ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Cloud Information Accountability and Data Authenticity in Distributed Auditing Mechanism

B.Abhilash Reddy
B.Tech 4th Year,
Department of ECE
Sreenidhi Institute of Science & Technology,
Hyderabad.

P.Haripriya
B.Tech 4th Year,
Department of ECE
Sreenidhi Institute of Science & Technology,
Hyderabad.

Abstract:
Cloud computing enables highly scalable services to be simply frenzied over the Internet on an as-needed basis. A major aspect of the cloud services is that users' data are usually processed remotely in strange machines that users do not own or operate. While enjoying the expediency brought by this new budding technology, users' fears of losing control of their own data (mainly, financial and health data) can become a significant barrier to the wide adoption of cloud services. To concentrate on this problem, in this paper, we suggest a novel highly decentralized information accountability framework to keep track of the authentic usage of the users' data in the cloud. In particular, we suggest an object-centered approach that enables enclosing our logging mechanism together with users' data and policies. We force the JAR programmable abilities to both create a dynamic and itinerant object, and to ensure that any access to users' data will trigger authentication and programmed logging local to the JARs. To reinforce user's control, we also give distributed auditing mechanisms. We supply extensive experimental studies that demonstrate the efficiency and effectiveness of the proposed approaches.

Index Terms—Cloud computing, accountability, data sharing.

INTRODUCTION

Cloud computing presents a new way to supplement the current consumption and delivery model for IT services based on the Internet, by providing for vigorously scalable and often virtualized resources as a service over the Internet. To date, there are a number of prominent commercial and individual cloud computing services, counting Amazon, Google, Microsoft, Yahoo, and Sales force [19]. Details of the services provided are abstracted from the users who no longer need to be experts of technology infrastructure. Furthermore, users may not know the machines which actually process and host their data. While enjoying the expediency brought by this new technology, users also start perturbing about losing control of their own data. The data processed on clouds are frequently outsourced, leading to a number of issues connected to accountability, counting the handling of personally identifiable information. Such qualms are becoming a significant hurdle to the wide adoption of cloud services [30].To allay users' concerns, it is essential to provide an effective mechanism for users to monitor the usage of their data in the cloud. For instance, users need to be able to ensure that their data are handled according to the service level agreements made at the time they sign on for services in the cloud. Conservative access control approaches developed for closed domains such as databases and operating systems, or approaches by means of a centralized server in distributed environments, are not appropriate, due to the following features characterizing cloud environments.

First, data handling can be outsourced by the direct cloud service provider (CSP) to other entities in the cloud and theses entities can also delegate the errands to others, and so on. Second, entities are permitted to join and leave the cloud in a flexible manner. Accordingly, data handling in the cloud goes through a complex and dynamic hierarchical service chain which does not exist in conventional environments. To conquer the above problems, we propose a new approach, namely Cloud Information Accountability (CIA) framework, based on the concept of information accountability [44]. Unlike privacy protection technologies which are built on the hide-it-or-lose-it perspective, information accountability spotlights on keeping the data usage transparent and trackable. Our proposed CIA framework gives end-to-end accountability in a highly distributed manner. One of the main innovative features of the CIA framework lies in its ability of maintaining lightweight and powerful accountability that combines aspects of access control, usage control and substantiation. By means of the CIA, data holders can track not only whether or not the service-level agreements are being honored, but also implement access and usage control rules as needed. Connected with the accountability feature, we also develop two discrete modes for auditing: push mode and pull mode. The push mode refers to logs being sporadically sent to the data owner or stakeholder while the pull mode refers to an alternative approach whereby the user(or another authorized party) can retrieve the logs as needed.

The design of the CIA framework presents considerable challenges, including uniquely identifying CSPs, guaranteeing the reliability of the log, acclimatizing to a highly decentralized infrastructure, etc. Our fundamental approach toward addressing these issues is to leverage and extend the programmable capability of JAR (Java ARchives) files to automatically log the usage of the users' data by any entity in the cloud. Users will send their data alongside with any policies such as access control policies and logging policies that they want to impose, enclosed in JAR files, to cloud service contributors. Any access to the data will trigger an automated and authenticated logging mechanism local to the JARs. We pass on to this type of enforcement as "strong binding" since the policies and the logging mechanism travel with the data. This strong binding subsists even when copies of the JARs are produced; thus, the user will have control over his data at any location. Such decentralized logging mechanism convenes the dynamic nature of the cloud but also imposes challenges on ensuring the integrity of the logging. To manage with this issue, we provide the JARs with a central point of contact which forms a link between them and the user. It reports the error correction information sent by the JARs, which allows it to observe the loss of any logs from any of the JARs. Furthermore, if a JAR is not able to contact its central point, any access to its enclosed data will be denied.

RELATED WORK

In this segment, we first review related works addressing the privacy and security concerns in the cloud. Then, we briefly confer works which adopt analogous techniques as our approach but serve for different reasons.

Cloud Privacy and Security

Cloud computing has elevated a range of significant privacy and security issues [19], [25], [30]. Such concerns are due to the fact that, in the cloud, users' data and requests reside—at least for a definite amount of time—on the cloud cluster which is owned and preserved by a third party. Apprehensions arise since in the cloud it is not always obvious to individuals why their personal information is demanded or how it will be used or accepted on to other parties. To date, small work has been done in this space, in particular with reverence to accountability. Pearson et al. have projected accountability mechanisms to address privacy apprehensions of end users [30] and then enlarge a privacy manager [31]. Their basic thought is that the user's classified data are sent to the cloud in an encrypted form, and the processing is done on the encrypted data. The output of the processing is deobfuscated by the isolation manager to divulge the correct result. Though, the privacy manager gives only limited features in that it does not guarantee protection once the data are being revealed. In [7], the authors present a layered architecture for addressing the end-to-end trust supervision and accountability problem in amalgamated systems. The authors' focus is very dissimilar from ours, in that they mostly leverage trust associations for accountability, along with authentication and inconsistency detection. Additional, their solution requires third-party services to complete the monitoring and focuses on inferior level observing of system resources.

Researchers have examined accountability mostly as a demonstrable property through cryptographic mechanisms, mainly in the perspective of electronic commerce [10], [21]. A delegate work in this area is specified by [9]. The authors propose the usage of strategies attached to the information and present a logic for accountability data in distributed settings. Likewise, Jagadeesan et al. recently proposed a logic for conniving accountability-based distributed systems [20]. In [10], Crispo and Ruffo projected an interesting approach associated to accountability in case of delegation. Delegation is harmonizing to our work, in that we do not aspire at calculating the information workflow in the clouds. In a synopsis, all these works stay at a theoretical level and do not comprise any algorithm for tasks like obligatory logging.

To the finest of our knowledge, the only work advising a distributed approach to accountability is from Lee and colleagues [22]. The instigators have projected an agent-based system explicit to grid computing. Distributed jobs, along with the resource utilization at local machines are trailed by static software representatives. The idea of accountability policies in [22] is connected to ours, but it is mostly focused on resource utilization and on tracking of sub jobs practiced at multiple calculating nodes, rather than access control.

Other Related Techniques

With reverence to Java-based techniques for security, our methods are associated to self-defending objects (SDO) [17]. Self-defending objects are an expansion of the object-oriented programming model, where software objects that offer receptive functions or hold sensitive data are accountable for shielding those functions/data. Likewise, we also extend the perceptions of object-oriented programming. The key dissimilarities in our implementations is that the creators still rely on a centralized database to sustain the access records, while the substances being protected are detained as separate files. In preceding work, we provided a Java-based approach to stop privacy seepage from indexing [39], which could be incorporated with the CIA framework projected in this work since they build on connected architectures.

In provisions of authentication techniques, Appel and Felten [13] projected the Proof-Carrying authentication (PCA) framework. The PCA comprises a high order logic language that permits quantification over predicates, and hubs on access control for web services. While associated to ours to the scope that it helps upholding safe, high-performance, mobile code, the PCA's goal is highly dissimilar from our research, as it focuses on authenticating code, rather than scrutinizing content. Another work is by Mont et al. who projected an approach for strongly pairing content with access control, by Identity-Based Encryption (IBE) [26]. We also force IBE methods but in a very different way. We do not rely on IBE to bind the content with the rules. Instead, we use it to give strong assurances for the encrypted content and the log files, such as protection against preferred plaintext and cipher text attacks.

CLOUD INFORMATION ACCOUNTABILITY

In this part, we present an impression of the Cloud Information Accountability framework and discuss how the CIA framework assembles the design requirements conversed in the earlier section.

The Cloud Information Accountability framework projected in this work demeanors automated logging and disseminated auditing of pertinent access executed by any entity, carried out at any point of time at any cloud service contributor. It has two major gears: logger and log harmonizer.

Major Components

There are two main components of the CIA, the first being the logger, and the second being the log harmonizer. The logger is the part which is strongly joined with the user's data, so that it is downloaded when the data are accessed, and is copied when the information are copied. It grips a particular instance or copy of the user's data and is accountable for logging access to that example or copy. The log harmonizer forms the central component which permits the user access to the log records.

The logger is strongly joined with user's data (either single or multiple data objects). Its main tasks include mechanically logging access to data items that it holds, encrypting the log record by the public key of the content owner, and sporadically sending them to the log harmonizer. It may also be configured to guarantee that access and usage control policies connected with the data are privileged. For example, a data owner can identify that user X is only permitted to view but not to change the data. The logger will manage the data access even after it is downloaded by user X.

The logger necessitates only minimal support from the server (e.g., a legitimate Java virtual machine installed) in order to be installed. The tight pairing between data and logger, results in a highly disseminated logging system, therefore meeting our first design requirement. Moreover, since the logger does not require to be installed on any system or entail any special support from the server, it is not very invasive in its actions, thus fulfilling our fifth requirement. Lastly, the logger is also accountable for generating the error alteration information for each log record and sends the same to the log harmonizer. The error alteration information shared with the encryption and verification mechanism gives a robust and consistent recovery mechanism, therefore meeting the third obligation.

The log harmonizer is liable for inspecting.

Being the trusted component, the log harmonizer produces the master key. It clutches on to the decryption key for the IBE key pair, as it is accountable for decrypting the logs. On the other hand, the decryption can be accepted out on the client end if the path between the log harmonizer and the client is not hoped. In this case, the harmonizer sends the key to the client in a secure key swap.

It ropes two auditing strategies: push and pull. Under the push approach, the log file is pushed back to the data owner sometimes in an automated manner. The pull mode is an on-demand approach, whereby the log file is attained by the data owner as frequently as request. These two modes allow us to satisfy the aforesaid fourth design obligation. In case there survive multiple loggers for the same set of data objects, the log harmonizer will merge log proceedings from them prior to sending back to the data owner. The log harmonizer is also accountable for handling log file corruption. As well, the log harmonizer can itself complete logging in addition to auditing. Sorting out the logging and auditing functions recovers the performance. The logger and the log harmonizer are both executed as lightweight and transferable JAR files. The JAR file implementation gives automatic logging functions, which convenes the second design obligation.

Data Flow

The overall CIA framework, merging data, users, logger and harmonizer is drafted in Fig. 1. At the starting, each user generates a pair of public and private keys based on Identity-Based Encryption [4] (step 1 in Fig. 1). This IBE method is a Weil-pairing-based IBE method, which defends us against one of the most common attacks to our architecture. Using the produced key, the user will generate a logger component which is a JAR file, to store its data objects.

The JAR file incorporate a set of simple access control rules identify whether and how the cloud servers, and probably other data stakeholders (users, companies) are endorsed to access the content itself. Then, he sends the JAR file to the cloud service contributor that he subscribes to. To validate the CSP to the JAR (steps 3-5 in Fig. 1), we use Open SSL supported certificates, wherein a trusted certificate influence certifies the CSP. In the incident that the access is requested by a user, we use SAML-based authentication [8], where in a trusted identity contributor issues certificates confirming the user's uniqueness based on his username.

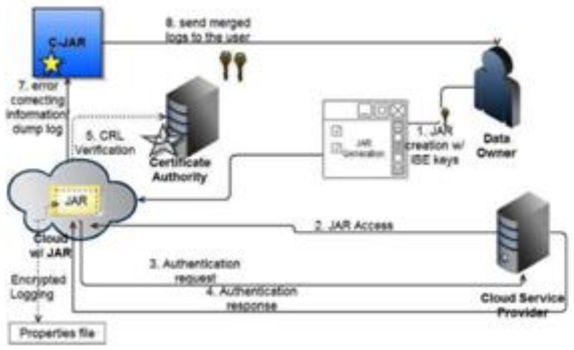

Fig. 1. Overview of the cloud information accountability framework.

Once the verification succeeds, the service contributor (or the user) will be allowed to admission the data enclosed in the JAR. Depending on the pattern settings distinct at the time of formation, the JAR will provide custom control associated with logging, or will give only logging functionality. As for the logging, every time there is an access to the data, the JAR will automatically generate a log record, encrypt it using the public key distributed by the data owner, and hoard it along with the data (step 6 in Fig. 1). The encryption of the log file stops unconstitutional changes to the file by assailants. The data owner could opt to salvage the same key pair for all JARs or generate different key pairs for separate JARs. Using separate keys can augment the security without introducing any overhead excluding in the initialization phase. Besides, some error alteration information will be sent to the log harmonizer to handle probable log file sleaze (step 7 in Fig. 1). To ensure dependability of the logs, each confirmation is signed by the body accessing the content. Further, individual records are messed together to create a chain structure, able to rapidly detect probable errors or misplaced records. The encrypted log files can later be decrypted and their integrity confirmed. They can be contacted by the data owner or other endorsed stakeholders at any time for auditing ideology with the aid of the log harmonizer (step 8 in Fig. 1).

END-TO-END AUDITING MECHANISM

In this section, we illustrate our distributed auditing mechanism counting the algorithms for data owners to query the logs regarding their data.

Push and Pull Mode

To allow customers to be timely and accurately educated about their data convention, our dispersed logging mechanism is harmonized by an innovative auditing method. We support two harmonizing auditing modes: 1) push mode; 2) pull mode.

Push mode. In this mode, the logs are sporadically pushed to the data owner (or auditor) by the harmonizer. The push action will be generated by either type of the subsequent two events: one is that the time elapses for a definite period according to the chronological timer inserted as part of the JAR file; the other is that the JAR file surpassed the size predetermined by the content owner at the time of formation. Subsequent to the logs are sent to the data owner, the log files will be discarded, so as to free the space for potential access logs. Together with the log files, the error correcting information for those logs is also discarded.

This push mode is the fundamental mode which can be accepted by both the Pure Log and the Access Log, despite of whether there is a demand from the data owner for the log files. This mode serves two necessary functions in the logging architecture: 1) it guarantees that the size of the log files does not detonate and 2) it enables timely detection and alteration of any loss or damage to the log files.

In relation to the latter function, we observe that the auditor, upon getting the log file, will confirm its cryptographic pledges, by checking the records' reliability and authenticity. By construction of the records, the auditor will be capable to quickly notice forgery of entries, using the checksum supplementary to each and every record.

Pull mode.

This mode permits auditors to recover the logs anytime when they want to check the recent access to their own information. The pull communication consists simply of an FTP pull command, which can be concerns from the command line. For naive users, a wizard contains a batch file can be simply built. The appeal will be sent to the harmonizer, and the customer will be informed of the data's positions and obtain an incorporated copy of the authentic and conserved log file.

The communication with the harmonizer commences with a simple handshake. If no reply is established, the log file proceedings an error. The data owner is then prepared through e-mails, if the JAR is configured to send error announcements. Once the handclasp is completed, the message with the harmonizer proceeds, using a TCP/IP protocol. If any of the aforesaid events (i.e., there is request of the log file, or the size or time surpasses the threshold) has happened, the JAR simply abandons the log files and resets all the variables, to create space for new proceedings.

PERFORMANCE STUDY

In this section, we first commence the settings of the test situation and then present the performance revise of our system.

Experimental Settings

We tested our CIA skeleton by setting up a little cloud, using the Emulab testbed [42]. In exacting, the test environment consists of numerous Open SSL-enabled servers:

One head node which is the certificate authority, and several calculating nodes. All of the servers is mounted with Eucalyptus [41]. Eucalyptus is an open source cloud execution for Linux-based schemes. It is slackly based on Amazon EC2, so bringing the dominant functionalities of Amazon EC2 into the release source province. We used Linux-based servers running Fedora 10 OS. Every server has a 64-bit Intel Quad Core Xeon E5530 processor,4 GB RAM, and a 500 GB Hard Drive. All of the servers is prepared to run the Open JDK runtime situation with IcedTea6 1.8.2.

Experimental Results

In the trials we first observe the time taken to create a log file and then calculate the overhead in the classification. With admiration to time, the overhead can occur at three points: during the verification, during encryption of a log record, and during the amalgamation of the logs. Moreover, with admiration to storage overhead, we observe that our construction is very lightweight, in that the only data to be amassed are given by the actual files and the related logs. Further, JAR act as a compressor of the records that it handles. In particular, several files can be handled by the equivalent logger constituent. To this extent, we examine whether a single logger constituent, used to grip more than one file, consequences in storage overhead.

Log Creation Time

In the first round of experiments, we are involved in finding out the time taken to generate a log file when there are entities endlessly contacting the data, causing incessant logging. Results are shown in Fig. 3. It is not surprising to see that the time to generate a log file increases linearly with the size of the log file. Particularly, the time to create a 100 Kb file is about 114.5 ms as the time to create a 1 MB file averages at 731 ms. With this trial as the baseline, one can choose the amount of time to be precise between dumps, keeping other variables similar to space restraints or network traffic in mind.

Authentication Time

The next point that the overhead can happen is during the verification of a CSP. If the time taken for this verification is too elongated, it may become a bottleneck for accessing the enclosed data. To estimate this, the head node subjected Open SSL certificates for the calculating nodes and we calculated the total time for the Open SSL authentication to be finished and the credential revocation to be checked. Considering one access at the time, we locate that the verification time averages around 920 ms which shows that not too much overhead is added during this phase. As of present, the verification takes place every time the CSP needs to access the information. The performance can be further enhanced by caching the certificates.

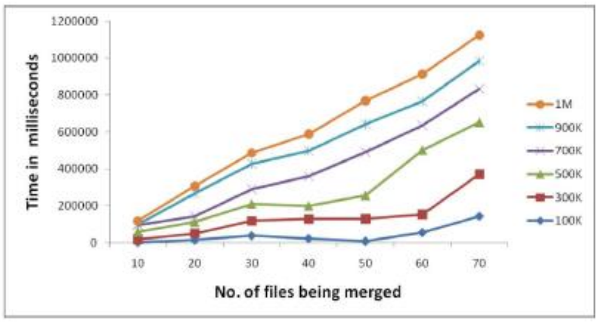

Fig 3. Time to merge log files.

The time for validating an end user is about the same when we deem only the actions requisite by the JAR, viz. obtaining a SAML certificate and then assessing it. This is because both the Open SSL and the SAML certificates are griped in a similar manner by the JAR. When we deem the user measures (i.e., submitting his username to the JAR), it averages at 1.2 minutes.

Time Taken to Perform Logging

This set of experiments revises the consequence of log file size on the logging performance. We calculate the average time taken to grant an access plus the time to write the consequent log record. The time for yielding any access to the data items in a JAR file comprises the time to appraise and enforce the appropriate policies and to locate the requested data items. In the experiment, we let multiple servers endlessly access the equal data JAR file for a minute and recorded the number of log records produced. Every access is just a view request and therefore the time for executing the action is negligible. Accordingly, the average time to log an exploit is about 10 seconds, which comprises the time taken by a user to double click the JAR or by a server to run the script to unlock the JAR. We also calculated the log encryption time which is about 300 ms (per record) and is seemingly unconnected from the log size.

Log Merging Time

To ensure if the log harmonizer can be a bottleneck, we calculate the amount of time necessary to merge log files. In this experiment, we guaranteed that each of the log files had10 to 25 percent of the records in frequent with one other. The exact number of records in general was random for every repetition of the research. The time was averaged over 10 repetitions. We tested the time to combine up to 70 log files of 100 KB, 300 KB, 500 KB, 700 KB, 900 KB, and 1 MB each. The results are shown in Fig. 4. We can scrutinize that the time augments almost linearly to the number of files and size of files, with the slightest time being taken for amalgamation two100 KB log files at 59 ms, as the time to combine 70 1 MB files was 2.35 minutes.

Size of the Data JAR Files

Finally, we examine whether a single logger, used to handle more than one file, consequences in storage overhead. We calculate the size of the loggers (JARs) by varying the number as well as size of data items detained by them. We tested the increase in size of the logger holding 10 content files (i.e., images) of the equivalent size as the file size increases. Instinctively, in case of larger size of data items held by a logger, the overall logger also augments in size. The size of logger produces from 3,500 to 4,035 KB when the size of content items varies from 200 KB to 1 MB. Generally, due to the density provided by JAR files, the size of the logger is ordered by the size of the largest files it contains. Notice that we deliberately did not comprise large log files (less than 5 KB), so as to focus on the overhead supplementary by having numerous content files in a single JAR.

Overhead Added by JVM Integrity Checking

We examine the overhead added by together the JRE installation/repair process, and by the time taken for calculation of hash codes.The time taken for JRE installation/repair averages approximately 6,500 ms. This time was measured by captivating the system time stamp at the start and end of the installation/repair.To compute the time slide added by the hash codes, we simply calculate the time taken for each hash computation. This time is originate to average around 9 ms. The number of hash commands diverges based on the size of the code in the code does not alter with the content, the number of hash commands remain stable.

CONCLUSION

We proposed pioneering approaches for mechanically logging any access to the data in the cloud jointly with an auditing mechanism. Our approach permits the data owner to not only audit his content but also implement strong back-end protection if needed. In addition, one of the main features of our work is that it allows the data proprietor to audit even those copies of its data that were made without his knowledge. In the future, we plan to refine our approach to confirm the honesty of the JRE and the confirmation of JARs [23]. For instance, we will investigate whether it is probable to leverage the notion of a secure JVM [18] being developed by IBM. This study is intended at providing software tamper confrontation to Java applications. In the long term, we plan to intend a widespread and more generic object-oriented approach to support autonomous protection of traveling content. We would like to hold up a variety of security policies, like indexing policies for text files, custom control for executables, and general accountability and provenance controls.

REFERENCES

[1] P. Ammann and S. Jajodia, “Distributed Timestamp Generation in Planar Lattice Networks,” ACM Trans. Computer Systems, vol. 11, pp. 205-225, Aug. 1993.

[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] E. Barka and A. Lakas, “Integrating Usage Control with SIP-Based Communications,” J. Computer Systems, Networks, and Comm., vol. 2008, pp. 1-8, 2008.

[4] D. Boneh and M.K. Franklin, “Identity-Based Encryption from the Weil Pairing,” Proc. Int’l Cryptology Conf. Advances in Cryptology, pp. 213-229, 2001.

[5] R. Bose and J. Frew, “Lineage Retrieval for Scientific Data Processing: A Survey,” ACM Computing Surveys, vol. 37, pp. 1-28, Mar. 2005.

[6] P. Buneman, A. Chapman, and J. Cheney, “Provenance Management in Curated Databases,” Proc. ACM SIGMOD Int’l Conf. Management of Data (SIGMOD ’06), pp. 539-550, 2006.

[7] B. Chun and A.C. Bavier, “Decentralized Trust Management and Accountability in Federated Systems,” Proc. Ann. Hawaii Int’l Conf. System Sciences (HICSS), 2004.

[8] OASIS Security Services Technical Committee, “Security Assertion Markup Language (saml) 2.0,” http://www.oasis-open.org/committees/tc home.php?wg abbrev=security, 2012.

[9] R. Corin, S. Etalle, J.I. den Hartog, G. Lenzini, and I. Staicu, “ALogic for Auditing Accountability in Decentralized Systems,” Proc. IFIP TC1 WG1.7

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Workshop Formal Aspects in Security and Trust, pp. 187-201, 2005.

[10] B. Crispo and G. Ruffo, "Reasoning about Accountability within Delegation," Proc. Third Int'l Conf. Information and Comm. Security (ICICS), pp. 251-260, 2001.

[11] Y. Chen et al., "Oblivious Hashing: A Stealthy Software Integrity Verification Primitive," Proc. Int'l Workshop Information Hiding, F. Petitcolas, ed., pp. 400-414, 2003.

[12] S. Etalle and W.H. Winsborough, "A Posteriori Compliance Control," SACMAT '07: Proc. 12th ACM Symp. Access Control Models and Technologies, pp. 11-20, 2007.

[13] X. Feng, Z. Ni, Z. Shao, and Y. Guo, "An Open Framework for Foundational Proof-Carrying Code," Proc. ACM SIGPLAN Int'l Workshop Types in Languages Design and Implementation, pp. 67-78,2007.

[14] Flickr, http://www.flickr.com/, 2012.

[15] 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.

[16] J. Hightower and G. Borriello, "Location Systems for Ubiquitous Computing," Computer, vol. 34, no. 8, pp. 57-66, Aug. 2001.

[17] J.W. Holford, W.J. Caelli, and A.W. Rhodes, "Using Self-Defending Objects to Develop Security Aware Applications in Java," Proc. 27th Australasian Conf. Computer Science, vol. 26,pp. 341-349, 2004.

[18] Trusted Java Virtual Machine IBM, http://www.almaden.ibm.com/cs/projects/jvm/, 2012.

[19] 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.

[20] R. Jagadeesan, A. Jeffrey, C. Pitcher, and J. Riely, "Towards a Theory of Accountability and Audit," Proc. 14th European Conf. Research in Computer Security (ESORICS), pp. 152-167, 2009.

[21] R. Kailar, "Accountability in Electronic Commerce Protocols," IEEE Trans. Software Eng., vol. 22, no. 5, pp. 313-328, May 1996.

[22] W. Lee, A. Cinzia Squicciarini, and E. Bertino, "The Design and Evaluation of Accountable Grid Computing System," Proc. 29thIEEE Int'l Conf. Distributed Computing Systems (ICDCS '09),pp. 145-154, 2009.