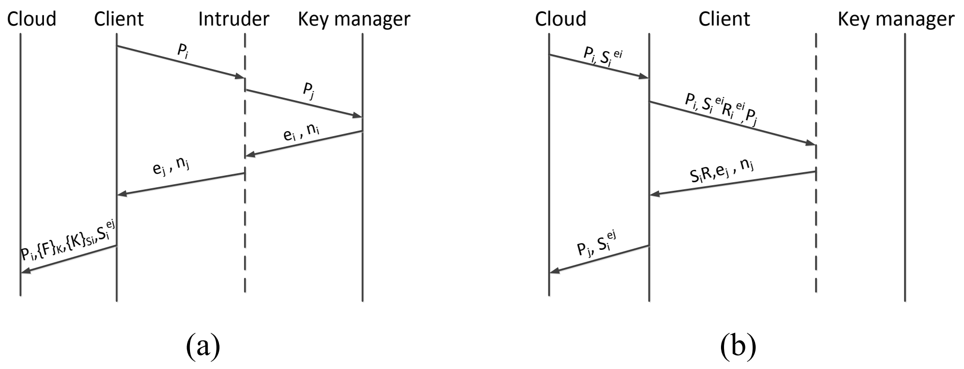


Fig. 3.5. (a) Man-in-the-middle attack that causes encryption with the wrong keys (b) Exploitation of policy renewal process.

decryption with a corresponding d_j as: $S_i = (S_i^{ej})d_j \ mod \ n$. Once S_i is decrypted, the intruder can easily decrypt *K* and gather *F*.

3.3.1.2. Intruder Stays Aside during Download Operation

The client downloads the data from the cloud and sends S_i to the *KM* for decryption. As S_i was encrypted by the public key that was originally generated by the intruder, the *KM* will not be able to decrypt the correct S_i. Therefore, access to the data will be denied. The denial of access will result in the loss of data. The *KM* generates the keys based on ABE having policies defined in P_i. During the attack, the *KM* generates the keys with P_j (modified P_i). Therefore, even the attributes will not correspond to the original policies. Same attack flow can be modeled for multiple *KMs* as shown in Fig. 3.6. As highlighted in Fig. 3.6, all S_i*'s* are encrypted with keys generated by the intruder and the corresponding d_i*'s* are held by the intruder. Therefore, the intruder can generate S_i. However, intruder must intercept *k* portions of S_i.

3.3.2. Policy Renewal

Fig. 3.5 (b) shows how the intruder can exploit the policy renewal process of FADE for denying access of data to a legitimate user. It is noteworthy to mention that the exploitation is only possible if initially the attack depicted in Fig. 3.5 (a) is already carried out. The client after

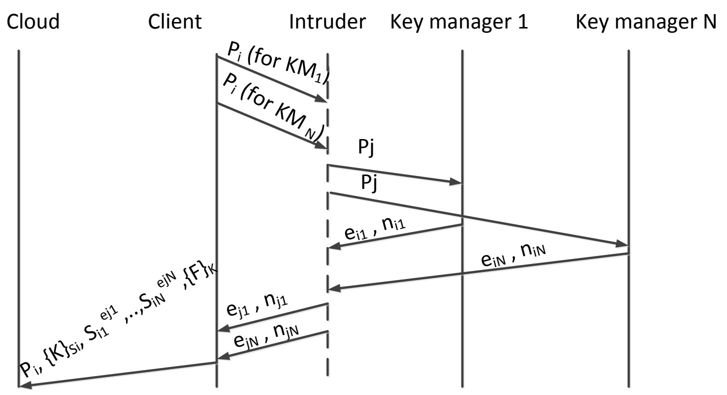


Fig. 3.6. Man-in-the-middle with multiple key managers.

downloading S_i from the cloud sends S_i along with P_i to the *KM*. The intruder intercepts the data, decrypts S_i with the corresponding private key, generates a new pair of public/private key, and sends it to the client. The client performs cryptographic operations (as it did earlier) and uploads the data and keys to the cloud.

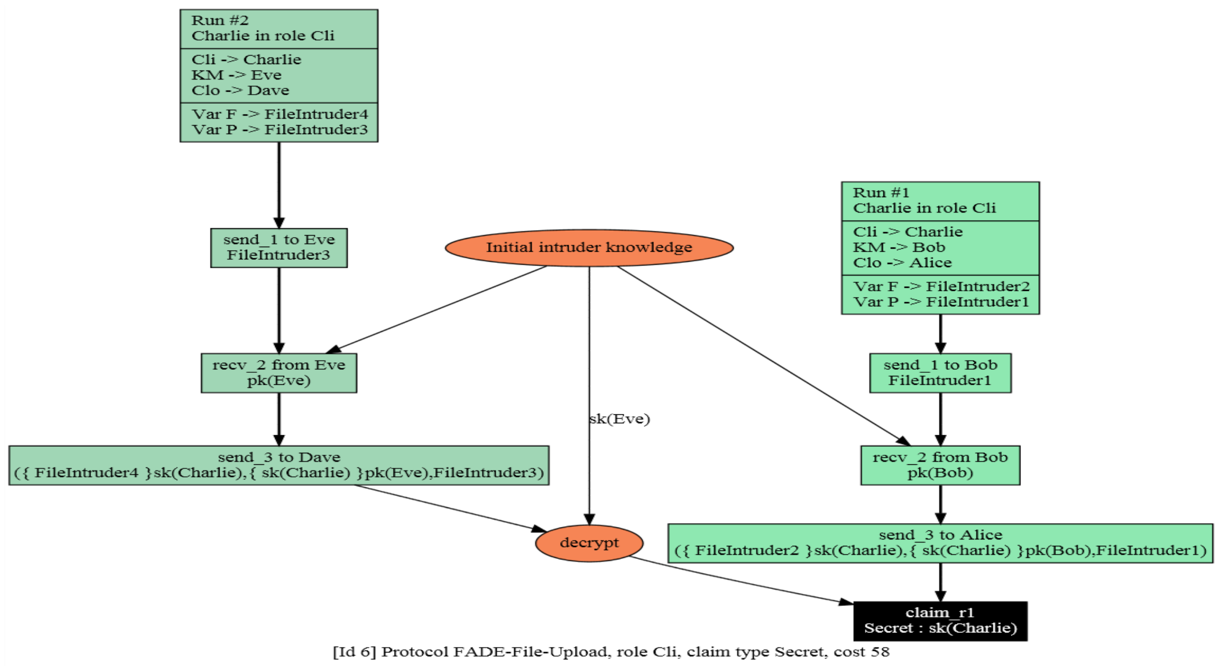


Fig. 3.7. Scyther verification of FADE.

3.3.3. Attack Verification through Scyther

In this section, we verify the attack defined in the previous section using Scyther, which is a graphical tool for analysis, verification, and falsification of security protocols [3.5]. We modeled FADE in Scyther and verified whether S_i and F remain secret under the setup or otherwise. The verification is performed by a "claim" (see Fig. 3.7) that S_i remains secret during the process. The Scyther verified the validity of the claim and reported the attack that was discussed in Section 3.3.1.

In Scyther, Charlie plays a role of a client, Bob as the *KM*, Alice as the cloud, and Eve as the intruder. The Run# 1 of the Scyther is not an intercepted run while Run#2 is a run where intruder plays the part. Eve intercepts the P_i and sends Charlie the generated public key *pk*(*Eve*). Later on Eve can use corresponding private key *sk*(*Eve*) to decrypt the secret key of Charlie (*sk*(*Charlie*)). In this model, *sk*(*Charlie*) is the same key as S_i, in the explained model. Our claim that S_i will remain secret is falsified by Scyther by producing the counter attack.

3.3.4. HLPN

Petri Nets provide graphical and mathematical representation of the system and can be applied to variety of systems for instance stochastic, deterministic, and asynchronous computations [3.24]. A HLPN is a 7-tuple $N = (P, T, F, \varphi, R_n, L, M_0)$, where P is set of places; T refers to the set of transitions such that $P \cap T = \emptyset$; Flow relations are defined by F such that $F \subseteq (P \times T) \cup (T \cup P)$; φ maps places P to the data types. Rules for transitions are defined by R_n; L is a label on F and M_0 represents the initial marking [3.24]. In the above definition, the structure of the Petri Net is given by P, T, and F; whereas, (φ, R_n, L) provide the static semantics of the Petri Net model.

3.3.5. SMT-Lib and Z3 solver

SMT has roots in Boolean Satisfiability Solvers (SAT) [3.11, 3.12, and 3.22]. SMT-Lib provides a common input platform and benchmarking framework that helps in the evaluation of the systems. We use Z3 solver with SMT-Lib that is a theorem prover developed at Microsoft Research. Z3 is an automated satisfiability checker. In addition, Z3 determines whether the set of formulae are satisfiable in the built-in theorems of SMT-Lib [3.21].

3.3.6. Verification through HLPN Model

In this section, we formally analyze the man-in-the-middle attack on FADE protocol. We use High Level Petri Nets (HLPN) and Z language [3.8, 3.11, 3.12, 3.22, 3.24, and 3.25] to perform formal analysis. HLPN define mathematical properties for the system and simulate the system to analyze the behavior. We verify HLPN model of FADE using Satisfiability Modulo Theories Library (SMT-Lib) and Z3 solver. To verify the model, the Petri Net model is first translated into SMT along with the specified properties. Subsequently, Z3 solver is used to determine whether or not the properties hold.

3.3.7. Formal Verification

The verification process checks for the correctness of the system. In model checking: (a) description of the system is provided stating properties or rules of the system, (b) system is represented by a model, and (c) some verification tool is used to check whether the model holds the specified properties or not. In this paper we use the bounded model checking to verify the man-in-the-middle attack on FADE.

The HLPN model for FADE is given in Fig. 3.8. The model is given with the intruder between the client and *KM*. The data types used in the model and their mappings are shown in

Table 3.2 and Table 3.3, respectively. All the rectangular black boxes in HLPN are transitions and belong to the set *T*. The circles are places and belong to the set *P*.

The process starts with the client sending *P_i* to the *KM*. The file is intercepted by the intruder. The file sending and receiving is performed on transitions *Send_P_i* and *Rcv_P_i*. Rule (3.1) and Rule (3.2) are mapped to the aforesaid transitions.

$$R(Send\_P_i) = \forall\, x_1 \in X_1, \forall\, x_2 \in X_2 \mid x_2 \coloneqq x_1[1] \wedge \\ X_2' = X_2 \cup \{x_2\}, \tag{3.1}$$

$$R\left(Rcv_{P_i}\right) = \forall\, x_3 \in X_3, \forall\, x_4 \in X_4 \mid x_4 \coloneqq x_3 \wedge \\ X_4' = X_4 \cup \{x_4\}. \tag{3.2}$$

The intruder generates *P_j* and sends it to the *KM*. The transition *Gen_fake* is fired upon interception of original *P_i*. Following are the three transition and the corresponding rules.

$$\boldsymbol{R}\left(Gen_{fake}\right) = \forall\, x_5 \in X_5, \forall\, x_6[2] \in X_6, \\ \forall\, x_6[3] \in X_6, \forall\, x_6[4] \in X_6, \\ \forall\, x_6[5] \in X_6 \mid x_5 \leftrightarrow X_6[2] \leftrightarrow X_6[3] \leftrightarrow X_6[4] \leftrightarrow X_6[5] \wedge \\ X_6' = X_6 \cup \{x_5, x_6[2], x_6[3], x_6[4], x_6[5]\}, \tag{3.3}$$

$$\boldsymbol{R}\left(Send_{P_j}\right) = \forall\, x_7[2] \in X_7, \forall\, x_8 \in X_8 \mid \\ x_8 \coloneqq X_7[2] \wedge \\ X_8' = X_8 \cup \{x_8\}, \tag{3.4}$$

$$\boldsymbol{R}\left(Rcv_{P_j}\right) = \forall\, x_9 \in X_9, \forall\, x_{10} \in X_{10} \mid x_{10} \coloneqq X_9 \wedge \\ X_{10}' = X_{10} \cup \{x_{10}\}. \tag{3.5}$$

The keys generated and sent by KM are intercepted by the intruder. The Following rules (3.6) and (3.7) capture the above three transitions.

Table 3.2. Data types used in FADE HLPN model.

Types	**Description**
Policy	A string type for describing file access policy.
File	A string type holding data to be protected.
K	A string type representing symmetric key.
S	A string type representing symmetric key.
e	Public Key parameter.
n	Public Key parameter.
d	Private Key parameter.
$\{F\}_K$	File encrypted with key *K*.
$\{K\}_S$	K encrypted with key S.
S^e	S encrypted with public key e.

Table 3.3. Mapping of data types and places.

Types	**Description**
φ(a1)	$\mathbb{P}$(Policy × File × K × S)
φ(c1)	$\mathbb{P}$(Policy)
φ(I1)	$\mathbb{P}$(Policy)
φ(I2)	$\mathbb{P}$(Policy ×Policy × e × n × d × e × n)
φ(c2)	$\mathbb{P}$(Policy)
φ(b1)	$\mathbb{P}$(Policy)
φ(b2)	$\mathbb{P}$(Policy × e × n × d)
φ(c3)	$\mathbb{P}$(e × n)
φ(c4)	$\mathbb{P}$(e × n)
φ(a2)	$\mathbb{P}$(e × n)
φ(a3)	$\mathbb{P}$(Policy ×$\{F\}_K$ × $\{K\}_S$ × Se)

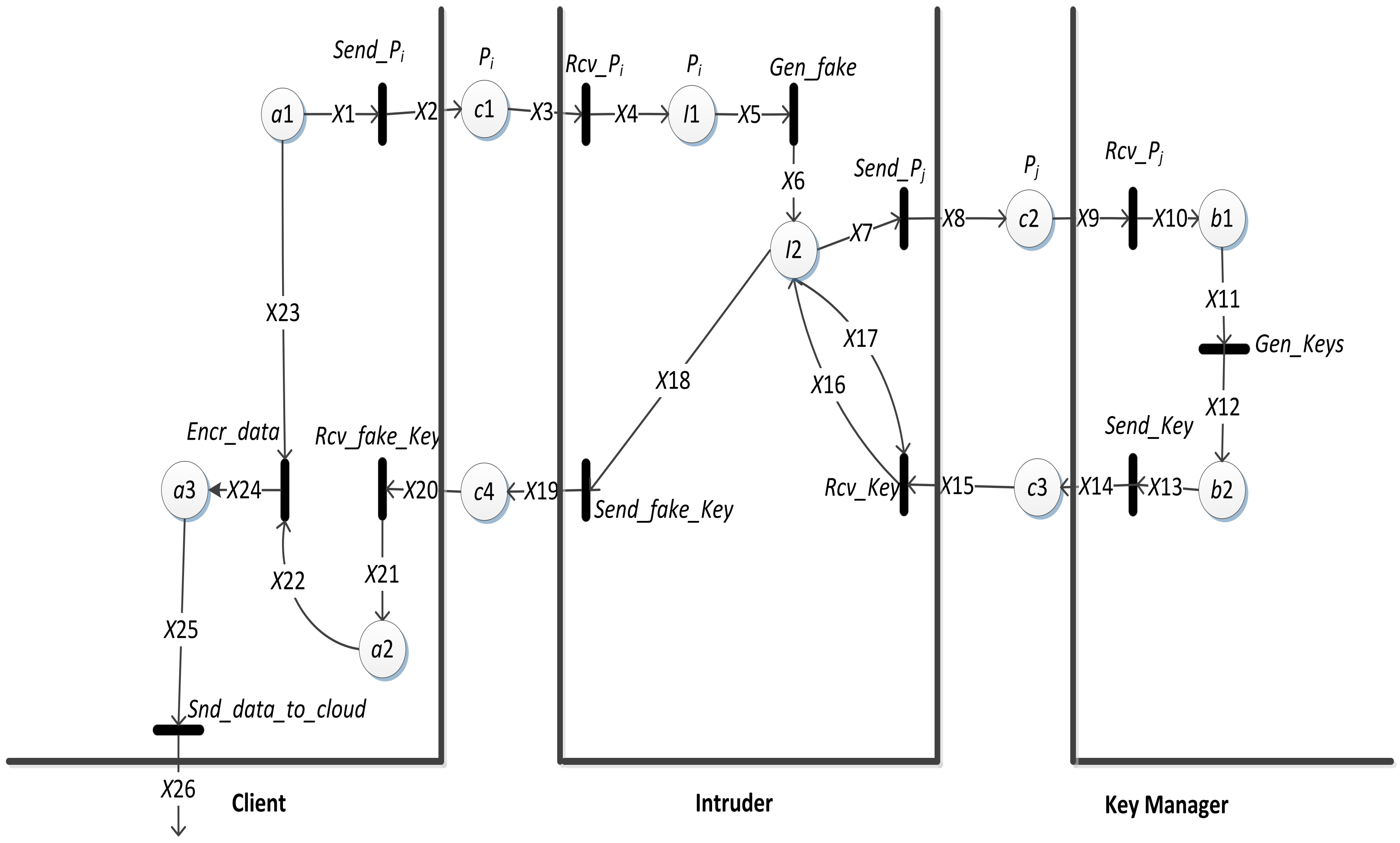


Fig. 3.8. FADE HLPN model with intruder.

$$\boldsymbol{R}(Gen\_Keys) = \forall x_{11} \in X_{11}, \forall x_{12}[1] \in X_{12}, \forall x_{12}[2] \in X_{12}, \forall x_{12}[3] \in X_{12},$$

$$\forall x_{12}[4] \in X_{12} | \forall x_{12}[1] \coloneqq X_{11} \wedge x_{12}[1] \leftrightarrow X_{12}[2] \leftrightarrow X_{12}[3] \leftrightarrow X_{12}[4] \wedge \quad (3.6)$$

$$X'_{12} = X_{12} \cup \{x_{12}[1], x_{12}[2], x_{12}[3], x_{12}[4]\},$$

$$\boldsymbol{R}(Send\_Key) = \forall x_{13}[2] \in X_{13}, \forall x_{13}[3] \in X_{13}, \forall x_{14} \in X_{14} | \quad (3.7)$$

$$x_{14}[1] \coloneqq x_{13}[2] \wedge x_{14}[2] \coloneqq x_{13}[3] \wedge$$

$$X'_{14} = X_{14} \cup \{x_{13}[2], x_{13}[3]\},$$

$$\boldsymbol{R}(Rcv\_Key) = \forall x_{15} \in X_{15}, \forall x_{17}[1] \in X_{17}, \forall x_{17}[2] \in X_{17}, \forall x_{17}[3] \in X_{17} \quad (3.8)$$

$$\forall x_{17}[4] \in X_{17} \forall x_{17}[5] \in X_{17} \forall x_{17}[6] \in X_{17} \forall x_{17}[7] \in X_{17} |$$

$$x_{17}[2] \leftrightarrow X_{15}[1] \leftrightarrow x_{15}[2] \wedge$$

$$X'_{16} = X_{16} \cup \{x_{17}[1], x_{17}[2], x_{17}[3], x_{17}[4], x_{17}[5], x_{15}[1], x_{15}[2]\}.$$

The intruder generates and sends (e_j, n_j) to the client as depicted in (3.9) and (3.10).

$$\boldsymbol{R}(Send\_fake\_Key) = \forall x_{18}[3] \in X_{18}, \forall x_{18}[4] \in X_{18}, \forall x_{19} \in X_{19} |$$

$$x_{19}[1] \coloneqq X_{18}[3] \wedge x_{19}[2] \coloneqq X_{18}[4] \wedge X'_{19} = X_{19} \cup \{x_{18}[3], x_{18}[4]\}, \quad (3.9)$$

$$(Rcv\_fake\_Key) = \forall x_{20} \in X_{20}, \forall x_{21}[4] \in X_{21} | \quad (3.10)$$

$$x_{21}[1] \coloneqq X_{20}[1] \wedge x_{21}[2] \coloneqq x_{20}[2] \wedge$$

$$X'_{21} = X_{21} \cup \{x_{21}[1], x_{21}[2]\},$$

The client performs the cryptographic operations with (e_j, n_j) and sends all the encrypted data to the cloud. This is represented by the following rules.

$$\boldsymbol{R}(Encr\_data) = \forall x_{21} \in X_{21}, \forall x_{22} \in X_{22}, \forall x_{23} \in X_{23}, \forall x_{24} \in X_{24} |$$

$$x_{24}[1] \coloneqq x_{23}[1] \wedge x_{24}[2] \coloneqq x_{23}[2] encr(x_{23}[3]) \wedge x_{24}[3] \quad (3.11)$$

$$\coloneqq x_{23}[3] encr(x_{23}[4] \wedge$$

$$x_{24}[4] \coloneqq x_{23}[4] encr(x_{21}[1], x_{22}[2]) \wedge$$

$$X'_{24} = X_{24} \cup \{x_{24}[1], x_{24}[2], x_{24}[3], x_{24}[4]\},$$

$$\begin{gathered}\boldsymbol{R}(Snd\_data\_to\_Cloud) = \forall\, x_{25} \in X_{25}, \forall\, x_{26} \in X_{26} | x_{26}[1] \coloneqq x_{25}[1] \land \\ x_{26}[2] \coloneqq x_{25}[2] \land x_{26}[3] \coloneqq x_{25}[3] \land x_{26}[4] \coloneqq x_{25}[4] \land \\ X'_{26} = X_{26} \cup \{x_{26}[1], x_{26}[2], x_{26}[3], x_{26}[4]\}. \end{gathered} \tag{3.12}$$

In the above, *Encr_data* is the most crucial transition. Security of data and the keys are highly dependent on this transition. If the encryption is performed by using (e_j, n_j), then the data security is compromised. In this context, the property that we verified using SMT-Lib and Z3 is that: if the intruder is present, then the encryption operation is performed using the wrong keys. The property of the model is described using a formal language called Computational Tree Logic (CTL*). The CTL* uses numerous temporal operators to represent various operations [3.7, 3.20]. For instance, *A* represents "for all paths", *G* denotes "globally", and *F* characterizes "future state". The property specified in CTL* using temporal operators is given as: $AG(a1 \rightarrow AF\ a3)$. After translating the above model into SMT-Lib, we performed bounded checking using Z3 solver. The mentioned property was satisfied by the solver in 310 msec.

3.4. DaSCE

From Section 4, it is evident that the security of S_i in FADE depends on the key exchange between the client and the *KM*. If the key exchange is compromised, then S_i is compromised, that in turn, leaks all the keys and the data. We observed that the reason for the said attack is the independence of communication steps between the client and the *KM* that allows the attacker to launch the attack and subvert the whole process. In this section, we propose improvements in the communication process between **(a)** client and the *KM*, and **(b)** client and the cloud. Our proposed changes link the communication steps so as to avoid attacker to overtake the process.

We use the station-to-station (STS) protocol [3.9] and digital signature for authentication and session key establishment before any other exchange takes place. The keys generated by the *KMs* and policy files are exchanged using session keys. Some modifications are required in the subsequent operations of the protocol as the session keys are introduced to the FADE. The following subsections discuss the proposed mechanisms.

3.4.1. DaSCE Keys

The DaSCE makes use of both symmetric and asymmetric keys. The confidentiality and integrity services for data are provided through symmetric keys that are secured by using asymmetric keys. Asymmetric key pairs are generated by third party KMs. Out of the key pair, only public key is transmitted to the client. For secure transmission of keys, a session key is established between client and KM through STS protocol. To avoid man-in-the-middle attack, both client and KM are authenticated by use of digital signatures. As a new session key is used for every communication session between client and KM, the session key is exchanged through key exchange process and is not randomly generated. This also avoids weakness of randomly generated keys. The symmetric keys are generated once for data encryption by client and

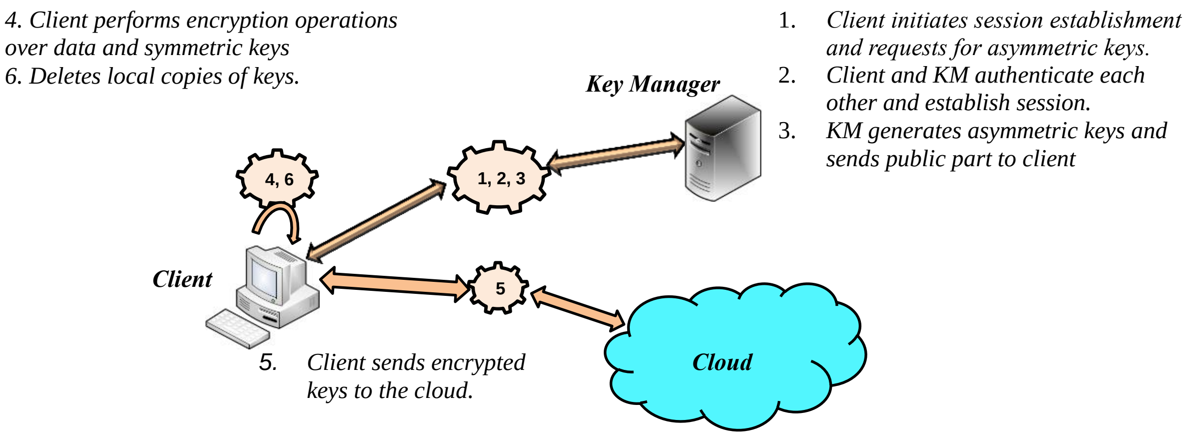

4. Client performs encryption operations over data and symmetric keys
6. Deletes local copies of keys.
Key Manager
1. Client initiates session establishment and requests for asymmetric keys.
2. Client and KM authenticate each other and establish session.
3. KM generates asymmetric keys and sends public part to client
4, 6
1, 2, 3
Client
5
Cloud
5. Client sends encrypted keys to the cloud.

Fig. 3.9. Key management in DaSCE.

encrypted by another symmetric key named S_i. The S_i is finally protected by the public key received from KM. The encrypted keys are stored at cloud and client deletes the local copies of the keys. For decryption purpose, client establishes a session with KM and sends S_i to KM after masking with random number R. The KM decrypts S_i and sends back to client. The client unmasks S_i to get the symmetric keys. Fig. 3.9 depicts the key management process.

3.4.2. File Upload

For the establishment of session key, we assume that the parameters required are fixed and publically available to all of the users. We call these parameters as α and p where, α is a large number known as the primitive root and p is a large prime number. The process comprises of following steps.

- The client generates a random number x and calculates $\alpha^x \bmod p$ and sends to the *KM*.
- The *KM* generates a random number y and calculates $\alpha^y \bmod p$. The *KM* also calculates $(\alpha^x)^y$ as a session key, *EK*, between client and *KM*.
- The *KM* generates digital signature over $\{\alpha^y, \alpha^x\}$ ($S_{KM}\{\alpha^y, \alpha^x\}$) and encrypts it with the generated session key to generate $EK(S_{KM}\{\alpha^y, \alpha^x\})$.
- The *KM* sends (α^y , $EK(S_{KM}\{\alpha^y, \alpha^x\})$) to the client.

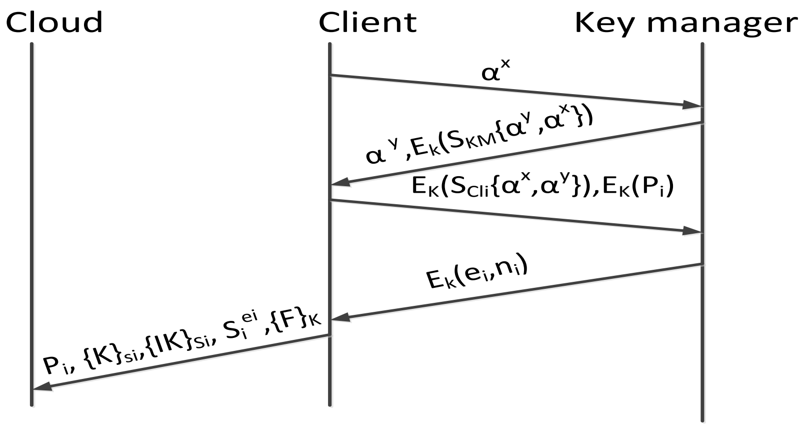

Cloud
Client
Key manager
α^x
$\alpha^y, E_K(S_{KM}\{\alpha^y, \alpha^x\})$
$E_K(S_{Cli}\{\alpha^x, \alpha^y\}), E_K(P_i)$
$E_K(e_i, n_i)$
$P_i, \{K\}_{si}, \{IK\}_{Si}, S_i^{ei}, \{F\}_K$

Fig. 3.10. DaSCE file upload with single key manager.

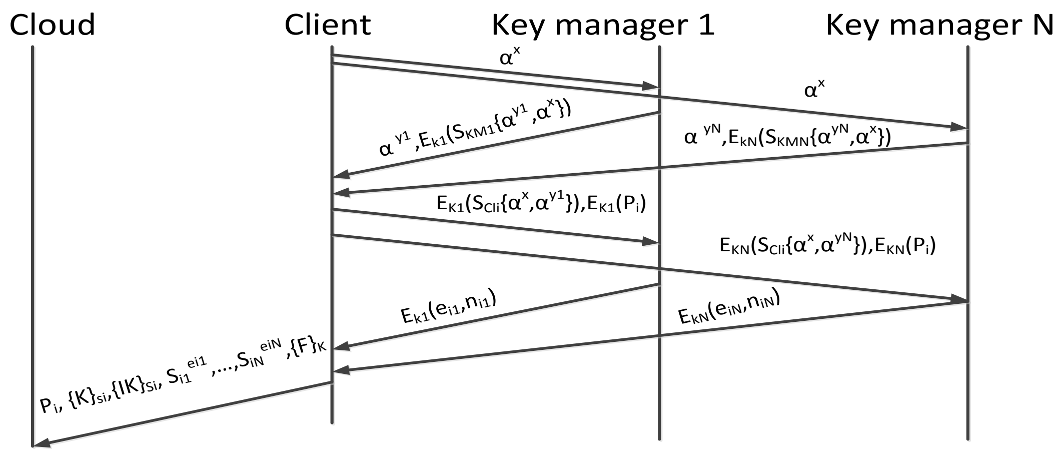

Cloud
Client
Key manager 1
Key manager N
α^x
α^x
α^y1,E_K1(S_KM1{α^y1,α^x})
α^yN,E_kN(S_KMN{α^yN,α^x})
E_K1(S_Cli{α^x,α^y1}),E_K1(P_i)
E_KN(S_Cli{α^x,α^yN}),E_KN(P_i)
E_K1(e_i1,n_i1)
E_kN(e_iN,n_iN)
P_i, {K}_Si,{IK}_Si, S_i1^ei1,...,S_iN^eiN,{F}_K

Fig. 3.11. DaSCE file upload with multiple key managers.

- The client verifies the signature using the public key of the *KM* and calculates the session key as $(\alpha^y)^x$.
- The client calculates $EK(S_{Cli}\{\ \alpha^x\ ,\ \alpha^y\ \})$ and encrypts P_i with *EK* and sends both of the calculated values to the *KM*. The sent message contains $EK(S_{Cli}\{\ \alpha^x\ ,\ \alpha^y\ \})$, $EK(P_i)$.
- The *KM* verifies the signature of the client. Upon successful verification, the *KM* decrypts P_i and generates (e_i, n_i) with P_i. The *KM* stores the decrypted P_i.
- The *KM* encrypts (e_i, n_i) with the *EK* to generate $(EK(e_i, n_i))$, *which is sent to the client.*
- *The client encrypts the file F* with key *K*, calculates MAC with *IK*; and encrypts *K* and *IK* with S_i. Afterwards S_i is encrypted with e_i. Subsequently, the client sends all the encrypted data to cloud.
- The client erases all of the keys except public key parameters received from the *KM*.

The file upload process is shown in Fig. 3.10. The calculations for session key include *mod p* operation which is not shown in the figure for clarity.

Similarly, the file upload process with multiple *KMs* is shown in Fig. 3.11. With multiple *KMs*, S_i is divided into *n* shares and each share is encrypted with the key from one of the managers according to (*k, n*)-threshold scheme. The interdependencies between file upload steps circumvent the man-in-the-middle attack. If higher level of security is required, then session key can also be established between the client and the cloud to keep the P_i exchange secure.

3.4.3. File Download

The file download process of DaSCE is depicted in Fig. 3.12. The process starts with the client downloading the data from the cloud. To decrypt *F*, we need *K* that is encrypted with S_i. The S_i is encrypted with (e_i, n_i) received from *KM*. The client establishes the session key with the *KMs* and during the process both the client and the *KMs* authenticate each other through digital signatures. The process of key establishment and authentication is the same as discussed in Section 3.4.2. In the third step, after verifying the authenticity of the *KMs*, the client generates a random number *R* and encrypts it with the public key of the corresponding *KM*. The client then calculates $S_i^{ei}R^{ei}$ and sends it along with its own signature and encrypted P_i. We combine these steps to minimize the communication overhead. The *KM* after verifying the digital signature of

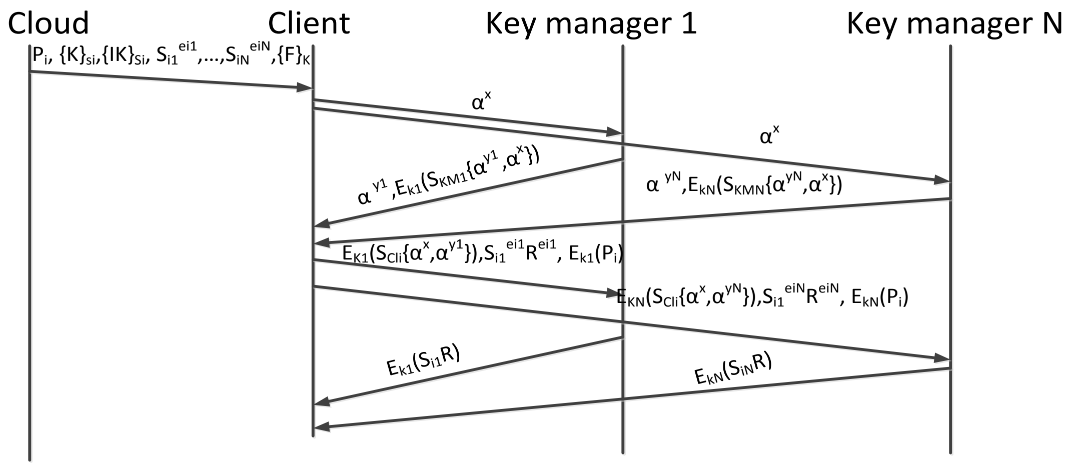

Cloud
Client
Key manager 1
Key manager N
P_i, $\{K\}_{si}$,$\{IK\}_{Si}$, S_{i1}^{ei1},…,S_{iN}^{eiN},$\{F\}_K$
α^x
α^x
α^{y1},$E_{k1}(S_{KM1}\{\alpha^{y1},\alpha^x\})$
α^{yN},$E_{kN}(S_{KMN}\{\alpha^{yN},\alpha^x\})$
$E_{K1}(S_{Cli}\{\alpha^x,\alpha^{y1}\})$,$S_{i1}^{ei1}R^{ei1}$, $E_{k1}(P_i)$
$E_{KN}(S_{Cli}\{\alpha^x,\alpha^{yN}\})$,$S_{i1}^{eiN}R^{eiN}$, $E_{kN}(P_i)$
$E_{k1}(S_{i1}R)$
$E_{kN}(S_{iN}R)$

Fig. 3.12. DaSCE file download with multiple key managers.

the client decrypts P_i and checks whether the policy still holds or otherwise. If the policy is valid, then the *KM* decrypts $S_i^{ei}R^{ei}$ with the corresponding d_i to generate S_iR. The purpose of R is to mask the actual value of S_i. The *KM* encrypts S_iR with the session key, which is sent to the client.

It is noteworthy to mention that in FADE, S_iR is returned by applying ABE. However, in the DaSCE, we do not use ABE, instead session key is used to send S_iR to the legitimate user. Therefore, the access control is being managed by the aforementioned technique. The client after receiving S_iR extracts S_i from S_iR. It is important to remember that with multiple *KMs*, a share of S_i will be received from at least k *KMs*. Consequently k number of S_is will be used to generate S_i. The client decrypts K and IK using S_i. It verifies the integrity of F using IK and decrypts F upon successful verification.

3.4.4. Policy Revocation

The same process of key establishment, as discussed in Section 3.4.2, is used for the policy revocation in DaSCE. The client encrypts P_i with the session key and sends to *KM*. The *KM* after performing decryption on P_i revokes the keys generated with P_i. The deleted keys include the private key d_i and associated prime numbers p_i and q_i. It also sends acknowledgement to the client.

When d_i associated with P_i is deleted, the corresponding S_i cannot be decrypted. This results in logical deletion of F as K cannot be decrypted without S_i. Therefore, we say that F is assuredly deleted. It is noteworthy that assured deletion does not correspond to the physical deletion of data. It is difficult to get the assurance of file deletion from the system outside the administrative control of data owner. For assured deletion we used the concept introduced in [3.34] and [3.35], where the inaccessibility of data is assured by deleting certain important information from the system. The DaSCE ensures the inaccessibility of the keys to make the data

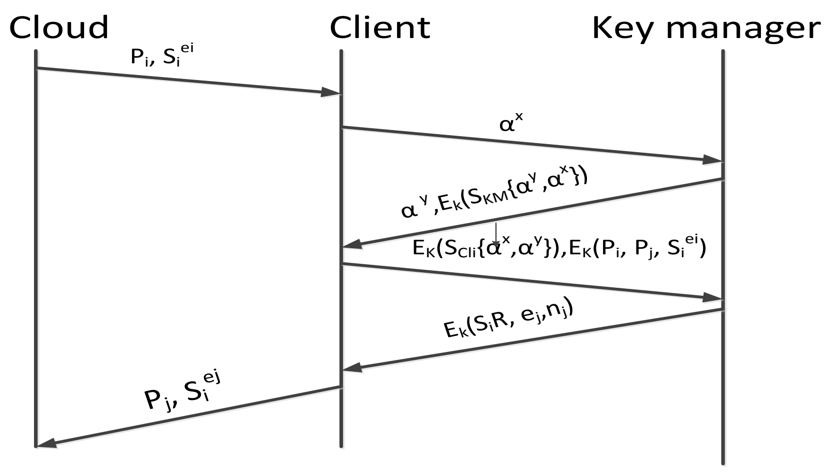

Cloud
Client
Key manager
P_i, S_i^{ei}
α^x
$\alpha^y, E_k(S_{KM}\{\alpha^y, \alpha^x\})$
$E_K(S_{Cli}\{\alpha^x, \alpha^y\}), E_K(P_i, P_j, S_i^{ei})$
$E_k(S_iR, e_j, n_j)$
P_j, S_i^{ej}

Fig. 3.13. DaSCE policy renewal.

unrecoverable. Therefore, the main security property of file assured deletion is that even if a *KM* does not remove the key from its storage, the data files remain encrypted and unrecoverable. The concept of file assured deletion is also termed as self-destructing data in the literature. For details about file assured deletion, readers are encouraged to see [3.34] and [3.35].

To boost the level of trust in the proposed scheme, the key generation and management is not dependent on a single *KM*. Shamir's secret sharing scheme is applied to counter any malicious *KM*. Any malicious *KM* cannot get hold of S_i independently. At least *k* number of *KMs* needs to be compromised in order to get access of enough d_i*'s* that can be used to decrypt S_i. It is also noteworthy that for decryption process S_i is sent to *KM*. However, S_i is not sent in plain as discussed in Section 5.3. The S_i is masked by multiplication with *R*. Therefore, even if malicious *KM* keeps the resultant decrypted information, the extraction of S_i will remain a challenge. Therefore, aforementioned case of malicious *KM* seems hard to be translated into successful attack. If we build a case of a malicious user that somehow has got hold of some other user's encrypted S_i, the malicious user has to go through the authentication process of at least *k* number of *KMs* to decrypt the S_i. We will see in Section 3.4.6 that *KMs* do not give access to the unauthorized users.

3.4.5. Policy Renewal

The client downloads S_i and P_i; establishes session key with the *KM*; and sends P_i, $S_i^{ei}R^{ei}$, and P_j to *KM* by session key encryption. The *KM* decrypts $S_i^{ei}R^{ei}$ to obtain S_iR and generates new public/private key pair for P_j. Therefore, the *KM* sends S_iR and new public parameters (e_j, n_j) to the client. The client extracts S_i and re-encrypts it with (e_j, n_j). Finally, the client sends P_j and encrypted S_i to the cloud. Fig. 3.13 shows the process with single *KM*. The P_i in Fig. 3.13 is older policy file while P_j is the newer policy file.

3.4.6. Analysis of DaSCE through the HLPN

We use HLPN to verify that man-in-the-middle cannot forge the encryption keys exchanged between the client and the *KM*. If the intruder intercepts the messages, then the system would be able to identify the attack. The HLPN model for DaSCE is shown in Fig. 3.14. We assume an intruder between the client and the *KM* to check the behavior of the protocol in the attack scenario. The lines in Fig. 3.14 connecting ($c1$, $c2$) and ($c3$, $c4$) would be the information flow of X_a and X_b, respectively, if there is no intruder between the client and *KM*. Due to the space limitation and for simplicity we have not given the HLPN of the whole process.

Fig. 3.14 only depicts the process of *KM* authentication. Nevertheless, the next step regarding authentication of client before exchanging keys will be the replication of the steps. Therefore, next step will have similar verification results. The associated data types and the mappings of places to data types are shown in Table 3.4 and Table 3.5, respectively. We assume an intruder between the client and the *KM* to check the behavior of the protocol in the attack scenario. The lines in Fig. 3.14 connecting ($c1$, $c2$) and ($c3$, $c4$) would be the information flow of X_a and X_b, respectively, if there is no intruder between the client and *KM*. Due to the space limitation and for simplicity we have not given the HLPN of the whole process.

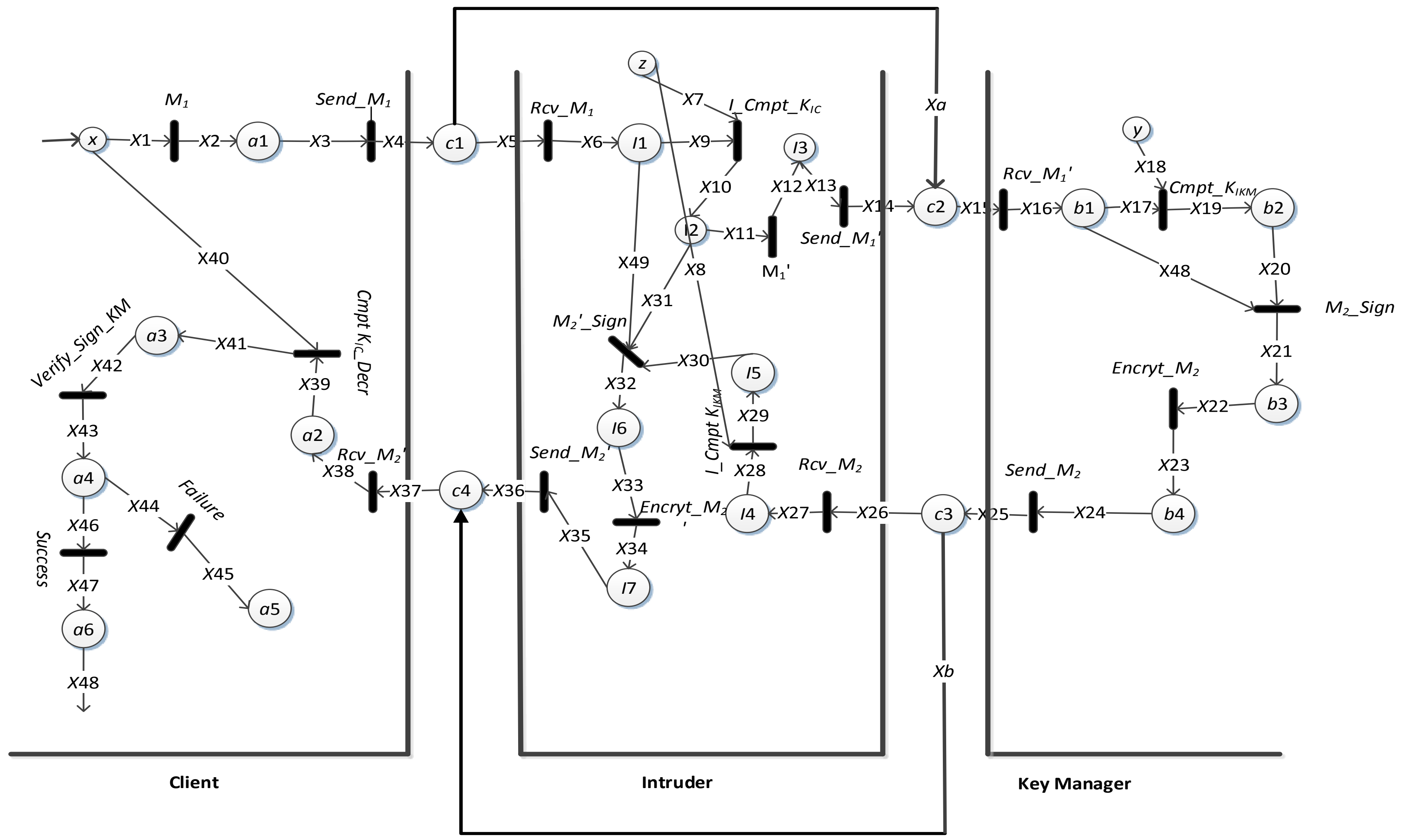


Fig. 3.14. HLPN for DaSCE.

Table 3.4. Data types for HLPN of DaSCE.

Types	**Description**
X	Big integer type random number for client
A	Big integer type number
Z	Big integer type random number for intruder
Y	Big integer type random number for key manager
M_1	Big integer type number representing α power x
M_1'	Big integer type number representing α power z
M_2	Big integer type number representing α power y
d_i	Private key of entity i from $\{Cli, Clo, KM\}$
e_i	Public key of entity i from $\{Cli, Clo, KM\}$
K_{IKM}	Session key between Intruder and Key Manager
K_{IC}	Session key between Intruder and Client
γ_s	$\{M_2, M_1'\}d_{KM}$ [M_2 and M_1' signed with d_{KM}].
γ_a	$\{ \gamma_s \} K_{IKM}$ [γ_s encrypted with K_{IKM}]
M_2'	M_1',(γ_a)K_{IC} [M_1' and γ_a encrypted with K_{IC}]
γ_i	$\{ M_1', M_1\}d_i$ [$M_1'$and M_1 signed with d_i].
γ_b	$\{ \gamma_i \} K_{IC}$ [γ_i encrypted with K_{IC}]
EM	Error Message (Message not coming from valid KM)

Table 3.5. Mapping of data types and places for HLPN of DaSCE.

Types	**Description**	**Types**	**Description**
φ(x)	$\mathbb{P}(x)$	φ(b4)	$\mathbb{P}(M_2 \times \gamma_a)$
φ(a1)	$\mathbb{P}(M_1)$	φ(c3)	$\mathbb{P}(M_2 \times \gamma_a)$
φ(c1)	$\mathbb{P}(M_1)$	φ(I4)	$\mathbb{P}(M_2 \times \gamma_a)$
φ(I1)	$\mathbb{P}(M_1)$	φ(I5)	$\mathbb{P}(K_{IKM})$
φ(I2)	$\mathbb{P}(M_1' \times K_{IC})$	φ(I6)	$\mathbb{P}(M_1' \times d_i \times \gamma_i \times K_{IKM} \times K_{IC})$
φ(I3)	$\mathbb{P}(M_1')$	φ(I7)	$\mathbb{P}(M_1' \times \gamma_b)$
φ(c2)	$\mathbb{P}(M_1')$	φ(c4)	$\mathbb{P}(M_1' \times \gamma_b)$
φ(b1)	$\mathbb{P}(M_1')$	φ(a2)	$\mathbb{P}(M_1' \times \gamma_b)$
φ(b2)	$\mathbb{P}(M_2 \times K_{IKM})$	φ(a3)	$\mathbb{P}(K_{IC} \times \gamma_b)$
φ(b3)	$\mathbb{P}(M_2 \times d_{KM} \times \gamma_s \times K_{IKM})$	φ(a4)	$\mathbb{P}(K_{IC} \times \gamma_b \times e_{KM})$

The process starts with the client requiring an upload of data to the cloud. The client generates a random number *x*, calculates its parameters (as explained in Section 3.4.2), and sends to *KM*. However, the intruder intercepts the messages. The aforementioned process is carried out at transitions M_1, $Send\_M_1$, and Rcv_M_1. The rules for these transitions are:

$$\boldsymbol{R}(M_1) = \forall\, x_1 \in X_1, \forall x_2 \in X_2 \,|\, x_2 \coloneqq pow(\alpha, x_1) \wedge \quad (3.13)$$
$$X_2' = X_2 \cup \{x_2\},$$

$$\boldsymbol{R}(Send\_M_1) = \forall\, x_3 \in X_3, \forall x_4 \in X_4 \,|\, x_4 \coloneqq x_3 \wedge \quad (3.14)$$
$$X_4' = X_4 \cup \{x_4\},$$

$$\boldsymbol{R}(Rec\_M_1) = \forall\, x_5 \in X_5, \forall x_6 \in X_6 \,|\, x_6 \coloneqq x_5 \wedge \quad (3.15)$$
$$X_6' = X_6 \cup \{x_6\}.$$

The transition $I\_Cmpt\_K_{IC}$ is fired when the intruder successfully intercepts the message that is originated for the *KM*. The intruder generates its own random number *z* and calculates a key between the client and itself. The intruder also generates fake message for the *KM* at transition M_1'and sends it to the *KM* through transition $Send\_\ M_1$'. Rules (3.16) – (3.19) are mapped to following transitions.

$$\boldsymbol{R}(I\_Cmpt\_K_{IC}) = \forall\, x_7 \in X_7, \forall x_9 \in X_9, \forall x_{10} \in X_{10} \,|\, x_{10}[1] \coloneqq pow(\alpha, x_7) \wedge \quad (3.16)$$
$$x_{10}[2] \coloneqq pow(x_9, x_7) \wedge$$
$$X_{10}' = X_{10} \cup \{x_{10}[1], x_{10}[2]\},$$

$$\boldsymbol{R}(R\_M_1') = \forall\, x_{11} \in X_{11}, \forall x_{12} \in X_{12} \,|\, x_{12} \coloneqq x_{11} \wedge X_{12}' = X_{12} \cup \{x_{12}\}, \quad (3.17)$$

$$\boldsymbol{R}(Send\_M_1') = \forall\, x_{13} \in X_{13}, \forall x_{14} \in X_{14} \,|\, x_{14} \coloneqq x_{13} \wedge \quad (3.18)$$
$$X_{14}' = X_{14} \cup \{x_{14}\},$$

$$\boldsymbol{R}(Rec\_M_1') = \forall\, x_{15} \in X_{15}, \forall x_{16} \in X_{16} \,|\, x_{16} \coloneqq x_{15} \wedge \quad (3.19)$$
$$X_{16}' = X_{16} \cup \{x_{16}\}.$$

The *KM* assuming that the message comes from the client, calculates the session key by the parameters sent by the intruder. The *KM* also signs the received and generated parameters by the private key and sends to the client that is actually received by intruder. Following transitions and rules correspond to the explained steps.

$$R(Compt\_K_{IKM}) = \forall\, x_{17} \in X_{17}, \forall x_{18} \in X_{18}, \forall x_{19} \in X_{19} | x_{19}[1] \coloneqq pow(\alpha, x_{18}) \wedge x_{19}[2] \coloneqq pow(x_{17}, x_{18}) \wedge X'_{19} = X_{19} \cup \{x_{19}[1], x_{19}[2]\}, \tag{3.20}$$

$$\boldsymbol{R}(M_2\_Sign) = \forall\, x_{20} \in X_{20}, \forall x_{21} \in X_{21}, \forall x_{48} \in X_{48} | x_{21}[1] \coloneqq x_{20}[1] \wedge x_{21}[2] \coloneqq sign(x_{20}[1], x_{48}) \wedge x_{21}[3] \coloneqq x_{20}[2] \wedge X'_{21} = X_{21} \cup \{x_{21}[1], x_{21}[2], x_{21}[3]\}, \tag{3.21}$$

$$\boldsymbol{R}(Encryt\_M_2) = \forall\, x_{22} \in X_{22}, \forall x_{23} \in X_{23} | x_{23}[1] \coloneqq x_{22}[1] \wedge x_{23}[2] \coloneqq encrypt(x_{22}[2], x_{22}[3]) \wedge X'_{23} = X_{23} \cup \{x_{23}[1], x_{23}[2], x_{23}[3]\}, \tag{3.22}$$

$$\boldsymbol{R}(Send\_M_2) = \forall\, x_{24} \in X_{24}, \forall x_{25} \in X_{25} | x_{25} \coloneqq x_{24} \wedge X'_{25} = X_{25} \cup \{x_{25}\}, \tag{3.23}$$

$$\boldsymbol{R}(Rcv\_M_2) = \forall\, x_{26} \in X_{26}, \forall x_{27} \in X_{27} | x_{27} \coloneqq x_{26} \wedge X'_{27} = X_{27} \cup \{x_{27}\}. \tag{3.24}$$

After receiving the message from the *KM*, the intruder generates the session key between the *KM* and itself. At this stage, the intruder sets up the keys with both the client and the *KM*. The intruder prepares a response for the client and sends the prepared response. The response includes the signed parameters. The intruder uses its private key for the signing purpose. The client accepts the response thinking it to be from *KM*. The following rules highlight the process.

$$\boldsymbol{R}(I\_Cmpt\_K_{IKM}) = \forall\, x_{28} \in X_{28}, \forall x_{29} \in X_{29}, \forall x_8 \in X_8 |\, x_{29} := pow(x_{28}[1], x_8) \wedge \quad (3.25)$$

$$X'_{29} = X_{29} \cup \{x_{29}\},$$

$$\boldsymbol{R}(M_2{'}\_Sign) = \forall\, x_{30} \in X_{30}, \forall x_{31} \in X_{31}, \forall x_{32} \in X_{32}, \forall x_{49} \in X_{49} | \quad (3.26)$$

$$x_{32}[1] := x_{31}[1] \wedge x_{32}[2] := sign(x_{31}[1], x_{49}) \wedge x_{32}[3] := x_{30} \wedge x_{32}[4]$$

$$:= x_{31}[2] \wedge$$

$$X'_{32} = X_{32} \cup \{x_{32}[1], x_{32}[2], x_{32}[3], x_{32}[4]\},$$

$$\boldsymbol{R}(Encr\_M_2{'}) = \forall\, x_{33} \in X_{33}, \forall x_{34} \in X_{34} |\, x_{34}[1] := x_{33}[1] \wedge \quad (3.27)$$

$$x_{34}[2] := encrypt(x_{33}[2], x_{33}[4]) \wedge$$

$$X'_{34} = X_{34} \cup \{x_{34}[1], x_{34}[2]\},$$

$$\boldsymbol{R}(Send\_M_2{'}) = \forall\, x_{35} \in X_{35}, \forall x_{36} \in X_{36} |\, x_{36} := x_{35} \wedge \quad (3.28)$$

$$X'_{36} = X_{36} \cup \{x_{36}\},$$

$$\boldsymbol{R}(Rcv\_M_2{'}) = \forall\, x_{37} \in X_{37}, \forall x_{38} \in X_{38} |\, x_{38} := x_{37} \wedge \quad (3.29)$$

$$X'_{38} = X_{38} \cup \{x_{38}\}.$$

The client after receiving the response completes the process of generating the session key. However, the key generated is between the client and the intruder, instead of being between the client and the *KM*. Following this, the client decrypts the received parameters and verifies the digital signature over them. The verification is performed using a public key of the *KM* as the client is supposedly interacting with the *KM*. The verification mechanism gives the false result and the client terminates the process. However, if there is no intruder and the communication takes place between the client and *KM*, then valid signatures will result in information flow towards the place *a6* and communication will proceed. Following are the transitions and rules for aforesaid process at the client end.

$$\boldsymbol{R}(Cmpt\_K_{IC}\_Decr) = \forall x_{39} \in X_{39}, \forall x_{40} \in X_{40}, \forall x_{41} \in X_{41} | \tag{3.30}$$

$$x_{41}[1] \coloneqq pow(x_{38}[1], x_{40}) \wedge x_{41}[2] \coloneqq decrypt(x_{38}[2], x_{41}[1]) \wedge$$

$$X'_{41} = X_{41} \cup \{x_{41}[1], x_{41}[2]\},$$

$$\boldsymbol{R}(Ver\_Sign\_KM) = \forall x_{42} \in X_{42}, \forall x_{43} \in X_{43} | x_{43} \coloneqq verifysign(x_{42}[2]) \wedge \tag{3.31}$$

$$X'_{43} = X_{43} \cup \{x_{43}\}.$$

The following properties are verified using the SMT-Lib and Z3 solver.

- During communication, if state $I1$ (see Fig. 7.14, intruder side) is achieved (that means the intruder intercepts communication), then the control will terminate at state $a5$ which represents a failure to authenticate the KM and the process terminates. The property in CTL* is represented as $AG\ (I1 \rightarrow AF\ a5)$
- If there is no intruder and communication progresses on normal course (through lines X_a and X_b in Fig. 7.13), then the control will flow until it reaches $a6$, which represents success for authentication and secure exchange of the required key. CTL* property is $AG\ (a1 \rightarrow AF(\neg I1 \cup a6))$

Both of the properties are verified in SMT-Lib using Z3 solver that approximately took 321 msec.

3.5. Implementation and Performance Evaluation

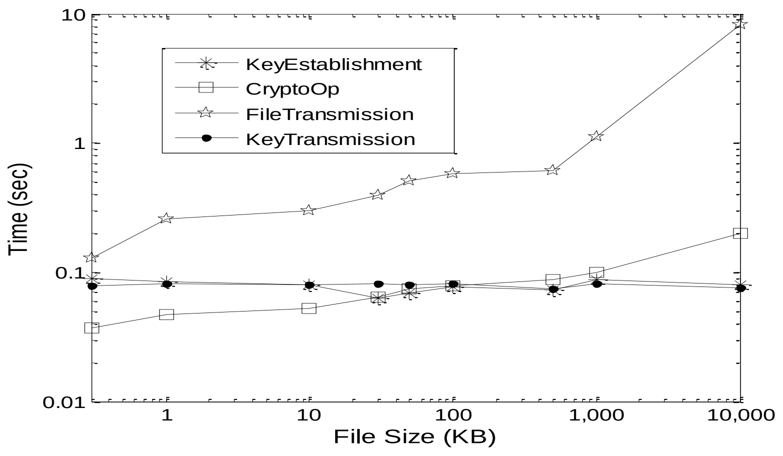
We used C# for implementing a working prototype of DaSCE. The .Net cryptographic packages were used for the involved cryptographic operations. Large prime numbers were handled by using the BigInt class. Policies were uploaded as a separate file to the cloud and the KM. The system consists of two servers (the cloud and the KM) and a client (work station). Multiple policies were combined using OR and/or AND operations. The policy and data files were not merged into a single file, to keep the policy renewal operation light weight. According to the processes described in Section 3.4, we also implemented the client side software functions, such as file upload, download, revocation, and renewal.

In our prototype, the client interacts with the *KM* (*s*) and the cloud for setting up the keys, and uploading/downloading data. The *KM* sets up the keys, revokes, and/or renews policies and manages the keys accordingly. We evaluated the DaSCE on the basis of: **(a)** Key(s) establishment time, **(b)** Key Transmission time, **(c)** File transmission time, and **(d)** Cryptographic operations time. It is noteworthy to mention that the time required for key establishment is the time for setting up a session key between the involved parties. The cryptographic operations time is the time taken by AES and MAC operations. Above given parameters collectively make up total file upload/download time. Moreover, the aforesaid parameters are evaluated using single *KM* and multiple *KMs*.

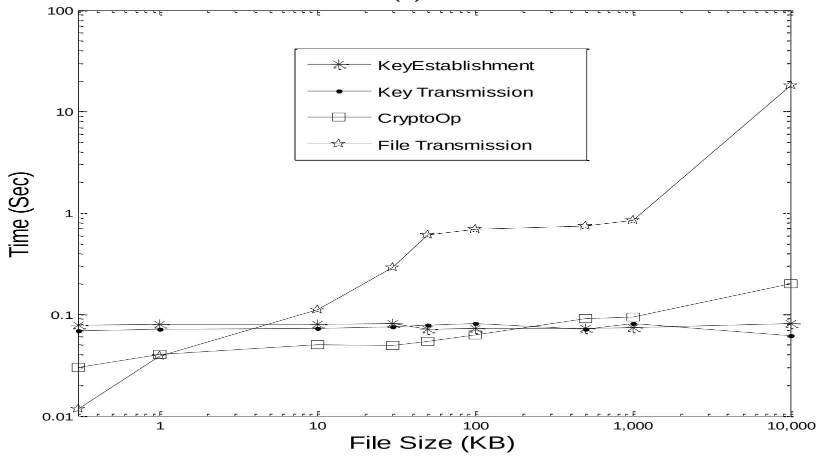
3.5.1. File Upload/Download with a Single Key Manager

We used files of nine different sizes (0.3 KB, 1 KB, 10 KB, 30 KB, 50 KB, 100 KB, 500 KB, 1 MB, and 10 MB) to measure the time consumption in file upload and download process. The results are provided in Fig. 3.15. In general, the file transmission time increased with the increase in file size. However, in some cases the change in file transmission time was small that may be caused due to network conditions at various times. Nevertheless, file transmission time was dependent on the network. In file upload case, cryptographic operations time varied between 0.037 sec and 0.201 sec. The cryptographic operations time increased with the increase in the file size. In the case of 10 MB file, the cryptographic operations time makes 2.35% of total file upload time and 2.45% of file transmission time. The time for session key establishment almost remained constant (having slight changes). The largest time taken during the key establishment was noted to be 0.0898 sec that constituted 2.67% of the total upload time.

The percentage for key establishment time was 2.39% for 10 MB file. Similarly, in case of file download operations the cryptographic operations time varied from 0.039 sec to 0.211 sec. The cryptographic operations time was dependent on the size of the file; therefore, it increased with the larger file size. However, it made lower percentage of total upload time and file transmission time. The key establishment time does not depend on the file size; therefore, it remains almost constant. Slight changes were possibly due to network transmission conditions. The DaSCE and FADE takes same amount of time for cryptographic operations. However,



(a)



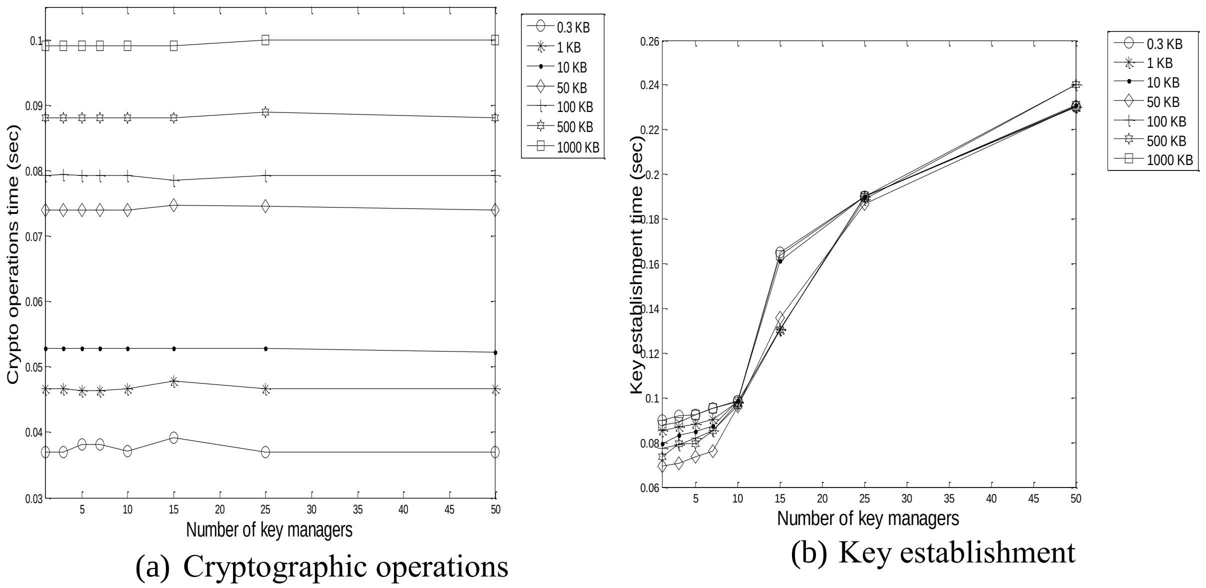
(b)

Fig. 3.15 Performance of file upload and download operations for DaSCE.

unlike FADE, we perform additional steps for key establishment in DaSCE that makes an additional overhead. Therefore, key establishment process increases the time consumption of DaSCE as compared to the protocols that run without establishing the session keys. It is noteworthy that the increase in time consumption upturns the security level for policy files, symmetric, and asymmetric keys used in the DaSCE. In the following section we will see the impact of key establishment time with increase in number of KMs.

3.5.2. File Upload/Download with Multiple Key Managers

We evaluated the performance of DaSCE by using multiple *KMs*. The file sizes we used were 0.3 KB, 1 KB, 10 KB, 50 KB, 100 KB, 500 KB, and 1MB. The number of *KMs* used was one, three, five, seven, fifteen, 25, and 50. Fig. 3.16 revealed the key establishment time and the cryptographic operation time for the aforementioned files sizes and the *KMs*. The key establishment time increased with the increase in the number of *KMs*. This is because the client had to complete all the message passing steps necessary to establish the key with all the *KMs*.



(a) Cryptographic operations

(b) Key establishment

Fig. 3.16 File upload with multiple key managers.

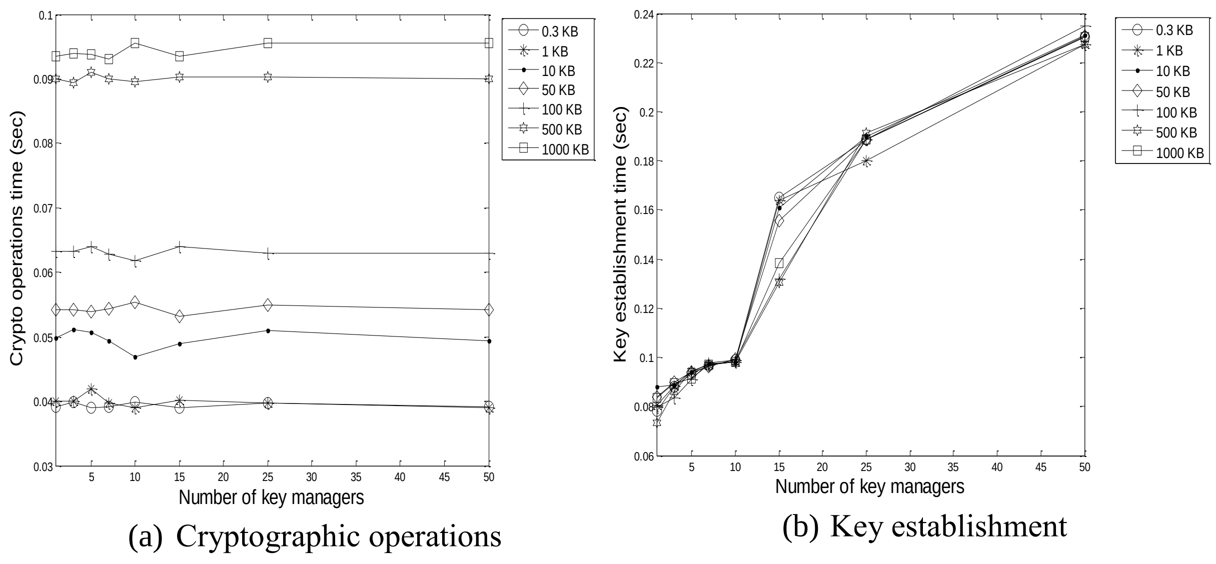
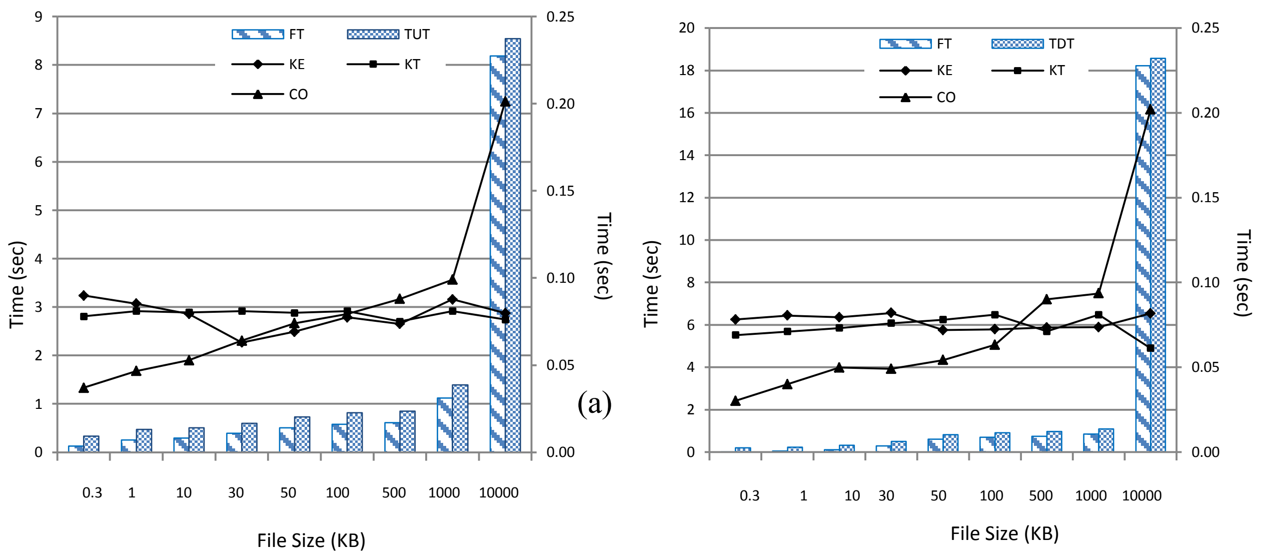


Fig. 3.17 File download with multiple key managers.

The key establishment time varies between 0.069 (single KM) seconds and 0.24 seconds (50 KMs). It must be noted that there was slight increase in the key establishment up to ten KMs. However, with higher number of KMs the increase followed a higher trend. As discussed earlier, the increase in time consumption due to key establishment augments the security level. Therefore, we say that user has to select the number of KMs judicially. A balance between tolerate able time consumption and security level in needed while deciding the number of KMs. In the coming discussion we will also see that the key establishment time constitutes low percentage of total time. The cryptographic operation time remained constant for the file of same size as final symmetric encryption is done on client with generated keys (symmetric key, *K*). Fig. 3.17 depicts the key establishment time and cryptographic operation time taken by file download with multiple *KMs*.

It must be noted that the key establishment constituted a low percentage of the total consumed time (see Fig. 3.17). Fig. 3.18 contains time comparisons of total upload/download



Upload time (b) Download time

Fig. 3.18 Total upload/download time vs key establishment time.
KE = key establishment time, KT= key transmission time, CO = cryptographic operations time, FT= file transmission time, TUT= total upload time, TDT= total download time.

time with key establishment and other constituent times for single key managers with different file sizes.

It can be noted that as the amount of data increases, the percentage of key establishment time decreases to less significant number as compared to the total upload and download time, for the file (see Fig. 3.18). Therefore, an increase in number of key managers will increase the security as well as the time consumption due to key establishment.

3.5.3. Discussion

We present DaSCE that augments the security level by introducing additional steps for the key establishment process. Because of the mutual exclusion of communication events between the client and the KM, FADE fell short on issues of securing the keys and authentication of participating parties. The DaSCE resolves the aforesaid issues by introducing digital signature for the authentication and session key establishment process before any other exchange takes place. Moreover, the concept of “assured file deletion” is used to make the file inaccessible or unrecoverable by deleting important information (d_i). Comparing the performance of FADE and DaSCE, FADE has less performance overheads as compared to DaSCE. However, unlike FADE, the DaSCE provides high security standards and does not compromise the keys under man-in-the-middle attack. It is noteworthy that DaSCE does not introduce substantial performance and monetary overhead that can lead to higher management cost. However, as compared to FADE, the performance overhead of DaSCE are slightly higher because of the supplementary steps taken to increase the level of security for the keys that upturns the security level for policy files, symmetric, and asymmetric keys used in the DaSCE.

3.6. References

[3.1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Ktaz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoics, and M. Zaharia, “A View of Cloud Computing,” *Communications of the ACM*, Vol. 53, No. 4, 2010, pp. 50-58.

[3.2] M. S. Blumenthal, “Is Security Lost in the Clouds?” *Communications and Strategies*, No. 81, 2011, pp. 69-86.

[3.3] C.Cachinand M.Schunter, "A cloud you can trust," *IEEE Spectrum*, Vol. 48, No. 12, 2011,pp. 28-51.

[3.4] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, “Controlling data in the cloud: outsourcing computation without outsourcing control,” In Proceedings of the ACM workshop on Cloud computing security,pp. 85-90, 2009.

[3.5] C. Cremers, “The Scyther Tool: Verification, falsification, and analysis of security protocols,” In Computer Aided Verification, Springer Berlin Heidelberg, 2008, pp. 414-418.

[3.6] Cloud Security Alliance, https://downloads.cloudsecurityalliance.org/initiatives/cdg/CSA_CCAQIS_Survey.pdf (accessed March 24, 2013).

[3.7] D.R. Dams, “Flat fragments of CTL and CTL*: spreading the expressive and distinguishing powers,” Logic Journal of IGPL, Vol. 17, No. 1, 1999, pp. 55-78.

[3.8] J. Desel and J.Esparza, “Free Choice Petri Nets,” Cambridge Tracts in Theoretical Computer Science, Vol. 40, Cambridge, UK: Cambridge Univ. Press, 1995.

[3.9] W. Diffie, P. C. V. Oorschot, and M. J. Wiener, “Authentication and authenticated key exchanges,” Designs, Codes and Cryptography, Vol. 2, No. 2, 1992, pp. 107-125.

[3.10] T. Dillon, C. Wu, and E. Chang, “Cloud Computing: Issues and Challenges,” In proceedings of 24th International Conference on Advanced Information Networking and Applications, pp. 27-33, 2010.

[3.11] N. En and N. Srensson, “An extensible SAT-solver,” *Lecture Notes in Computer Science*, vol. 2919, Springer, 2003, pp. 502-518.

[3.12] C P. Gomes, H. Kautz, A. Sabharwal, and B. Selman, “Satisfia-bility solvers,” *In Handbook of Knowledge Representation, Elsevier*, 2007.

[3.13] A. N. Khan, M. L. Mat Kiah, S. U. Khan, and S. A. Madani, “Towards secure mobile cloud computing: a survey,” *Future Generation Computer Systems*, Vol. 29, No. 5, 2013, pp. 1278-1299.

[3.14] A. Juels and A. Opera, “New approaches to security and availability for cloud data,” *Communications of the ACM*, Vol. 56, No. 2, 2013, pp. 64-73.

[3.15] S. Kamara and K. Lauter, “Cryptographic cloud storage,” *Financial Cryptography and Data Security,* Springer Berlin Heidelberg, 2010, pp. 136-149.

[3.16] M. Kaufman,”Data security in the world of cloud computing,” *IEEE Security and Privacy*, Vol. 7, No. 4, 2009, pp. 61-64.

[3.17] K. M. Khan, and Q. Malluhi, "Establishing trust in cloud computing," *IT professional*, Vol. 12, No. 5, 2010,pp. 20-27.

[3.18] H. Lin and W. Tzeng, “A secure decentralized erasure code for distributed network storage,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 11, Nov. 2010, pp. 1586-1594.

[3.19] H. Lin and W. Tzeng, “A secure erasure code-based cloud storage system with secure data forwarding,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 6, June 2012, pp. 995-1003.

[3.20] M. Maidl, “The common fragment of CTL and LTL,” *IEEE symposium on foundations of computer science*, pp. 643-652, 2000.

[3.21] S. U. R. Malik, S. K. Srinivasan, S. U. Khan, and L. Wang, “A Methodology for OSPF Routing Protocol Verification,” *12th International Conference on Scalable Computing and Communications (ScalCom)*, Changzhou, China, Dec. 2012.

[3.22] L. Moura and N. Bjrner, “Satisfiability Modulo Theories: An appetizer,” *Lecture Notes in Computer Science*, Vol. 5902, Springer, 2009, pp. 23-36.

[3.23] M. Mowbray, and S. Pearson, "A client-based privacy manager for cloud computing," *In Proceedings of the Fourth International (ICST) Conference on COMmunication System softWAre and middleware, ACM*, p. 5, 2009.

[3.24] T. Murata, “Petri Nets: Properties, Analysis and Applications,” *Proc. IEEE*, Vol. 77, No. 4, pp. 541-580, Apr. 1989.

[3.25] W. Reisig and G. Rozenberg, “Lectures on Petri Nets I: Basic Models,” *Lecture Notes in Computer Science*, Berlin: Springer-Verlag, Vol. 1491, 1998.

[3.26] A. Shamir, “How to Share a Secret,” *Comm. ACM*, Vol. 22, No. 11, Nov. 1979, pp. 612-613.

[3.27] D. Sun, G. Chang, L. Sun, and X. Wang, “Surveying and Analyzing Security, Privacy and Trust Issues in Cloud Computing Environments,” *Procedia Engineering*, Vol. 15, 2011, pp. 2852 – 2856.

[3.28] H. Takabi, J. B. D. Joshi, and G. J. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security and Privacy*, Vol. 8, No. 6, 2010,pp. 24-31.

[3.29] Y. Tang, P. P. Lee, J. C. S. Lui, and R. Perlman, “Secure Overlay Cloud Storage with Access Control and Assured Deletion,” *IEEE Transactions on Dependable and Secure Computing*, Vol. 9, No. 6, Nov. 2012, pp. 903-916.

[3.30] A. Yun, C. Shi, and Y. Kim, “On protecting integrity and confidentiality of cryptographic file system for outscored storage,” *Proceedings of 2009 ACM workshop on cloud computing security CCSA’09*, pp. 67-76, 2009.

[3.31] W. Jansen and T. Grance, "Guidelines on security and privacy in public cloud computing," *NIST special publication*, 800-144, 2011.

[3.32] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility,” *Future Generation computer systems*, Vol. 25, No. 6, 2009, pp. 599-616.

[3.33] A. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, “A survey of mobile cloud computing application models,” *IEEE Communications Surveys and Tutorials*, 2013, 1-21.

[3.34] R. Perlman, “File system design with assured delete,” *In Third IEEE International Security in Storage Workshop*, pp. 6, 2005.

[3.35] R. Geambasu, T. Kohno, A. A. Levy, and H. M. Levy, “Vanish: Increasing Data Privacy with Self-Destructing Data,” *In USENIX Security Symposium*, pp. 299-316. 2009.

4. DROPS: DIVISION AND REPLICATION OF DATA IN CLOUD FOR OPTICAL PERFORMANCE AND SECURITY

The paper is submitted in IEEE Transactions on Cloud Computing (TCC) and has gone through the first round of revisions. The authors of the paper are Mazhar Ali, Kashif Bilal, Samee U. Khan, Bharadwaj Veeravalli, Keqin Li, and Albert Y. Zomaya.

4.1. Introduction

Outsourcing data to a third-party administrative control, as is done in cloud computing, gives rise to security concerns [4.1]. The data compromise may occur due to attacks by other users and nodes within the cloud [4.2]. Therefore, high security measures are required to protect data within the cloud. However, the employed security strategy must also take into account the optimization of the data retrieval time [4.3]. In this paper, we propose Division and Replication of Data in the Cloud for Optimal Performance and Security (DROPS) that collectively approaches the security and performance issues. In the DROPS methodology, we divide a file into fragments, and replicate the fragmented data over the cloud nodes. Each of the nodes stores only a single fragment of a particular data file that ensures that even in case of a successful attack, no meaningful information is revealed to the attacker. Moreover, the nodes storing the fragments are separated with certain distance by means of graph T-coloring to prohibit an attacker of guessing the locations of the fragments. Furthermore, the DROPS methodology does not rely on the traditional cryptographic techniques for the data security; thereby relieving the system of computationally expensive methodologies. We show that the probability to locate and compromise all of the nodes storing the fragments of a single file is extremely low. We also compare the performance of the DROPS methodology with ten other schemes. The higher level

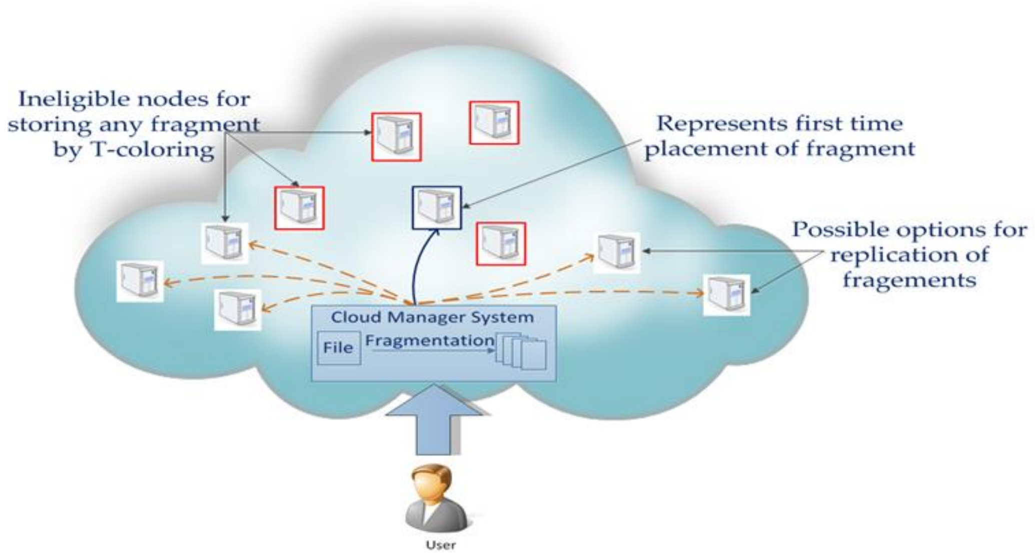

Ineligible nodes for storing any fragment by T-coloring
Represents first time placement of fragment
Possible options for replication of fragements
Cloud Manager System
File
Fragmentation
User

Fig. 4.1. The DROPS methodology.

of security with slight performance overhead was observed. The working of the DROPS methodology is shown as a high-level work flow in Fig. 4.1.

4.2. Preliminaries

Before we go into the details of the DROPS methodology, we introduce the related concepts in the following for the ease of the readers.

4.2.1. Data Fragmentation

The security of a large-scale system, such as cloud depends on the security of the system as a whole and the security of individual nodes. A successful intrusion into a single node may have severe consequences, not only for data and applications on the victim node, but also for the other nodes. The data on the victim node may be revealed fully because of the presence of the whole file [4.4]. A successful intrusion may be a result of some software or administrative vulnerability [4.4]. In case of homogenous systems, the same flaw can be utilized to target other nodes within the system. The success of an attack on the subsequent nodes will require less effort as compared to the effort on the first node. Comparatively, more effort is required for heterogeneous systems. However, compromising a single file will require the effort to penetrate only a single node. The amount of compromised data can be reduced by making fragments of a data file and storing them on separate nodes [4.4, 4.5]. A successful intrusion on a single or few nodes will only provide access to a portion of data that might not be of any significance. Moreover, if an attacker is uncertain about the locations of the fragments, the probability of finding fragments on all of the nodes is very low. Let us consider a cloud with *M* nodes and a file with *z* number of fragments. Let *s* be the number of successful intrusions on distinct nodes, such that $s > z$. The probability that *s* number of victim nodes contain all of the *z* sites storing the file fragments (represented by *P(s,z))* is given as:

$$P(s,z) = \frac{\binom{s}{z}\binom{M-s}{s-z}}{\binom{M}{s}} \tag{4.1}$$

If *M = 30*, *s = 10*, and *z = 7*, then *P(10, 7) = 0.0046*. However, if we choose *M = 50*, *s = 20*, and *z = 15*, then *P(20, 15) = 0.000046*. With the increase in *M*, the probability of a state reduces further. Therefore, we can say that the greater the value of *M*, the less probable that an attacker will obtain the data file. In cloud systems with thousands of nodes, the probability for an attacker to obtain a considerable amount of data reduces significantly. However, placing each fragment once in the system will increase the data retrieval time. To improve the data retrieval time, fragments can be replicated in a manner that reduces retrieval time to an extent that does not increase the aforesaid probability.

4.2.2. Centrality

The centrality of a node in a graph provides the measure of the relative importance of a node in the network. The objective of improved retrieval time in replication makes the centrality measures more important. There are various centrality measures; for instance, closeness centrality, degree centrality, betweenness centrality, eccentricity centrality, and eigenvector centrality. We only elaborate on the closeness, betweenness, and eccentricity centralities because we are using the aforesaid three centralities in this work. For the remainder of the centralities, we encourage the readers to review [4.6].

4.2.2.1. Betweenness Centrality

The betweenness centrality of a node *n* is the number of the shortest paths, between other nodes, passing through *n* [4.6]. Formally, the betweenness centrality of any node *v* in a network is given as:

$$C_b(v) = \sum_{a \neq v \neq b} \frac{\delta_{ab}(v)}{\delta_{ab}}, \tag{4.2}$$

Where δ_{ab} is the total number of shortest paths between a and b, and $\delta_{ab}(v)$ is the number of shortest paths between a and b passing through *v*. The variable $C_b(v)$ denotes the betweenness centrality for node *v*.

4.2.2.2. Closeness Centrality

A node is said to be closer with respect to all of the other nodes within a network, if the sum of the distances from all of the other nodes is lower than the sum of the distances of other candidate nodes from all of the other nodes [4.6]. The lower the sum of distances from the other nodes, the more central is the node. Formally, the closeness centrality of a node *v* in a network is defined as:

$$C_c(v) = \frac{N-1}{\sum_{a \neq v} d(v,a)}, \tag{4.3}$$

where *N* is total number of nodes in a network and *d(v, a)* represents the distance between node *v* and node a.

4.2.2.3. Eccentricity

The eccentricity of a node *n* is the maximum distance to any node from a node *n* [4.6]. A node is more central in the network, if it is less eccentric. Formally, the eccentricity can be given as:

$$E(v_a) = max_b d(v_a, v_b), \tag{4.4}$$

Where $\mathrm{d}(\mathrm{v_a}, \mathrm{v_b})$ represents the distance between node v_a and node v_b. It may be noted that in our evaluation of the strategies the centrality measures introduced above seem very meaningful and relevant than using simple hop-count kind of metrics.

4.2.3. T-coloring

Suppose we have a graph $G = (V, E)$ and set T containing non-negative integers including 0. The T-coloring is a mapping function *f* from the vertices of *V* to the set of non-negative integers, such that $|f(x) - f(y)| \notin T$, where $(x, y) \in E$. The mapping function f assigns a color to a vertex. In simple words, the distance between the colors of the adjacent vertices must not belong to *T*. Formulated by Hale [4.7]; the T-coloring problem for channel assignment assigns channels to the nodes, such that the channels are separated by a distance to avoid interference.

4.3. DROPS

4.3.1. System Model

Consider a cloud that consists of *M* nodes, each with its own storage capacity. Let S^i represents the name of *i*-th node and s_i denotes total storage capacity of S^i. The communication time between S^i and S^j is the total time of all of the links within a selected path from S^i to S^j represented by *c(i, j)*. We consider *N* number of file fragments such that O_k denotes *k*-th

Table 4.1. Notations and their meanings.

Symbols	**Meanings**
M	Total number of nodes in the cloud
N	Total number of file fragments to be placed
O_k	k-th fragment
o_k	Size of O_k
S^i	i-th node
s^i	Size of S^i
cen_i	Centrality measure for S^i
col_{S^i}	Color assigned to S^i
T	A set containing distances by which assignment of fragments must be separated
r_k^i	Number of reads for O_k from S^i
R_k^i	Aggregate read cost for r_k^i
w_k^i	Number of writes for O_k from S^i
W_k^i	Aggregate read cost for w_k^i
NN_k^i	Nearest neighbor of S^i holding O_k
c(i, j)	Communication cost between S^i and S^j
P_k	Primary node for O_k
R_k	Replication schema of O_k
RT	Replication time

fragment of a file while o_k represents the size of k-th fragment. Let the total read and write requests from S^i for O_k be represented by r_k^i and w_k^i , respectively. Let P_k denote the primary node that stores the primary copy of O_k. The replication scheme for O_k denoted by R_k is also stored at P_k. Moreover, every S^i contains a two-field record, storing P_k for O_k and NN_k^i that represents the nearest node storing O_k. Whenever there is an update in O_k, the updated version is sent to P_k that broadcasts the updated version to all of the nodes in R_k. Let $b(i, j)$ and $t(i, j)$ be the total bandwidth of the link and traffic between sites S^i and S^j, respectively . The centrality measure for S^i is represented by cen_i. Let col_{S^i} store the value of assigned color to S^i. The

col_{S^i} can have one out of two values, namely: open_color and close_color. The value open_color represents that the node is available for storing the file fragment. The value close_color shows that the node cannot store the file fragment. Let T be a set of integers starting from zero and ending on a pre-specified number. If the selected number is three, then $T = \{0, 1, 2, 3\}$. The set T is used to restrict the node selection to those nodes that are at hop-distances not belonging to T. For the ease of reading, the most commonly used notations are listed in Table 4.1.

Our aim is to minimize the overall total network transfer time or replication time (RT) or also termed as replication cost (RC). The RT is composed of two factors: (a) time due to read requests and (b) time due to write requests. The total read time of Ok by Si from $\mathrm{NN}_\mathrm{k}^\mathrm{i}$ is denoted by $\mathrm{R}_\mathrm{k}^\mathrm{i}$ and is given by:

$$R_k^i = r_k^i o_k c\left(i, NN_k^i\right). \tag{4.5}$$

The total time due to the writing of O_k by S^i addressed to the P_k is represented as W_k^i and is given as:

$$W_k^i = w_k^i o_k (c(i, P_k) + \sum_{(j \in R_k), j \neq i} c(P_k, j)). \tag{4.6}$$

The overall RT is represented by:

$$RT = \sum_{i=1}^{M} \sum_{k=1}^{N} (R_k^i + W_k^i) \tag{4.7}$$

The storage capacity constraint states that a file fragment can only be assigned to a node, if storage capacity of the node is greater or equal to the size of fragment. The bandwidth constraint states that $b(i,j) \geq t(i,j) \forall i, \forall j$. The DROPS methodology assigns the file fragments to the nodes in a cloud that minimizes the RT, subject to capacity and bandwidth constraints.

4.3.2. The Proposed Methodology (DROPS)

In a cloud environment, a file in its totality, stored at a node leads to a single point of failure [4.4]. A successful attack on a node might put the data confidentiality or integrity, or both at risk. The aforesaid scenario can occur both in the case of intrusion or accidental errors. In such systems, performance in terms of retrieval time can be enhanced by employing replication strategies. However, replication increases the number of file copies within the cloud. Thereby, increasing the probability of the node holding the file to be a victim of attack as discussed in Section 4.1. Security and replication are essential for a large-scale system, such as cloud, as both are utilized to provide services to the end user. Security and replication must be balanced such that one service must not lower the service level of the other.

In the DROPS methodology, we propose not to store the entire file at a single node. The DROPS methodology fragments the file and makes use of the cloud for replication. The fragments are distributed such that no node in a cloud holds more than a single fragment, so that even a successful attack on the node leaks no significant information. The DROPS methodology uses controlled replication where each of the fragments is replicated only once in the cloud to improve the security. Although, the controlled replication does not improve the retrieval time to the level of full-scale replication, it significantly improves the security.

In the DROPS methodology, user sends the data file to cloud. The cloud manager system (a user facing server in the cloud that entertains user's requests) upon receiving the file performs: (a) fragmentation, (b) first cycle of nodes selection and stores one fragment over each of the selected node, and (c) second cycle of nodes selection for fragments replication. The cloud manager keeps record of the fragment placement and is assumed to be a secure entity.

The fragmentation threshold of the data file is specified to be generated by the file owner. The file owner can specify the fragmentation threshold in terms of either percentage or the number and size of different fragments. The percentage fragmentation threshold, for instance, can dictate that each fragment will be of 5% size of the total size of the file. Alternatively, the owner may generate separate file containing information about the fragment number and size, for instance, fragment 1 of size 5,000 Bytes, fragment 2 of size 8,749 Bytes. We argue that the owner of the file is the best candidate to generate fragmentation threshold. The owner can best split the file such that each fragment does not contain significant amount of information as the owner is cognizant of all the facts pertaining to the data. The default percentage fragmentation threshold can be made a part of the Service Level Agreement (SLA), if the user does not specify the fragmentation threshold while uploading the data file. We primarily focus the storage system security in this work with an assumption that the communication channel between user and the cloud is secure.

Once the file is split into fragments, the DROPS methodology selects the cloud nodes for fragment placement. The selection is made by keeping an equal focus on both security and performance in terms of the access time. We choose the nodes that are most central to the cloud network to provide better access time. For the aforesaid purpose, the DROPS methodology uses the concept of centrality to reduce access time. The centralities determine how central a node is based on different measures as discussed in Section 4.2.2. We implement DROPS with three centrality measures, namely: (a) betweenness, (b) closeness, and (c) eccentricity centrality. However, if all of the fragments are placed on the nodes based on the descending order of centrality, then there is a possibility that adjacent nodes are selected for fragment placement. Such a placement can provide clues to an attacker as to where other fragments might be present, reducing the security level of the data. To deal with the security aspects of placing fragments, we use the concept of T-coloring that was originally used for the channel assignment problem [4.7].

We generate a non-negative random number and build the set T starting from zero to the generated random number. The set T is used to restrict the node selection to those nodes that are at hop-distances not belonging to T. For the said purpose, we assign colors to the nodes, such that, initially, all of the nodes are given the open_color. Once a fragment is placed on the node, all of the nodes within the neighborhood at a distance belonging to T are assigned close_color. In the aforesaid process, we lose some of the central nodes that may increase the retrieval time but we achieve a higher security level. If somehow the intruder compromises a node and obtains a fragment, then the location of the other fragments cannot be determined. The attacker can only keep on guessing the location of the other fragments. However, as stated previously in Section

Algorithm 4.1: Algorithm for fragment placement

```
**Inputs and initializations:**
$O = \{O_1, O_2, \dots, O_N\}$
$o = \{sizeof(O_1), sizof(O_2), \dots, \quad sizeof(O_N)\}$
$col = \{open\_color, close\_color\}$
$cen = \{cen_1, cen_2, \dots, cen_M\}$
$col \leftarrow open\_color \ \forall i$
$cen \leftarrow cen_i \ \forall i$
**Compute:**
**for each** $O_k \in O$ **do**
    select $S^i | S^i \leftarrow$ indexof (max($cen_i$))
     **if** $col_{S^i} = open\_clor \ and \ s_i \geq o_k$ **then**
$S^i \leftarrow O_k$
$s_i \leftarrow s_i - o_k$
$col_{S^i} \leftarrow close\_color$
$S^{i'} \leftarrow distance\left(S^i, T\right)$    ⊲  /* returns all nodes at distance $T$ from
node $S^i$ and stores in temporary set $S^{i'}$ */
$col_{S^i}{'} \leftarrow close\_color$
     **end if**
**end for**
```

Algorithm 4.2: Algorithm for fragment's replication

```
for each $O_k$ $in$ $O$ do
    select $S^i$ that has max$({R_k}^i + {W_k}^i)$
    if $col_{S^i} = open\_color$ $and$ $s_i \geq o_k$ then
$S^i \leftarrow O_k$
$s_i \leftarrow s_i - o_k$
$col_{S^i} \leftarrow close\_color$
$S^{i'} \leftarrow distance\ (S^i, T)$    ⊲ /* returns all nodes at distance $T$
from node $S^i$ and stores in temporary set $S^{i'}$ */
$col_{S^i}{}' \leftarrow close\_color$
    end if
end for
```

4.2.1, the probability of a successful coordinated attack is extremely minute. The process is repeated until all of the fragments are placed at the nodes. Algorithm 4.1 represents the fragment placement methodology.

In addition to placing the fragments on the central nodes, we also perform a controlled replication to increase the data availability, reliability, and improve data retrieval time. We place the fragment on the node that provides the decreased access cost with an objective to improve retrieval time for accessing the fragments for reconstruction of original file. While replicating the fragment, the separation of fragments as explained in the placement technique through T-coloring, is also taken care off. In case of a large number of fragments or small number of nodes, it is also possible that some of the fragments are left without being replicated because of the T-coloring. As discussed previously, T-coloring prohibits storing the fragment in neighborhood of a node storing a fragment, resulting in the elimination of a number of nodes to be used for storage. In such a case, only for the remaining fragments, the nodes that are not holding any fragment are selected for storage randomly. The replication strategy is presented in Algorithm 4.2.

To handle the download request from user, the cloud manager collects all of the fragments from the nodes and re-assembles them into a single file. Afterwards, the file is sent to the user.

Table 4.2. Various attacks handled by DROPS.

Attack	**Description**
Data Recovery	Rollback of VM to some previous state. May expose previously stored
Cross VM attack	Malicious VM attacking co-resident VM that may lead to data breach.
Improper media sanitization	Data exposure due to improper sanitization of storage devices.
E-discovery	Data exposure of one user due to seized hardware for investigations related to some other users.
VM escape	A malicious user or VM escapes from the control of VMM. Provides access to storage and compute devices.
VM rollback	Rollback of VM to some previous state. May expose previously stored data.

4.4. Discussion

A node is compromised with a certain amount of an attacker's effort. If the compromised node stores the data file in totality, then a successful attack on a cloud node will result in compromise of an entire data file. However, if the node stores only a fragment of a file, then a successful attack reveals only a fragment of a data file. Because the DROPS methodology stores fragments of data files over distinct nodes, an attacker has to compromise a large number of nodes to obtain meaningful information. The number of compromised nodes must be greater than *n* because each of the compromised nodes may not give fragment in the DROPS methodology as the nodes are separated based on the T-coloring. Alternatively, an attacker has to compromise the authentication system of cloud [4.8]. The effort required by an attacker to compromise a node (in systems dealing with fragments/shares of data) is given in [4.8] as:

$$E_{Conf} = \min(E_{Auth}, n \times E_{BreakIn}), \tag{4.8}$$

where E_{Conf} is the effort required to compromise the confidentiality, E_{Auth} is the effort required to compromise authentication, and $E_{BreakIn}$ is the effort required to compromise a single node. Our focus in this paper is on the security of the data in the cloud and we do not take into account the security of the authentication system. Therefore, we can say that to obtain n fragments, the effort of an attacker increases by a factor of n. Moreover, in case of the DROPS methodology, the attacker must correctly guess the nodes storing fragments of file. Therefore, in the worst case scenario, the set of nodes compromised by the attacker will contain all of the nodes storing the file fragments. From Equation (4.1), we observe that the probability of the worst case to be successful is very low. The probability that some of the machines (average case) storing the file fragments will be selected is high in comparison to the worst case probability. However, the compromised fragments will not be enough to reconstruct the whole data. In terms of the probability, the worst, average, and best cases are dependent on the number of nodes storing fragments that are selected for an attack. Therefore, all of the three cases are captured by Equation (4.1).

Besides the general attack of a compromised node, the DROPS methodology can handle the attacks in which attacker gets hold of user data by avoiding or disrupting security defenses. Table 4.2 presents some of the attacks that are handled by the DROPS methodology. The presented attacks are cloud specific that stem from clouds core technologies. Table 4.2 also provides a brief description of the attacks. It is noteworthy that even in case of successful attacks (that are mentioned), the DROPS methodology ensures that the attacker gets only a fragment of file as DROPS methodology stores only a single fragment on the node. Moreover, the successful attack has to be on the node that stores the fragment.

4.5. Experimental Setup and Results

The communicational backbone of cloud computing is the Data Center Network (DCN) [4.9]. In this paper, we use three DCN architectures namely: (a) Three tier, (b) Fat tree, and (c) DCell [4.10]. The Three tier is the legacy DCN architecture. However, to meet the growing demands of the cloud computing, the Fat tree and Dcell architectures were proposed [4.9]. Therefore, we use the aforementioned three architectures to evaluate the performance of our scheme on legacy as well as state of the art architectures. The Fat tree and Three tier architectures are switch-centric networks. The nodes are connected with the access layer switches. Multiple access layer switches are connected using aggregate layer switches. Core layers switches interconnect the aggregate layer switches.. The Dcell is a server centric network architecture that uses servers in addition to switches to perform the communication process within the network [4.10]. A server in the Dcell architecture is connected to other servers and a switch. The lower level dcells recursively build the higher level dcells. The dcells at the same level are fully connected. For details about the aforesaid architectures and their performance analysis, the readers are encouraged to read [4.9] and [4.10].

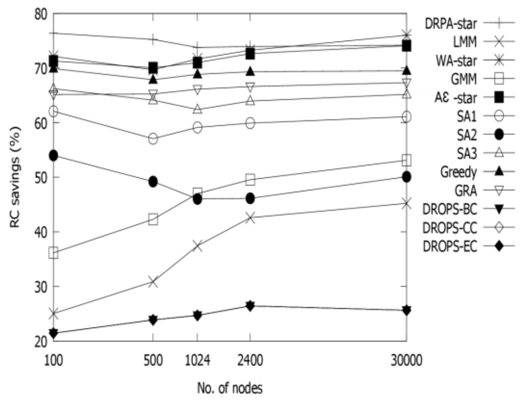
4.5.1. Comparative Techniques

We compared the results of the DROPS methodology with fine-grained replication strategies, namely: (a) DRPA-star, (b) WA-star, (c) AƐ-star, (d) SA1, (e) SA2, (f) SA3, (g) Local Min-Min, (h) Global Min-Min, (i) Greedy algorithm, and (j) Genetic Replication Algorithm (GRA). The DRPA-star is a data replication algorithm based on the A-star best-first search algorithm. The DRPA-star starts from the null solution that is called a root node. The communication cost at each node n is computed as: $cost(n) = g(n) + h(n),$ where $g(n)$ is the path cost for reaching n and $h(n)$ is called the heuristic cost and is the estimate of cost from n to the goal node. The DRPA-star searches all of the solutions of allocating a fragment to a node. The solution that minimizes the cost within the constraints is explored while others are discarded. The selected solution is inserted into a list called the OPEN list. The list is ordered in the ascending order so that the solution with the minimum cost is expanded first. The heuristic used by the DRPA-star is given as $h(n) = max(0, (mmk(n)g(n)))$, where $mmk(n)$ is the least cost replica allocation or the maxmin RC. Readers are encouraged to see the details about DRPA-star in [4.11]. The WA-Star is a refinement of the DRPA-star that implements a weighted function to evaluate the cost. The function is given as: $f(n) = f(n) + h(n) + \mathcal{E}(1 - (d(n)/D)h(n)$. The variable $d(n)$ represents the depth of the node n and D denotes the expected depth of the goal node [4.11]. The AƐ-star is also a variation of the DRPA-star that uses two lists, OPEN and FOCAL. The FOCAL list contains only those nodes from the OPEN list that have f greater than or equal to the lowest f by a factor of 1 + Ɛ. The node expansion is performed from the FOCAL list instead of the OPEN list. Further details about WA-Star and AƐ-star can be found in [4.11]. The SA1 (suboptimal assignments), SA2, and SA3 are DRPA-star based heuristics. In SA1, at level R or below, only the best successors of node n having the least expansion costs are selected. The SA2 selects the best successors of node n only for the first time when it reaches the depth level R. All other successors are discarded. The SA3 works similar to the SA2, except that the nodes are removed from OPEN list except the one with the lowest cost. Readers are encouraged to read [4.11] for further details about SA1, SA2, and SA3. The LMM can be considered as a special case of the bin packing algorithm. The LMM sorts the file fragments based on the RC of the fragments to be stored at a node. The LMM then assigns the fragments in the ascending order. In case of a tie, the file fragment with minimum size is selected for assignment (name local Min-Min is derived from such a policy). The GMM selects the file fragment with global minimum of all the RC associated with a file fragment. In case of a tie, the file fragment is selected at random. The Greedy algorithm first iterates through all of the M cloud nodes to find the best node for allocating a file fragment. The node with the lowest replication cost is selected. The second node for the fragment is selected in the second iteration. However, in the second iteration that node is selected that produces the lowest RC in combination with node already selected. The process is repeated for all of the file fragments. Details of the greedy algorithm can be found in [4.12]. The GRA consists of chromosomes representing various schemes for storing file fragments over cloud nodes. Every chromosome consists of M genes, each representing a node. Every gene is a N bit string. If the k-th file fragment is to be assigned to S^i, then the k-th bit of i-th gene holds the value of one. Genetic algorithms perform the operations of selection, crossover, and mutation. The value for the crossover rate (μ_c) was selected as 0.9, while for the mutation rate (μ_m) the value was 0.01. The use of the values for μ_c and μ_m is advocated in [4.13].The best chromosome represents the solution. GRA utilizes mix and match strategy to reach the solution. More details about GRA can be obtained from [4.13].

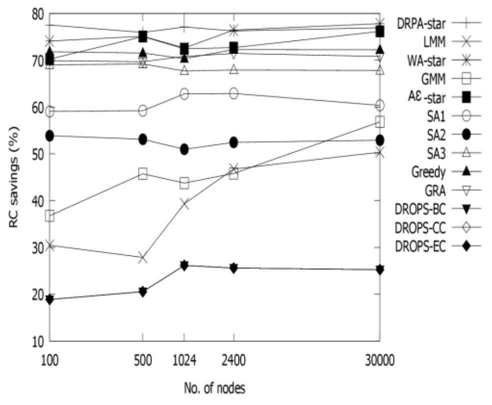
4.5.2. Workload

The sizes of files were generated using a uniform distribution between 10Kb and 60 Kb. The primary nodes were randomly selected for replication algorithms. For the DROPS methodology, the S^i's selected during the first cycle of the nodes selection by Algorithm 4.1 were considered as the primary nodes.

The capacity of a node was generated using a uniform distribution between $(\frac{1}{2}CS)C$ and $(\frac{13}{2}CS)C$, where$0 \leq C \geq 1$. For instance, for $CS = 150$ and $C = 0.6$ the capacities of the nodes were uniformly distributed between 45 and 135. The mean value of g in the OPEN and FOCAL



(a)



(b)

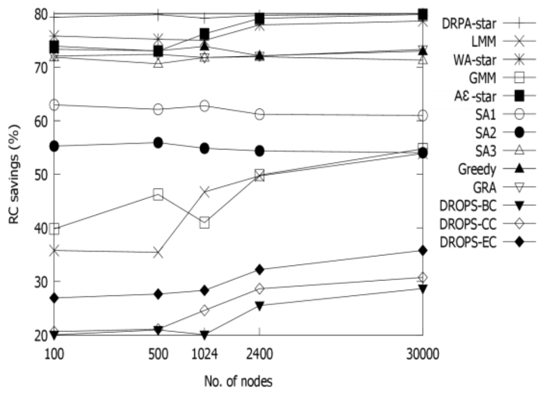
Fig. 4.2. (a) RC versus number of nodes (Three tier) (b) RC versus number of nodes (Fat tree).

lists was selected as the value of Ɛ, for WA-star and AƐ-star, respectively. The value for level *R* was set to $\left\lfloor \frac{d}{2} \right\rfloor$, where *d* is the depth of the search tree (number of fragments).

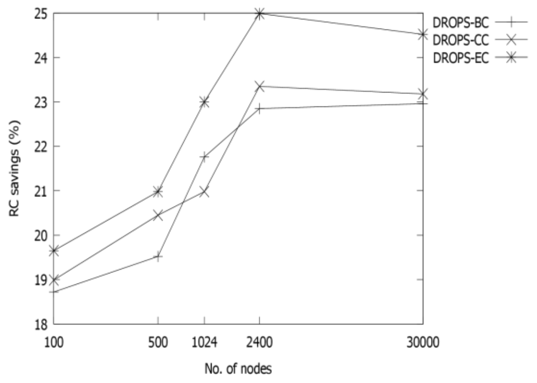
The read/write (R/W) ratio for the simulations that used fixed value was selected to be 0.25 (The R/W ratio reflecting 25% reads and 75% writes within the cloud). The reason for choosing a high workload (lower percentage of reads and higher percentage of writes) was to evaluate the performance of the techniques under extreme cases. The simulations that studied the impact of change in the R/W ratio used various workloads in terms of R/W ratios. The R/W ratios selected were in the range of 0.10 to 0.90. The selected range covered the effect of high, medium, and low workloads with respect to the R/W ratio.

4.5.3. Results and Discussion

We compared the performance of the DROPS methodology with the algorithms discussed in Section 4.5.1. The behavior of the algorithms was studied by: (a) increasing the number of nodes in the system, (b) increasing the number of objects keeping number of nodes constant, (c) changing the nodes storage capacity, and (d) varying the read/write ratio. The



(a)



(b)

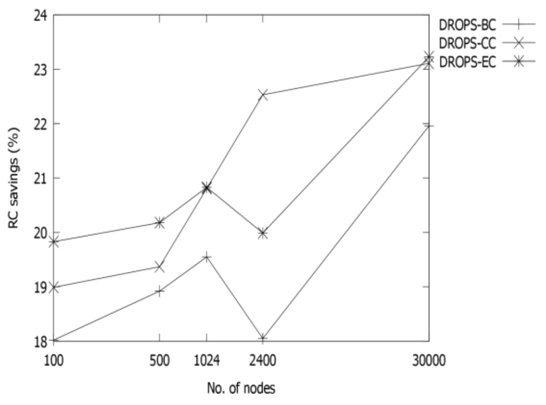
Fig. 4.3. (a) RC versus number of nodes (Dcell) (b) RC versus number of nodes for DROPS variations with maximum available capacity constraint (Three tier).

aforesaid parameters are significant as they affect the problem size and the performance of algorithms [4.11].

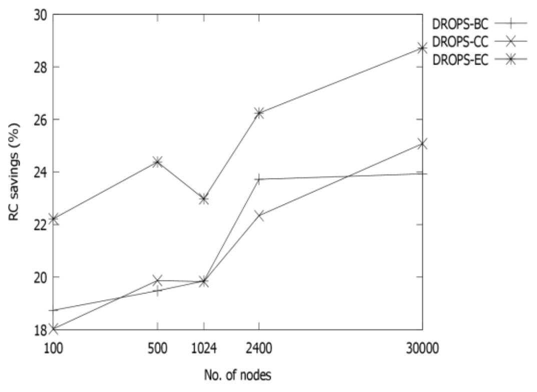
4.5.3.1. Impact of increase in number of cloud nodes

We studied the performance of the placement techniques and the DROPS methodology by increasing the number of nodes. The performance was studied for the three discussed cloud architectures. The numbers of nodes selected for the simulations were 100, 500, 1,024, 2,400, and 30,000. The number of nodes in the Dcell architecture increases exponentially [4.9]. For Dcell architecture, with two nodes in the *$Dcell_0$*, the architecture consists of 2,400 nodes. However, increasing a single node in the Dcell0, the total nodes increases to 30, 000 [4.9]. The number of file fragments was set to 50. For the first experiment we used *$C = 0.2$*. Fig. 4.2 (a), Fig. 4.2 (b), and Fig. 4.3 (a) show the results for the Three tier, Fat tree, and Dcell architectures, respectively. The reduction in network transfer time for a file is termed as RC. In the figures, the BC stands for the betweenness centrality, the CC stands for closeness centrality, and the EC stands for eccentricity centrality.

The interesting observation is that although all of the algorithms showed similar trend in performance within a specific architecture, the performance of the algorithms was better in the Dcell architecture as compared to three tier and fat tree architectures. This is because the Dcell architecture exhibits better inter node connectivity and robustness [4.9]. The DRPA-star gave best solutions as compared to other techniques and registered consistent performance with the increase in the number of nodes. Similarly, WA-star, AƐ-star, GRA, greedy, and SA3 showed almost consistent performance with various numbers of nodes. The performance of LMM and GMM gradually increased with the increase in number of nodes since the increase in the number of nodes increased the number of bins. The SA1 and SA2 also showed almost constant performance in all of the three architectures. However, it is important to note that SA2 ended up with a decrease in performance as compared to the initial performance. This may be due to the fact that SA2 only expands the node with minimum cost when it reaches at certain depth for the first time. Such a pruning for the first time might have purged nodes by providing better global access time. The DROPS methodology did not employ full-scale replication. Every fragment is replicated only once in the system. The smaller number of replicas of any fragment and separation of nodes by T-coloring decreased the probability of finding that fragment by an attacker. Therefore, the increase in the security level of the data is accompanied by the drop in performance as compared to the comparative techniques discussed in this paper. It is important to note that the DROPS methodology was implemented using three centrality measures namely: (a) betweenness, (b) closeness, and (c) eccentricity. However, Fig. 4.2(a) and Fig. 4.2(b) show only a single plot. Due to the inherent structure of the Three tier and Fat tree architectures, all of the nodes in the network are at the same distance from each other or exist at the same level. Therefore, the centrality measure is the same for all of the nodes. This results in the selection of



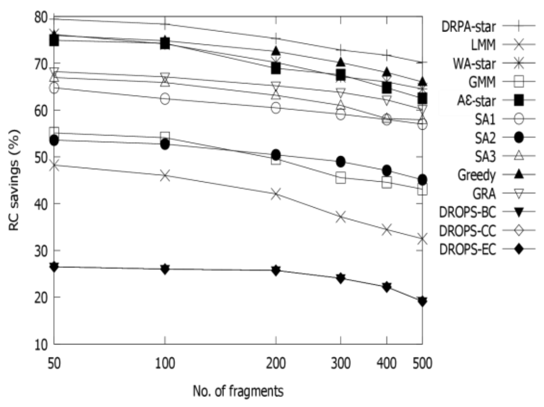
(a)



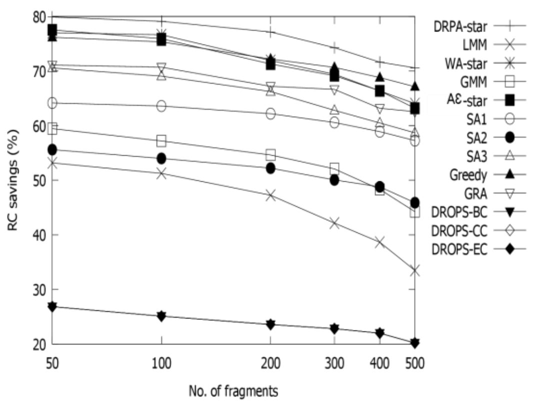
(b)

Fig. 4.4. RC versus number of nodes for DROPS variations with maximum available capacity constraints (a) Fat tree (b) Dcell.

same node for storing the file fragment. Consequently, the performance showed the same value and all three lines are on the same points. However, this is not the case for the Dcell architecture. In the Dcell architecture, nodes have different centrality measures resulting in the selection of different nodes. It is noteworthy to mention that in Fig 4.3(a), the eccentricity centrality performs better as compared to the closeness and betweenness centralities because the nodes with higher eccentricity are located closer to all other nodes within the network. To check the effect of closeness and betweenness centralities, we modified the heuristic presented in Algorithm 4.1. Instead of selecting the node with criteria of only maximum centrality, we selected the node with: (a) maximum centrality and (b) maximum available storage capacity. The results are presented in Fig. 4.3 (b), Fig. 4.4 (a), and Fig. 4.4 (b). It is evident that the eccentricity centrality resulted in the highest performance while the betweenness centrality showed the lowest performance. The reason for this is that nodes with higher eccentricity are closer to all other nodes in the network that results in lower RC value for accessing the fragments.



(a)

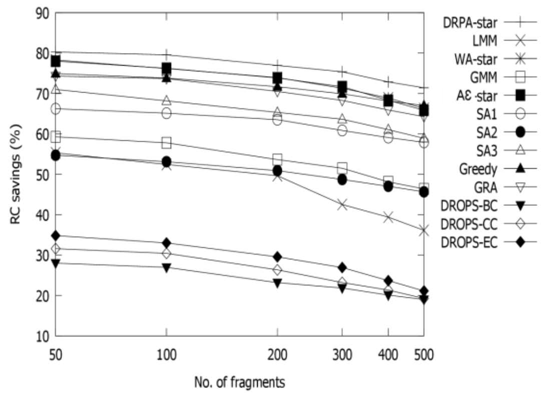


(b)

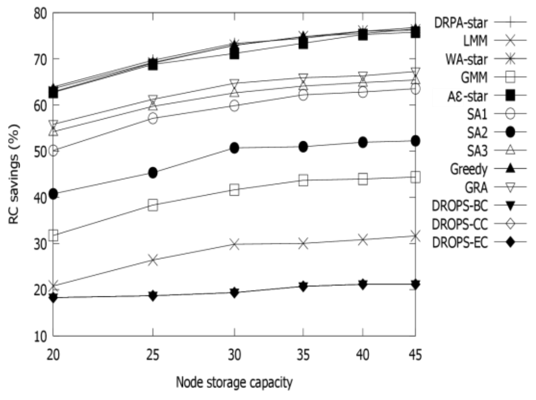
Fig. 4.5. (a) RC versus number of file fragments (Three tier) (b) RC versus number of file fragments (Fat tree).

4.5.3.2. Impact of increase in number of file fragments

The increase in number of file fragments can strain the storage capacity of the cloud that, in turn may affect the selection of the nodes. To study the impact on performance due to increase in number of file fragments, we set the number of nodes to 30,000. The numbers of file fragments selected were 50, 100, 200, 300, 400, and 500. The workload was generated with *C = 45%* to observe the effect of increase number of file fragments with fairly reasonable amount of memory and to discern the performance of all the algorithms. The results are shown in Fig. 4.5 (a), Fig. 4.5 (b), and Fig. 4.6 (a) for the Three tier, Fat tree, and Dcell architectures, respectively. It can be observed from the plots that the increase in the number of file fragments reduced the performance of the algorithms, in general. However, the greedy algorithm showed the most improved performance. The LMM showed the highest loss in performance that is little above 16%. The loss in performance can be attributed to the storage capacity constraints that prohibited the placements of some fragments at nodes with optimal retrieval time. As discussed earlier, the DROPS methodology produced similar results in three tier and fat tree architectures. However, from the Dcell architecture, it is clear that the DROPS methodology with eccentricity centrality maintains the supremacy on the other two centralities.



(a)



(b)

Fig. 4.6. (a) RC versus number of file fragments (Dcell) (b) RC versus nodes storage capacity (Three tier).

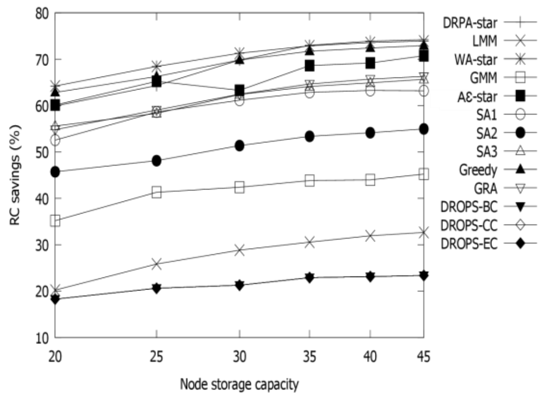
4.5.3.3. Impact of increase in storage capacity of nodes

Next, we studied the effect of change in the nodes storage capacity. A change in storage capacity of the nodes may affect the number of replicas on the node due to storage capacity constraints. Intuitively, a lower node storage capacity may result in the elimination of some optimal nodes to be selected for replication because of violation of storage capacity constraints. The elimination of some nodes may degrade the performance to some extent because a node giving lower access time might be pruned due to non-availability of enough storage space to store the file fragment. Higher node storage capacity allows full-scale replication of fragments, increasing the performance gain. However, node capacity above certain level will not change the performance significantly as replicating the already replicated fragments will not produce considerable performance increase. If the storage nodes have enough capacity to store the allocated file fragments, then a further increase in the storage capacity of a node cannot cause the fragments to be stored again. Moreover, the T-coloring allows only a single replica to be stored on any node. Therefore, after a certain point, the increase in storage capacity might not affect the performance.

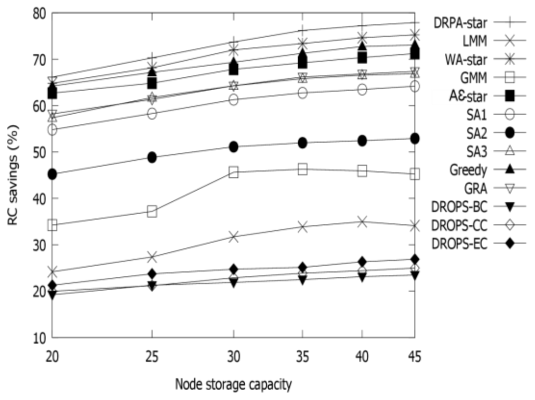
We increase the nodes storage capacity incrementally from 20% to 40%. The results are shown in Fig. 4.6 (b), Fig. 4.7 (a), and Fig. 4.7 (b). It is observable from the plots that initially, all of the algorithms showed significant increase in performance with an increase in the storage capacity. Afterwards, the marginal increase in the performance reduces with the increase in the storage capacity. The DRPA-star, greedy, WA-star, and AƐ-star showed nearly similar performance and recorded higher performance. The DROPS methodology did not show any considerable change in results when compared to previously discussed experiments (change in number of nodes and files). This is because the DROPS methodology does not go for a full-scale replication of file fragments rather they are replicated only once and a single node only stores a single fragment. Single time replication does not require high storage capacity. Therefore, the change in nodes storage capacity did not affect the performance of DROPS to a notable extent.

4.5.3.4. Impact of increase in the read/write ratio

The change in R/W ratio affects the performance of the discussed comparative techniques. An increase in the number of reads would lead to a need of more replicas of the fragments in the cloud. The increased number of replicas decreases the communication cost



(a)

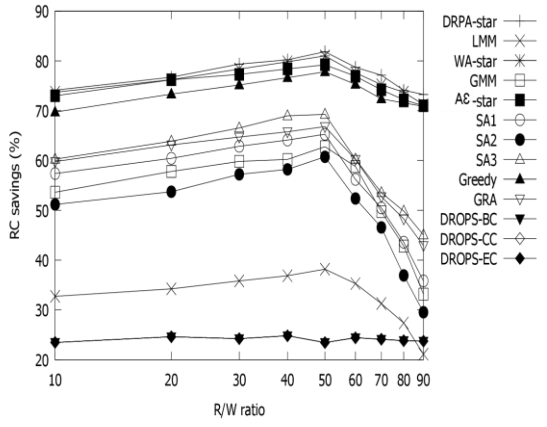


(b)

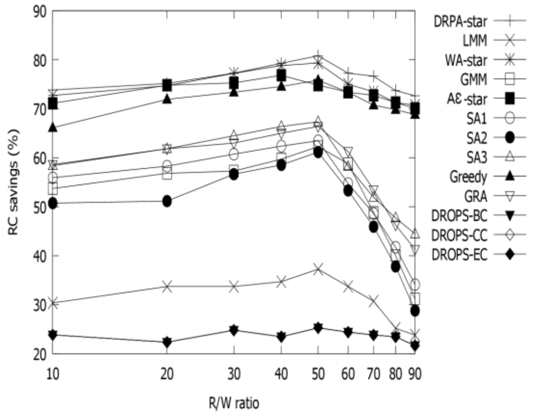
Fig. 4.7. (a) RC versus nodes storage capacity (Fat tree) (b) RC versus nodes storage capacity (Dcell).

associated with the reading of fragments. However, the increased number of writes demands that the replicas be placed closer to the primary node. The presence of replicas closer to the primary node results in decreased RC associated with updating replicas. The higher write ratios may increase the traffic on the network for updating the replicas. Fig. 4.8 (a), Fig. 4.8 (b), and Fig. 4.9 show the performance of the comparative techniques and the DROPS methodology under varying R/W ratios. It is observed that all of the comparative techniques showed an increase in the RC savings up to the R/W ratio of 0.50. The decrease in the number of writes caused the reduction of cost associated with updating the replicas of the fragments. However, all of the comparative techniques showed some sort of decrease in RC saving for R/W ratios above 0.50. This may be attributed to the fact that an increase in the number of reads caused more replicas of fragments resulting in increased cost of updating the replicas. Therefore, the increased cost of updating replicas underpins the advantage of decreased cost of reading with higher number of replicas at R/W ratio above 0.50.

It is also important to mention that even at higher R/W ratio values the DRPA-star, WA-star, AƐ-star, and Greedy algorithms almost maintained their initial RC saving values. The high



(a)



(b)

Fig. 4.8. (a) RC versus R/W ratio (Three tier) (b) RC versus R/W ratio (Fat tree).

performance of the aforesaid algorithms is due to the fact that these algorithms focus on the global RC value while replicating the fragments. Therefore, the global perception of these algorithms resulted in high performance. Alternatively, LMM and GMM did not show substantial performance due to their local RC view while assigning a fragment to a node. The SA1, SA2, and SA3 suffered due to their restricted search tree that probably ignored some globally high performing nodes during expansion. The DROPS methodology maintained almost consistent performance as is observable from the plots. The reason for this is that the DROPS methodology replicates the fragments only once, so varying R/W ratios did not affect the results considerably. However, the slight changes in the RC value are observed. This might be due to the reason that different nodes generate high cost for R/W of fragments with different R/W ratio.

As discussed earlier, the comparative techniques focus on the performance and try to reduce the RC as much as possible. The DROPS methodology, on the other hand, is proposed to collectively approach the security and performance. To increase the security level of the data, the DROPS methodology sacrifices the performance to certain extent. Therefore, we see a drop in

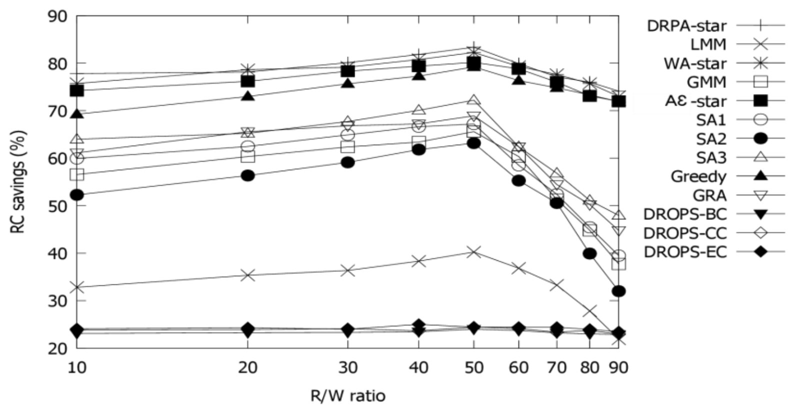


Fig. 4.9. RC versus R/W ratio (Dcell).

the performance of the DROPS methodology as compared to discussed comparative techniques. However, the drop in performance is accompanied by much needed increase in security level.

Moreover, it is noteworthy that the difference in performance level of the DROPS methodology and the comparative techniques is least with the reduced storage capacity of the nodes (see Fig. 4.6 (b), Fig. 4.7 (a), and Fig. 4.7 (b)). The reduced storage capacity proscribes the comparative techniques to place as many replicas as required for the optimized performance. A further reduction in the storage capacity will tend to even lower the performance of the comparative techniques. Therefore, we conclude that the difference in performance level of the DROPS methodology and the comparative techniques is least when the comparative techniques reduce the extensiveness of replication for any reason.

Due to the fact that the DROPS methodology reduces the number of replicas, we have also investigates the fault tolerance of the DROPS methodology. If two nodes storing the same file fragment fail, the result will be incomplete or faulty file. We randomly picked and failed the nodes to check that what percentage of failed nodes will result in loss of data or selection of two nodes storing same file fragment. The numbers of nodes used in aforesaid experiment were 500,

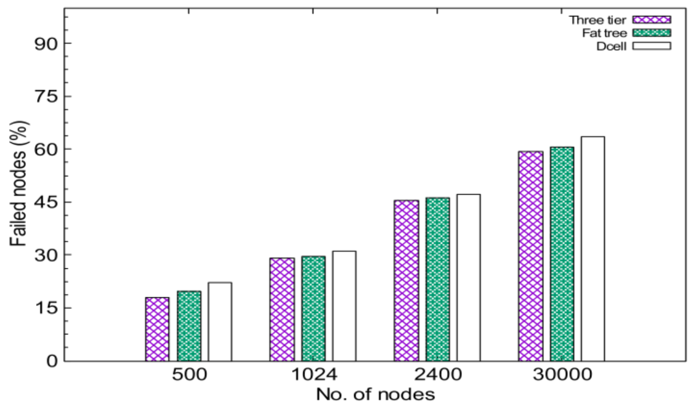


Fig. 4.10. Fault tolerance level of DROPS.

1,024, 2,400, and 30, 000. The number of file fragments was set to 50. The results are shown in Fig. 4.10. As can be seen in Fig. 4.10, the increase in number of nodes increases the fault tolerance level. The random failure has generated a reasonable percentage for a soundly decent number of nodes.

4.6. References

[4.1] K. Hashizume, D. G. Rosado, E. Fernndez-Medina, and E. B. Fernandez, “An analysis of security issues for cloud computing,” Journal of Internet Services and Applications, Vol. 4, No. 1, 2013, pp. 1-13.

[4.2] W. A. Jansen, “Cloud hooks: Security and privacy issues in cloud computing,” In 44th Hawaii IEEE International Conference on System Sciences (HICSS), 2011, pp. 1-10.

[4.3] A. N. Khan, M.L. M. Kiah, S. A. Madani, and M. Ali, “Enhanced dynamic credential generation scheme for protection of user identity in mobile-cloud computing, The Journal of Supercomputing, Vol. 66, No. 3, 2013, pp. 1687-1706.

[4.4] A. Mei, L. V. Mancini, and S. Jajodia, “Secure dynamic fragment and replica allocation in large-scale distributed file systems,” IEEE Transactions on Parallel and Distributed Systems, ol. 14, No. 9, 2003, pp. 885-896.

[4.5] M. Tu, P. Li, Q. Ma, I-L. Yen, and F. B. Bastani, “On the optimal placement of secure data objects over Internet,” In Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, pp. 14-14, 2005.

[4.6] M. Newman, Networks: An introduction, Oxford University Press, 2009.

[4.7] W. K. Hale, “Frequency assignment: Theory and applications,” Proceedings of the IEEE, Vol. 68, No. 12, 1980, pp. 1497-1514.

[4.8] J. J. Wylie, M. Bakkaloglu, V. Pandurangan, M. W. Bigrigg, S. Oguz, K. Tew, C. Williams, G. R. Ganger, and P. K. Khosla, “Selecting the right data distribution scheme for a survivable storage system,” Carnegie Mellon University, Technical Report CMU-CS-01-120, May 2001.

[4.9] K. Bilal, M. Manzano, S. U. Khan, E. Calle, K. Li, and A. Zomaya, “On the characterization of the structural robustness of data center networks,” IEEE Transactions on Cloud Computing, Vol. 1, No. 1, 2013, pp. 64-77.

[4.10] K. Bilal, S. U. Khan, L. Zhang, H. Li, K. Hayat, S. A. Madani, N. Min-Allah, L. Wang, D. Chen, M. Iqbal, C. Z. Xu, and A. Y. Zomaya, “Quantitative comparisons of the state of the art data center architectures,” Concurrency and Computation: Practice and Experience, Vol. 25, No. 12, 2013, pp. 1771-1783.

[4.11] S. U. Khan, and I. Ahmad, “Comparison and analysis of ten static heuristics-based Internet data replication techniques,” Journal of Parallel and Distributed Computing, Vol. 68, No. 2, 008, pp. 113-136.

[4.12] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, “On the placement of web server replicas,” In Proceedings of INFOCOM 2001, Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, Vol. 3, pp. 1587-1596, 2001.

[4.13] T. Loukopoulos and I. Ahmad, “Static and adaptive distributed data replication using genetic algorithms,” Journal of Parallel and Distributed Computing, Vol. 64, No. 11, 2004, pp. 1270-1285.

5. CONCLUSIONS

The cloud computing exhibits remarkable potential for providing cost effective, easy to manage, elastic, and powerful resources on the fly over the internet. The cloud computing, upsurges the capabilities of the hardware resources by optimal and shared utilization. The above mentioned features encourage the organizations and individual users to shift their data, applications and services to the cloud. Even the critical infrastructure, for example, power generation and distribution plants are being migrated to the cloud computing paradigm. However, the services provided by third-party cloud service providers entail additional security threats. The migration of users' assets (data, applications etc.) outside the administrative control in a shared environment where numerous users are collocated escalates the security concerns.

Security is one of the biggest obstacles that hamper the widespread adoption of cloud computing. Several business and research organization are reluctant in completely trusting the cloud computing to shift digital assets to the third-party service providers. The conventional IT infrastructure keeps the digital assets in the administrative domain of the organizations. All of the processing, movement, and management of data/application are performed within the organizational administrative domain. On the other hand, organizations do not enjoy administrative control of cloud services and infrastructure. The security measures taken by the cloud service providers (CSP) are generally transparent to the organizations. The presence of large numbers of users that are not related to the organizations, aggravate the concerns further. The users might be trusted by the CSP but they may not be of trust to each other. The aforementioned reasons keep the customers under uncertainties about their digital assets located at the cloud resulting in reluctance to adopt cloud computing.

Data being one of the prime assets of the organizations must be protected from all sorts of security threats. The data in the cloud is much more vulnerable to risks in terms of confidentiality, integrity, and availability in comparison to the conventional computing model. The ever increasing number of users and applications leads to enhanced security risks. In a shared environment, the security strength of the cloud equals the security strength of its weakest entity. Not only the malicious entity collocated with the victim data, but also any non-malicious but unsecure entity can result in breach of data. A successful attack on a single entity will result in unauthorized access to the data of all the users. Violation of integrity may also result from multi-tenant nature of the cloud. Employee of SaaS providers, having access to information may also act as a potential risk.

In Chapter 3, we proposed the DaSCE protocol, a cloud storage security system that provide key management, access control, and file assured deletion. Assured deletion was based on policies associated with the data file uploaded to the cloud. On the revocation of policies, access keys were deleted by the *KMs* that result in halting of the access to the data. Therefore, the files were logically deleted from the cloud. The key management was accomplished using (*k*, *n*) threshold secret sharing mechanism. We modeled and analyzed FADE. The analysis highlighted some issues in key management of FADE. The DaSCE improved the key management and authentication processes. The working of the DaSCE protocol was formally analyzed using HLPN, SMT-Lib, and Z3 solver. The performance of the DaSCE was evaluated based on the time consumption during file upload and download. The results revealed that the DaSCE protocol can be practically used for clouds for security of outsourced data.

Chapter 4 presented the DROPS methodology, a cloud storage security scheme that collectively deals with the security and performance in terms of retrieval time. The data file was fragmented and the fragments are dispersed over multiple nodes. The nodes were separated by means of T-coloring. The fragmentation and dispersal ensured that no significant information was obtainable by an adversary in case of a successful attack. No node in the cloud, stored more than a single fragment of the same file. The performance of the DROPS methodology was compared with full-scale replication techniques. The results of the simulations revealed that the simultaneous focus on the security and performance caused increased security level of data accompanied by a slight performance drop.