IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 8, Issue 6 (Jan. - Feb. 2013), PP 49-55
www.iosrjournals.org

A Secure Data Distribution Assessment for Virtual Storage Systems Using JAR file Authentication

P.Angaiyarkanni[1], C.Ramesh[2]
[1]PG Scholar, Department of CSE, Bannari Amman Institute of Technology, India
[2]Research Scholar, Department of CSE, Bannari Amman Institute of Technology, India

Abstract : *Cloud computing has the great potential to dramatically change the landscape of the current IT industry. Cloud services are provided based on user request. In cloud environment user's data are usually processed remotely in unknown machines that users do not own or operate. Control of user's data is reduced on data sharing under remote machines. Centralized monitoring applications are not suitable for highly dynamic data access environment. Data access management can be done through the cloud service providers. The Cloud Information Accountability framework used to keep track of the actual usage of the users' data in the cloud. It combines the aspects of access control, usage control and authentication. The data are sending along with access control policies and logging policies enclosed in JAR files, to cloud service providers. Current framework does not provide authentication and integrity.*

The proposed system focus on the accountability framework to be improved to provide authentication scheme for JAR files and combines the data and runtime integrity verification method. Log data analysis is provided with indexing and aggregation functions. The system includes the data and executable access control model.

Keywords- *Cloud computing, Accountability, CIA Framework, JAR Files, Data sharing.*

I. Introduction

Cloud computing presents a new approach to increase the current consumption and delivery model for real IT services based on the user request through internet. It provides dynamically scalable and oftenly virtualized resources as a service over the Internet on demand .To date there are number of notable commercial and individual services , including Amazon, Google, Microsoft, Yahoo, and Sales force [6]. Details of the services provided are abstracted from the users who no longer need to be experts of technology transportation. In addition cloud users may not know the machines which actually process and retrieve their information. By this new technology, users also start perturbing about manage of their personal data. The information processed on clouds are frequently outsourced, and also the number of issues are related to accountability, including the handling of personally articulate information. Such uncertainties are apt to the spacious espousal of cloud services [10].

The design of the CIA framework presents generous challenges, including excellently identifying the cloud service provider (CSP), consistency of the log files, and adapting to a highly decentralized communications etc. Our basic approach towards addressing these issues is to force and widen the programmable facility of JAR (Java ARchives) files to automatically maintain a record of the usage of the users' data by any entity around the cloud. The end users will send their own data along with any policies such as access control policies and logging policies that they crave to implement, together with JAR files, towards cloud service providers. Any admittance to the information will trigger an mechanical and legitimate logging method limited to the JARs. We refer to this type of enforcement as "strong binding", while the policies and the logging mechanism travel among the data. This strong binding comes even when copies of the JARs are created; thus, the user will have power over his data at any site. Such decentralized logging method met the forceful nature of the cloud but also imposes challenges on ensuring the reliability of the logging.

To cope with this issue, we provide the JARs with a central point of contact which forms a link between cloud service provider and the actual user. It records the error correction information sent via the JARs, which allows it to monitor the loss of any logs from any of the JAR files. Moreover, if a JAR is not able to contact its center point, any admission to its enclosed data will be left.

Currently, we focus on image files since images represent a very common content type for end users and organizations and are increasingly hosted in the cloud as part of the storage services offered by the utility computing paradigm featured by cloud computing [1]. Further, images are often exposed to social and personal behavior of users, or archiving important files from organizations. In aiding to, our aim can handle personal identifiable information provided they are stored as image files.

We have made the following new attempts in our approach. First, we integrated integrity checks and oblivious hashing (OH) technique to our system in order to strengthen the dependability of our system in case of consoled JRE. We also updated the log records structure to provide additional guarantees of integrity and authentication. Second, we extended the security analysis to cover more possible attacked scenarios. Third, we report the results of new experiments and provide a thorough evaluation of the system performances. Fourth, we have added a detailed discussion on related works to prepare readers with a better understanding of background knowledge. At last, we have improved the presentation by adding more examples and illustration graphs **.**

II. Related Work

With respect to Java-based techniques for security, our approach is related to Self Defending Objects. Self-defending objects(SDO) are an extension of the object-oriented programming model, where software objects that produce the sensitive functions or hold sensitive data are responsible for protecting those data. we also expand the concepts of object-oriented programming. The key variation in our implementations is that the authors still rely on a centralized database to maintain all processing records, while the items being protected are held as separate files. In preceding work, we introduce the Java-based approach to prevent privacy leakage from indexing [9], which could be integrated with the CIA framework proposed in this work since they build on related architectures.

In terms of authentication techniques, Appel and Felten introduced the concept of Proof-Carrying authentication structure. The PCA includes a high order logic language that allows quantification over logic, and focuses on access control for web services. While related to our research, it helps for maintaining safe and security, high performance, mobile code, the PCA objective is highly different from our survey, as it focuses on validating code, rather than monitoring the information. Another work done by Mont et al.,Proposed the concept of strong coupling content with access control, using Identity-Based Encryption (IBE). We also persuade IBE techniques, but in a very different manner. We do not rely on IBE to attach the content with the rules. Instead, we use it to afford strong guarantees for the encrypted content and the log files, such as security against chosen plaintext and ciphertext attacks.

In addition, our work may look similar to works on secure data provenance [5], but in fact greatly differs from them in terms of objective , techniques, and application domains. Data provenance aims to guarantee data integrity by securing the data. They ensure that no one can add or remove entries in the middle of a provenance chain without perception, so that data are correctly delivered to the end user . Differently, our work is to provide data accountability concept, to monitor the usage of the data and ensure that any access to the data is tracked. Since it is in a distributed surroundings, we also log where the data goes. However, the data integrity is confirmed by the receivers using specific policies instead of sender.

Along the lines of extended content protection, usage control [3] is being investigated as an extension of current access control performance. Current efforts on usage control are primarily focused on conceptual analysis of usage control requirements and on languages to express constraints at various level of granularity. While some notable results have been achieved in this respect [4], thus far, there is no concrete contribution addressing the problem of usage constraints enforcement, especially in distributed settings. The few existing solutions are partial, restricted to a single domain, and often specialized [7]. Finally, general outsourcing techniques have been investigated over the past few years [2]. Although only [8] is particular to the cloud environment, some of the outsourcing protocols may also be applied in this domain. In this work, we do not cover issues of data storage security which are a complementary aspect of the privacy issues.

III. Cloud Information Accountability Framework

The Cloud Information Accountability framework proposed in this paper .It performs automated logging and distributed auditing mechanisms of relevant access performed by any entity and carried out at any point of time at any cloud service provider. CIA has two major elements: logger and log harmonizer.

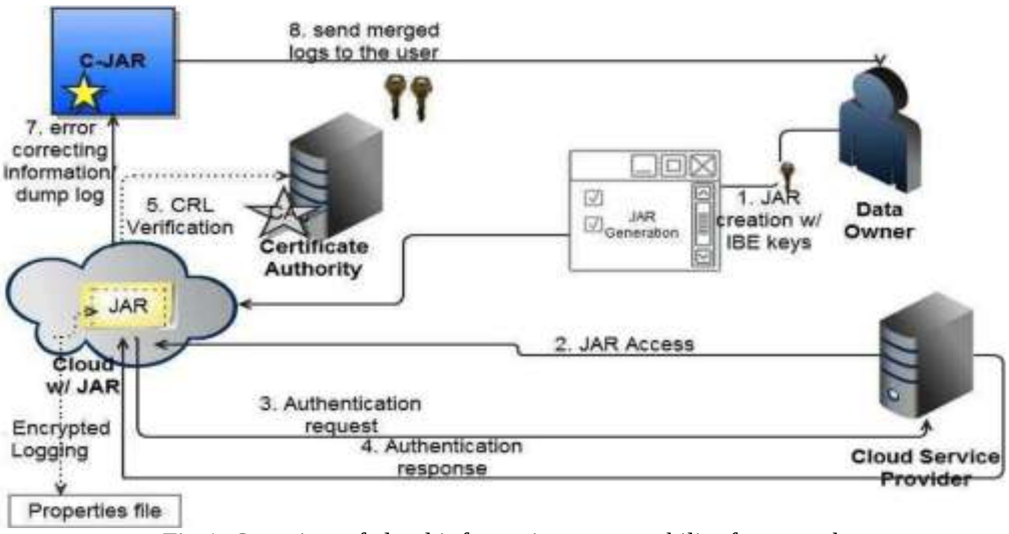

Fig 1: Overview of cloud information accountability framework

The logger is the element which is strongly coupled with the individual data, so that it is downloaded when the data are entree, and is copied whenever the data are copied. It handles a particular occurrence or copy of the user's data and is responsible for logging access to that instance or copy.

The log harmonizer forms the central part which allows the user access to the log files. The logger is strongly coupled with user's data. Main task of log harmonizer is automatically logging access to data items that contains, encrypting the log record using the public key of the content owner, and sending them to the log harmonizer at regular intervals. It may also be configured to ensure that access and usage control policies associated with the data are honored. For example, a data provider can specify that user X is only allowed to view the data but not to change the data. The logger will control the data access even after it is downloaded by user X.

The logger requires only minimum support from the server in order to distribute the data.The tight coupling between data and logger, provides high distributed logging system .Thus the logger does not need to be installed on any system or it should requires any special support from the server,but it is not very intrusive in its actions for satisfying further requirements. Finally, the loggers should responsible for generating the error correction information for each log record and finally send to the log harmonizer.Thus the error correction information combined with the encryption and authentication mechanism provides a robust and reliable recovery mechanism.

The log harmonizer is responsible for auditing. Being the trusted component, the log harmonizer generates the skeleton key. It holds on to the decryption key for the IBE key pair, and it is responsible for decrypting the logs. Instead, the decryption can be carried out on the client end if the path between the log harmonizer and the client is not trusted manner. In this case, the harmonizer sends the key to the client in a secure key exchange.

Two auditing strategies are processed in accountability: they are push and pull mode. In push strategy, the log file is pushed back to the data owner periodically in an automated manner. Pull mode is an on-demand approach, in this the log file is obtained by the data owner as often as requested. These two auditing modes allow us to satisfy the aforementioned fourth design requirement. If suppose it exist in multiple loggers for the same set of data items, thus the log harmonizer will merge log records from them before sending back to the data owner. The log harmonizer make it responsible for handling log file corruption. Also, the log harmonizer can itself carry out logging in addition to auditing. The logging and auditing functions improves the over all performance. The logger and the log harmonizer are implemented as lightweight and portable JAR files. This file implementation provides automatic logging functions, thus it should met the second design requirement.

IV. Automated Logging Mechanism

In this part, we first elaborate on the automated logging mechanism and then present techniques to guarantee dependability.

4.1. Logger Structure

We should leverage the programmable capability of JAR file to conduct the automated logging. The logger component is a nested Java JAR file which should stores a user's data items and corresponding log files. Thus our proposed JAR file consists of one outer JAR enclosing one or more inner JARs.

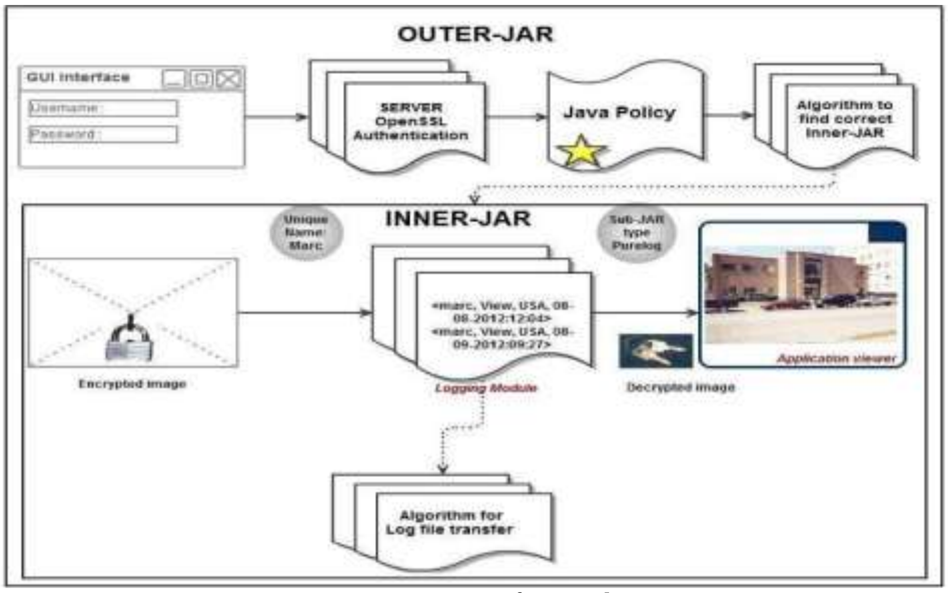
OUTER-JAR
GUI interface
Username:
Password:
SERVER OpenSSL Authentication
Java Policy
Algorithm to find correct Inner-JAR
INNER-JAR
Unique Name: Marc
Sub-JAR type Purelog
<marc, View, USA, 08-08.2012:12:04>
<marc, View, USA, 08-09.2012:09:27>
Encrypted image
Logging Module
Decrypted image
Application viewer
Algorithm for Log file transfer

Fig 2: Structure of JAR File

The main responsibility of the outer JAR is to handle authentication of entities which want to access the data stored in the JAR file. Each inner JAR file contains the encrypted data, for that the class files should facilitate the retrieval of log files and should displays the enclosed data in a suitable format, and a log file for each encrypted items. Thus we support two options:

- PureLog:It should record every access to the data and is mainly used for pure auditing records.
- AccessLog : It follows two functions that as logging actions and enforcing access control. Thus the JAR file will record the time when the request is made. If the access request is granted, then the JAR will additionally record the access information along with the duration for which the access is allowed.

The two kinds of logging modules allows the data owner to enforce certain access conditions either proactively or reactively. For example, services like billing may just need to use PureLogs.Thus the Access Logs will be necessary for services which should need to enforce service-level agreements that as limiting the visibility to some sensitive content at a given location.

For that purpose we should carry out the functions, the inner JAR contains a class file for writing the log records, another class file should corresponds with the log harmonizer, the encrypted data, one more class file for displaying or downloading the data and the public key of the IBE key pair that is necessary for encrypting the log records. For that no secret keys should ever stored in the system. Thus the outer JAR may contain one or more inner JARs, in addition to that we should use the class file for authenticating the servers or the users, another class file used for finding the correct inner JAR, another class file which checks the JVM's validity using oblivious hashing. Furthermore, a class file is used for managing the GUI for user authentication and the Java Policy.

4.2 Log Record Generation

Log records are generated by the logger component. Logging process occurs at any access to the data in the JAR files, and new log entries are appended sequentially, in order of creation LR = $<r_1, \ldots, r_k>$. Each record ri is encrypted individually and appended to the log file. In particular, a log record is in the following form:

r_i = <ID, Act, T, Loc, h((ID, Act, T, Loc)|$r_i - 1$| . . . |r_1), sig>: Here, r_i indicates that an entity identified by ID has performed an action Act on the user's data at time T at location Loc. The component h((ID, Act, T, Loc)|$r_i - 1$| . . . |r_1) corresponds to the checksum of the log records preceding the newly one inserted, concatenated with the main content of the record itself. The checksum is computed using a collision-free hash function. The component sig denotes the signature of the log record created by the server. Suppose more than one file is handled by the same logger, an additional ObjID field is added to each record.

4.3 Dependability of Logs

In this part, we discuss how we guarantee the dependability of logs. Specifically, our aim to prevent the following types of attacks. First, an attacker may try to avoid the auditing mechanism by storing the JAR files remotely, corrupting the JAR, or trying to prevent them from communicating with the end user. Second, the attacker may try to compromise the JRE used to run the JAR files.

V. Data Auditing Schemes

In this part, we describe distributed auditing mechanism including the algorithms for data owners to query the logs regarding their data.

5.1. Push and Pull Mode

To permit the users to be timely and accurately informed about our data usage, our distributed logging mechanism is complemented by an new auditing mechanism. It supports two auditing modes: 1) push mode; 2) pull mode.

In push mode, the logs are sporadically pushed to the data owner by the log harmonizer. This push action will be triggered by either type of the following two events: one is that the time elapses for a certain amount of period according to the temporal timer inserted as part of the JAR file; the other is that the JAR file exceeds the size is stipulated by the content owner at the time of log creation. After the logs are sent to the data owner, the files will be dumped, then free the space for further future access logs. The error correcting information is also dumped with log files.

Also push mode is the basic type which can be adopted by both the PureLog and the AccessLog, moreover of whether there is a request from the data owner for the log files. It performs two essential functions in the logging architecture: 1) it ensures that the size of the log files does not explode and 2) it enables timely detection and correction of any loss or damage to the log files.

The pull mode allows auditors to retrieve the logs anytime when they want to check the recent admittance to their own data. The pull message is an simply FTP pull command, which is issued from command line. On behalf of naive users, a wizard comprising a batch file can be easily built. The request message will be sent to the harmonizer, then user will be informed of the data's locations and obtain an integrated copy of the authentic and sealed log file.

5.2 Log Retrieval Algorithm

Pushing or pulling auditing strategies have fascinating tradeoffs in accountability mechanisms. Pushing strategy is favorable when there are a large number of accesses to the data within a short period of time. Suppose, the data are not pushed out frequently enough, the log file may become very large in size , it increases the cost of operations like copying data. This pushing mode may be preferred by data owners because, who are the owner of organizations and should to keep track of the data usage constantly over time. Such data owners, receiving the logs automatically can diminish the load of the data analyzers. The maximum size of logs are pushed out is a parameter which can be easily configured as creating the logger component. Pull strategy is needed only when the data owner suspects some misuse of his data; the pull mode allows him to monitor the usage of his content immediately. A hybrid approach can actually be implemented to benefit of the consistent information offered by pushing mode and the convenience of the pull mode. Supporting both pushing and pulling modes helps protecting from some nontrivial attacks.

Require: *size:* maximum size of the log file specified by the data owner, *time:* maximum time allowed to elapse before the log file is dumped, *tbeg:* timestamp at which the last dump occurred, *log:* current log file, *pull:* indicates whether a command from the data owner is received.

```
1.  Let TS(NTP) be the network time protocol timestamp
2.  When pull = 0
3.  rec : = □ UID, OID, Access Type, Result, Time, Loc□
4.  curtime : =TS(NTP)
5.  l size : = size of (log) / / current size of the log file
6.  if (( cutime –tbeg) < time) &&
          ( lsize < size ) && (pull = = 0) then
7.  log : = log + ENCRYPT (rec) / /  ENCRYPT is the encryption function used to encrypt the log record
8.  PING to CJAR / / send a PING to the log  harmonizer to check if it is alive
9.  if PING –CJAR then
10.     PUSH RS (rec) / / write the error correcting bits
11.  else
12.     EXIT (1) / /  error if no PING is  received
13.  end if
14.  end if
15.  if ((cutime – tbeg ) > time ||(l size > = size ) ||(pull □ 0 ) then
16.     / / Check if PING is received
17.  if PING –CJAR then
18.    PUSH log  //write the log file to the harmonizer
19.    RS(log) : = NULL / / reset the error correction records
```

```
20.    tbeg : = TS (NTP) / / reset the tbeg variable
21.    pull : = 0
22.  else
23.    EXIT (1) / / error if  no PING is received
24.  end if
25.  end if
```

Fig 3: Push and Pull mode

The log retrieval algorithm for the Push and Pull modes is outlined in Figure.1. This algorithm presents logging and synchronization steps with log harmonizer in case of PureLog. First, this algorithm checks whether the size of the JAR file has exceeded a stipulated size or the normal time of two consecutive dejection has elapsed. The size and time threshold for a dejection are specified by the data owner at the time of creation of the JAR files. The algorithm also checks whether the data owner has requested a dejection of the log files. If nothing actions has occurred, it proceeds to encrypt the log record and write the error-correction information to the log harmonizer.

The communication of log harmonizer begins with a simple handshake. If there is no response is received, the log report the error information. The data owner is then alerted through e-mails, if the JAR is configured to mail the error notifications. Once the handshake is finished, the communication of log harmonizer is proceeds, with the help TCP/IP protocol. If any of the aforementioned events are occur, the JAR's simply dumps the log files and resets all the variables, to create space for new records. In case of AccessLog, the above algorithm is modified by adding an additional check. Precisely, the AccessLog checks whether the CSP accessing the log satisfies all the conditions specified in the policies references with it. If all the circumstances are satisfied, access is established; otherwise, access is denied. Irrespective of the access control outcome, the attempted access of data in the JAR file will be logged.

This auditing mechanism have two main advantages. First, it guarantee the high level of availability of the log files. Second, the use of the harmonizer minimizes the amount of workload for human users in going through long log files sent by different copies of JAR files.

VI. Secure Data Distribution for VMs

The system is designed to perform data center management and access control activities. Decentralized access control monitoring is provided in the system. Object based access monitoring is performed for the data owners. The system is divided into six major modules. They are data owner, cloud data center, client, JAR authentication, security and access control and attack verification.

Data owner share the data files under the cloud environment. Data center maintains the shared data for the data owner. Cloud client downloads and access the shared data from the data centers. JAR(Java ARchieve) files are used to monitor the data access under the clients. Code and data privilege mechanism are used for the security process. Attack properties and shared data files are protected from attackers.

6.1 Data Owner

The data owner shares the data files to the clients. Data files are provided with different access permissions. Access permissions are assigned by the data owner based on the user group. The system is designed with multiple data owners.

6.2 Cloud Data Center

The cloud data center provides storage spaces for the cloud users. Shared data files provided by data owners are uploaded to the cloud data centers. Client requests are processed by the data centers. Access logs are maintained under the cloud data centers

6.3 Client

The client application is designed to access the data files under the cloud environment. The data owner assigns the client access levels. Data files are provided with reference to the access levels. Client collects the data files from the data centers.

6.4 JAR Authentication

The JAR files are distributed from the data centers with the data files. The classes in the JAR components are authenticated by the data centers. The JAR execution is initiated after the access verification process. Authentication methods are used to control anonymous JAR component access.

6.4 Security and Access control

The security and access control methods are used to verify the JAR components. Data access levels are monitored and verified with client permissions. Client monitoring codes are provided with different access levels. Access level based functions are integrated in the monitoring component.

6.4 Attack Verification

The attack verification is carried out with integrity checking methods. Data and runtime integrity checking methods are used in the system. The data integrity verification is used to check the data transmission process. The runtime verification is performed to verify the code execution process.

VII. Conclusion

The data centers are used to share the data around cloud nodes. Cloud Information Accountability (CIA) framework is used to perform data access monitoring process. The CIA model is enhanced with authentication and integrity analysis models. The system security is ensured with data, executable access control mechanism, provides decentralized auditing model. Accountability monitoring is carried out under the usage environment. Policy based model integrates security and accounting process. Also it provides platform independent accountability management model. By the analysis, it is shown as better choice for data accessibility.

References

[1] Smitha Sundareswaran, Anna C. Squicciarini, and Dan Lin, “Ensuring Distributed Accountability for Data Sharing in the Cloud”. IEEE Transactions on Dependable And Secure Computing, Vol. 9, No. 4, July/August 2012.
[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, “Provable Data Possession at Untrusted Stores,” Proc. ACM Conf. Computer and Comm. Security, pp. 598-609, 2007.
[3] A. Pretschner, M. Hilty, F. Schuo¨ tz, C. Schaefer, and T. Walter, “Usage Control Enforcement: Present and Future,” IEEE Security & Privacy, vol. 6, no. 4, pp. 44-53, July/Aug. 2008.
[4] A. Pretschner, F. Schuo tz, C. Schaefer, and T. Walter, “Policy Evolution in Distributed Usage Control,” Electronic Notes Theoretical Computer Science, vol. 244, pp. 109-123, 2009.
[5] R. Hasan, R. Sion, and M. Winslett, “The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance,” Proc. Seventh Conf. File and Storage Technologies, pp. 1-14, 2009.
[6] P.T. Jaeger, J. Lin, and J.M. Grimes, “Cloud Computing and Information Policy: Computing in a Policy Cloud?,” J. Information Technology and Politics, vol. 5, no. 3, pp. 269-283, 2009.
[7] F. Martinelli and P. Mori, “On Usage Control for Grid Systems,” Future Generation Computer Systems, vol. 26, no. 7, pp. 1032-1042, 2010.
[8] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, “Enabling Public Verifiability and Data Dynamics for Storage Security in Cloud Computing,” Proc. European Conf. Research in Computer Security (ESORICS), pp. 355-370, 2009.
[9] A. Squicciarini, S. Sundareswaran, and D. Lin, “Preventing Information Leakage from Indexing in the Cloud,” Proc. IEEE Int’l Conf. Cloud Computing, 2010.
[10] S. Pearson and A. Charlesworth, “Accountability as a Way Forward for Privacy Protection in the Cloud,” Proc. First Int’l Conf. Cloud Computing, 2009.

Bibliography

Ms. P.Angaiyarkanni received **her B.TECH IT Institute of Road and Transport Technology, Cithode** and she is now pursuing her ME. in Computer Science & Engineering at Bannari Amman Institute of Technology, Sathyamangalam, Erode, Tamil Nadu, India. She had participated in Various National Level Technical Symposium held at Engineering Colleges in Tamil Nadu and got many Prizes. She had published 2 Papers in Various National and International Conferences held at Engineering Colleges. She had attended more than 5 workshops and seminars in diverse disciplines held at various Engineering Colleges.

Prof. C.Ramesh received his M.E in Computer Science & Engineering and is now pursuing his Ph.D. in cloud computing at Anna University of Technology, Coimbatore. Currently, he is working as Assistant Professor (Senior Grade) in the department Computer Science and Engineering, Bannari Amman Institute of Tech, Sathyamangalam, Erode, Tamil Nadu, India. He has 7 years of experience in teaching field. He has published papers in various National Conferences and he has presented 3 papers in International Conferences held at various reputed engineering colleges. His interests include Data Mining and Grid Computing.