International Journal of Application or Innovation in Engineering & Management (IJAIEM)
Web Site: www.ijaiem.org Email: editor@ijaiem.org
Volume 3, Issue 7, July 2014 **ISSN 2319 - 4847**

BILINEAR PAIRING BASED PUBLIC AUDITING FOR SECURE CLOUD STORAGE USING TPA

Ms. Shweta khidrapure[1], Prof. Archana lomte[2]

[1]ME Computer Engg

[2]Computer Dept. BSIOTR, Wagholi

ABSTRACT

Cloud data security is concern for the client while using the cloud services provided by the service provider. In this paper we are analyzed various mechanisms to ensure reliable data storage using cloud services. It mainly focuses on the way of providing computing resources in form of service rather than a product and utilities are provided to users over internet. In the cloud, application and services move to centralized huge data center and services and management of this data may not be trustworthy into cloud environment the computing resources are under control of service provider and the third-party-auditor ensures the data integrity over out sourced data. Third-party-auditor not only read but also may be change the data. Therefore a mechanism should be provided to solve the problem. In this paper, we utilize the public key based homomorphism authenticator and uniquely integrate it with random mask technique to achieve a privacy-preserving public auditing system for cloud data storage security while keeping all above requirements in mind. To support efficient Handling of multiple auditing tasks, we further explore the technique of bilinear aggregate signature to extend our main result into a multi-user setting, where TPA can perform multiple auditing tasks simultaneously.

Keywords— CSP,IAAS,PAAS,SAAS,TPA etc

1. INTRODUCTION

In cloud computing, security is most important task.Cloud computing entrusts services with users data, software and computation on a published application programming interface over a network. Cloud provides a platform for many types of services. It has a considerable overlap with software as a service (SaaS), as in [5] End users access cloud based applications through a web browser or a light weight desktop or a mobile app while the business software and data are stored on servers at a remote location. Cloud application providers strive to give same or better service and performance than if the software programs were installed. When we talk about cloud Security, maintaining data integrity is one of the most important and difficult task. When we talk about cloud users, they are using cloud services provided by the cloud provider, as in [5] and again, in the case of maintaining integrity of the data, so we cannot trust the service provider to handle the data, as he himself can modify the original data and the integrity may be lost. If a smart hacker hacks the cloud server and steals the data and modifies it then in some cases this modification is not even identified by the cloud provider. So, in this case, we take the help of a trusted third party auditor to check for the integrity of our data. This third party auditor takes care of our data and makes sure that data integrity is maintained. We view the procedure of integrity checking as a key's proficiency within software, platform, and infrastructure security focus area of our cloud architecture. Our vision for helping assure ongoing system integrity in a virtualized environment includes an evolution of integrity checking competences, as in [5] Each phase, in this evolution relies on secure start up enabled and provides an increasing level of assurance and. This evolution begins with one-time integrity checks at system

2. LITERATURE SURVE

Ateniese et al. [1] are the first to consider public auditability in their "provable data possession" PDP) model for ensuring possession of data files on untrusted storages. They utilize the RSA-based homomorphic linear authenticators for auditing outsourced data and suggest randomly sampling a few blocks of the file. However, among their two proposed schemes, the one with public auditability exposes the linear combination of sampled blocks to external auditor. When used directly, their protocol is not provably privacy preserving, and thus may leak user data information to the external auditor. Juels et al. [2] describe a "proof of retrievability" (PoR)model, where spot-checking and error-correcting codes are used to ensure both "possession" and "retrievability" of data files on remote archive service systems. However, the number of audit challenges a user can perform is fixed a priori, and public auditability is not supported in their main scheme. Although they describe a straightforward Merkle-tree construction for public PoRs, this approach only works with encrypted data. Later, Bowers et al. [3] propose an improved framework for POR protocols that generalizes Juels' work. [2], Bloom filters have seen various uses:

Web cache sharing ([3])

Collaborating Web caches use Bloom filters (dubbed "cache summaries") as compact representations for the local set of cached files. Each cache periodically broadcasts its summary to all other members of the distributed cache. Using all summaries received, a cache node has a (partially outdated, partially wrong) global image about the set of files stored in the aggregated cache. The Squid Web Proxy Cache [1] uses "Cache Digests" based on a similar idea.

Query filtering and routing ([4, 6, 7])

The Secure wide-area Discovery Service [6], subsystem of Ninja project [5], organizes service providers in a hierarchy. Bloom filters are used as summaries for the set of services offered by a node. Summaries are sent upwards in the hierarchy and aggregated. A query is a description for a specific service, also represented as a Bloom filter. Thus, when a member node of the hierarchy generates/receives a query, it has enough information at hand to decide where to forward the query: downward, to one of its descendants (if a solution to the query is present in the filter for the corresponding node), or upward, toward its parent (otherwise).

The OceanStore [7]

replica location service uses a two-tiered approach: first it initiates an inexpensive, probabilistic search (based on Bloom filters, similar to Ninja) to try and find a replica. If this fails, the search falls-back on (expensive) deterministic algorithm (based on Plaxton replica location algorithm). Alas, their description of the probabilistic search algorithm is laconic. (An unpublished text [11] from members of the same group gives some more details. But this does not seem to work well when resources are dynamic.) A differential file contains a batch of database records to be updated. For performance reasons the database is updated only periodically (i.e., midnight) or when the differential file grows above a certain threshold. However, in order to preserve integrity, each reference/query to the database has to access the differential file to see if a particular record is scheduled to be updated. To speed-up this process, with little memory and computational overhead, the differential file is represented as a Bloom filter.

Free text searching ([10]).

Basically, the set of words that appear in a text is succinctly represented using a Bloom filter.

RELATED WORK

EXISTING SYSTEM

To securely introduce an effective third party auditor (TPA), the following two fundamental requirements have to be met: 1) TPA should be able to efficiently audit the cloud data storage without demanding the local copy of data, and introduce no additional on-line burden to the cloud user; 2) The third party auditing process should bring in no new vulnerabilities towards user data privacy.

PROBLEM STATEMENT

Utilization of the public key based Bilinear Pairing authenticator to achieve a privacy-preserving public auditing system for cloud data storage security while keeping all above requirements in mind. To support efficient handling of multiple auditing tasks, we further explore the technique of bilinear aggregate signature to extend our main result into a multi-user setting, where TPA can perform multiple auditing tasks simultaneously. Extensive security and performance analysis shows the proposed schemes are provably secure and highly efficient. We also show how to extent our main scheme to support batch auditing for TPA upon delegations from multi-users. And we have increased the efficiency of TPA.

OBJECTIVES

- Public auditability To allow TPA to verify the correctness of the cloud data on demand without retrieving a copy of the whole data or introducing additional online burden to the cloud users.
- Storage correctness To ensure that there exists no cheating cloud server that can pass the TPA's audit without indeed storing users' data intact.
- Privacy preserving To ensure that the TPA cannot derive users' data content from the information collected during the auditing process.
- Batch auditing To enable TPA with secure and efficient auditing capability to cope with multiple auditing delegations from possibly large number of different users simultaneously.
- Lightweight To allow TPA to perform auditing with minimum communication and computation overhead.
- Efficiency:To allow the TPA to perform verification in a short period of time using Bloom Filters.

MOTIVATION

In particular, simply downloading all the data for its integrity verification is not a practical solution due to the expensiveness in I/O and transmission cost across the network. Besides, it is often insufficient to detect the data corruption only when accessing the data, as it does not give users correctness assurance for those unaccessed data and might be too late to recover the data loss or damage. Encryption does not completely solve the problem of protecting data privacy against third-party auditing but just reduces it to the complex key management domain. Unauthorized data leakage still remains possible due to the potential exposure of decryption keys.

SCOPE

In this paper the Bilinear Pairing based algorithm is presented, which is based on the idea of generating strong public key, to some extent, and providing the stong data encryption.For this the ideas from some recent variants of the pairing methods are used, which are quite good for generating the strong key. Bloom filters are used in the project for storing the public key which is in encrypted format.When the client sends a request to the TPA for data verification, then the TPA sends a request to the cloud storage for asking public key kept on cloud storage. As here the public key verification is done by TPA and as it is in encrypted format TPA will have zero knowledge of the data. Bloom filter data verification is faster than any other verification methods so here the verification time is minimized to improve the efficiency of TPA.

PROPOSED SYSTEM

In this paper, we utilize the public key based homomorphism authenticator and uniquely integrate it with random mask technique to achieve a privacy-preserving public auditing system for cloud data storage security while keeping all above requirements in mind. To support efficient Handling of multiple auditing tasks, we further explore the technique of bilinear aggregate signature to extend our main result into a multi-user setting, where TPA can perform multiple auditing tasks simultaneously. Extensive security and performance analysis shows the proposed schemes are provably secure and highly efficient. We also show how to extent our main scheme to support batch auditing for TPA upon delegations from multi-users.

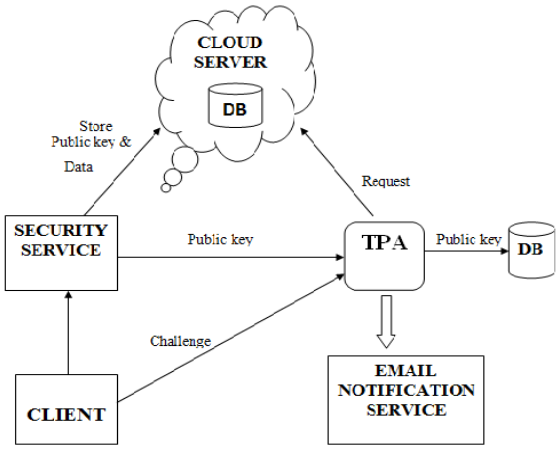


Fig 1: A Proposed system Architecture

Modules

The system is proposed to have the following modules along with functional requirements.

1) Module 1 will be of base paper (AES algorithm is used for cryptography)
2) Cloud base architecture
3) Cloud base document management using “Alfresco” open source software.
5) Use of bloom filters for generating hash file.

ADVANTAGES OF PROPOSED SYSTEM

1) We motivate the public auditing system of data storage security in Cloud Computing and provide a privacy-preserving auditing protocol. Our scheme enables an external auditor to audit user’s cloud data without learning the data content.
2) To the best of our knowledge, our scheme is the first to support scalable and efficient privacy preserving public storage auditing in Cloud. Specifically, our scheme achieves batch auditing where multiple delegated auditing tasks from different users can be performed simultaneously by the TPA in a privacy-preserving manner.
3) We prove the security and justify the performance of our proposed schemes through concrete experiments and comparisons with the state-of-the-art. Abbreviations

SYSTEM IMPLEMENTATION

It is the long dreamed vision of computing as a utility where users can remotely store data into the cloud so as to enjoy the on-demand high quality applications and services from a shared pool of configurable computing resources. By data outsourcing, users can be relieved from the burden of local data storage and maintenance. Thus, enabling public auditability for cloud data storage security is of critical importance so that users can resort to an external audit party to check the integrity of outsourced data when needed. To securely introduce an effective third party auditor (TPA), the following two fundamental requirements have to be met: 1) TPA should be able to efficiently audit the cloud data storage without demanding the local copy of data, and introduce no additional on-line burden to the cloud user. Specifically, our

contribution in this work can be summarized as the following three aspects:

1) We motivate the public auditing system of data storage security in Cloud Computing and provide a privacy-preserving auditing protocol, i.e., our scheme supports an external auditor to audit user's outsourced data in the cloud without learning knowledge on the data content.
2) To the best of our knowledge, our scheme is the first to support scalable and efficient public auditing in the Cloud Computing. In particular, our scheme achieves batch auditing where multiple delegated auditing tasks from different users can be performed simultaneously by the TPA.
3) We prove the security and justify the performance of our proposed schemes through concrete experiments and comparisons with the state-of-the-art Fig 2 shows the architecture of cloud data storage services..

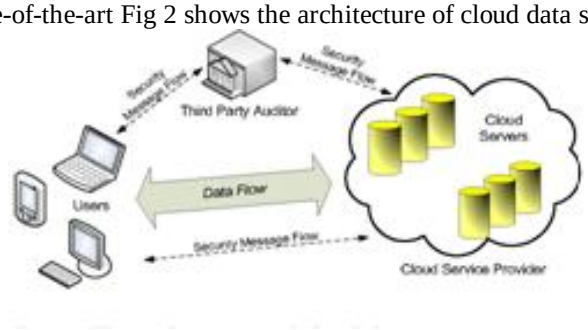


Fig 2: Architecture of cloud data Service

To enable privacy-preserving public auditing for cloud data storage under the aforementioned model, our protocol design should achieve the following security and performance guarantee:

1) Public auditability: to allow TPA to verify the correctness of the cloud data on demand without retrieving a copy of the whole data or introducing additional on-line burden to the cloud users.
2) Storage correctness: to ensure that there exists no cheating cloud server that can pass the audit from TPA without indeed storing users' data intact.
3) Privacy-preserving: to ensure that there exists no way for TPA to derive users' data content from the information collected during the auditing process.
4) Batch auditing: to enable TPA with secure and efficient auditing capability to cope with multiple auditing delegations from possibly large number of different users simultaneously.
5) Lightweight: to allow TPA to perform auditing with minimum communication and computation overhead.

Bloom Filter

Bloom filters [2] are compact data structures for probabilistic representation of a set in order to support membership queries (i.e. queries that ask: "Is element *X* in set *Y*?"). This compact representation is the payoff for allowing a small rate of *false positives* in membership queries; that is, queries might incorrectly recognize an element as member of the set.

Constructing Bloom Filters

Consider a set $A = \{a_1, a_2, ..., a_n\}$ of *n* elements. Bloom filters describe membership information of *A* using a bit vector *V* of length *m*. For this, *k* hash functions, $h_1, h_2, ..., h_k$ with $h_i : X \rightarrow \{1..m\}$, are used as described below. The following procedure builds an *m* bits Bloom filter, corresponding to a set A and using $h_1, h_2, ..., h_k$ hash functions. Therefore, if a_i is member of a set *A*, in the resulting Bloom filter *V* all bits obtained corresponding to the hashed values of a_i are set to 1. Testing for membership of an element *elm* is equivalent to testing that all corresponding bits of *V* are set:

Bloom Filters

the Math (this follows the description in [1]**)**

One prominent feature of Bloom filters is that there is a clear tradeoff between the size of the filter and the rate of false positives. Observe that after inserting *n* keys into a filter of size *m* using *k* hash functions, the probability that a particular bit is still 0 is:

$$p_0 = \left(1 - \frac{1}{m}\right)^{kn} \approx 1 - e^{-\frac{kn}{m}}. \quad (1)$$

(Note that we assume perfect hash functions that spread the elements of *A* evenly throughout the space {1..*m*}. In practice, good results have been achieved using MD5 and other hash functions [10].)

Hence, the probability of a false positive (the probability that all *k* bits have been previously set) is:

$$p_{err} = (1 - p_0)^k = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-\frac{kn}{m}}\right)^k \quad (2)$$

In (2) p_{err} is minimized for $k = \frac{m}{n} \ln 2$ hash functions. In practice however, only a small number of hash functions are used. The reason is that the computational overhead of each hash additional function is constant while the incremental benefit of adding a new hash function decreases after a certain threshold (see Figure 2).

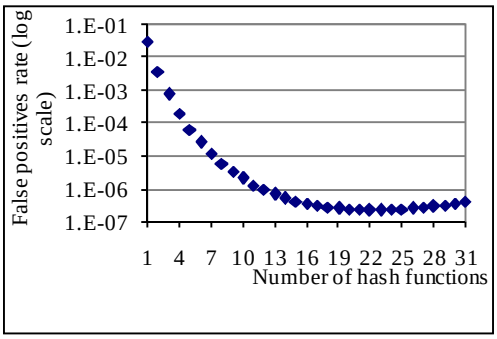


Figure 3: False positive rate

False positive rate as a function of the number of hash functions used. The size of the Bloom filter is 32 bits per entry (m/n=32). In this case using 22 hash functions minimizes the false positive rate. Note however that adding a hash function does not significantly decrease the error rate when more than 10 hashes are already used.

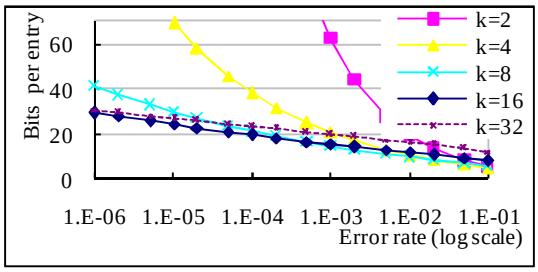


Figure 4: Size of Bloom filter

Size of Bloom filter (bits/entry) as a function of the error rate desired. Different lines represent different numbers of hash keys used. Note that, for the error rates considered, using 32 keys does not bring significant benefits over using only 8 keys. (2) is the base formula for engineering Bloom filters. It allows, for example, computing minimal memory requirements (filter size) and number of hash functions given the maximum acceptable false positives rate and number of elements in the set (as we detail in **Error! Reference source not found.**3).

$$\frac{m}{n} = \frac{-k}{\ln\left(1 - e^{\frac{\ln p_{err}}{k}}\right)} \text{ (bits per entry)} \qquad (3)$$

Bloom filters are compact data structures for probabilistic representation of a set in order to support membership queries. The main design tradeoffs are the number of hash functions used (driving the computational overhead), the size of the filter and the error (collision) rate. Formula (2) is the main formula to tune parameters according to application requirements.

Mathematical Model

Let G be a cyclic additive group generated by P whose order is a prime q and G be a cyclic multiplicative group with the same order q. Let e: G×G->G be the map with following properties.

1) Bilinearity: e(aP,bQ)=e(P,Q)^{ab} for all P,Q ∈ G and a,b ∈ Z_q
2) Non-degeneracy: There exist P,Q ∈ G_1 such that e(P,Q)≠1
3) Computability: There is an efficient algorithm to compute e(P,Q) for all P,Q ∈ G_1

Algorithms

AES is a block cipher with a block length of 128 bits. AES allows for three different key lengths: 128, 192, or 256 bits. Encryption consists of 10 rounds of processing for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. Except for the last round in each case, all other rounds are identical. Each round of processing includes one single-

byte based substitution step, a row-wise permutation step, a column-wise mixing step, and the addition of the round key. The order in which these four steps are executed is different for encryption and decryption. To appreciate the processing steps used in a single round, it is best to think of a 128-bit block as consisting of a 4 × 4 matrix of bytes, arranged as follows: Therefore, the first four bytes of a 128-bit input block occupy the first column in the 4×4 matrix of bytes. The next four bytes occupy the second column, and so on.

$$\begin{bmatrix} byte_0 & byte_4 & byte_8 & byte_{12} \\ byte_1 & byte_5 & byte_9 & byte_{13} \\ byte_2 & byte_6 & byte_{10} & byte_{14} \\ byte_3 & byte_7 & byte_{11} & byte_{15} \end{bmatrix}$$

THE OVERALL STRUCTURE OF AES

The overall structure of AES encryption/decryption is show n in Figure 5. The number of rounds shown in Figure 5, is for the case when the encryption key is 128 bit long. (As mentioned earlier, the number of rounds is 12 when the key is 192 bits and 14 when the key is 256.) Before any round-based processing for encryption can begin, the input state array is XORed with the first four words of the key schedule. The same thing happens during decryption except that now we XOR the cipher text state array with the last four words of the key schedule.

For encryption, each round consists of the following four steps:

1) Substitute bytes
2) Shift rows
3) Mix columns, and
4) Add round key.

The last step consists of XORing the output of the previous three steps with four words from the key schedule.

For decryption, each round consists of the following four steps:

1) Inverse shift rows,
2) Inverse substitute bytes,
3) Add round key, and
4) Inverse mix columns.

The third step consists of XORing the output of the previous two steps with four words from the key schedule. The last round for encryption does not involve the “Mix columns” step. The last round for decryption does not involve the “Inverse mix columns” step.

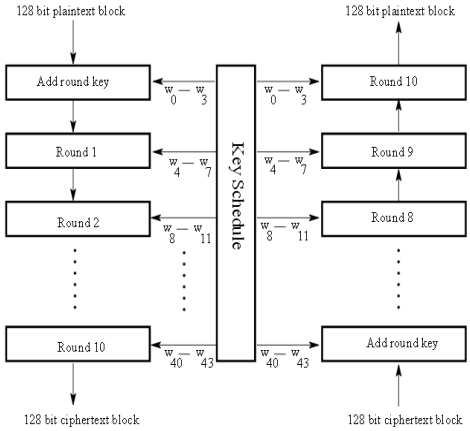


Figure5: The overall structure of AES for the case of 128-bit encryption key

AES Encryption AES Decryption

1. Substitute bytes

(called SubBytes for byte-by-byte substitution during the forward process) (The corresponding substitution step used during decryption is called InvSubBytes.) This step consists of using a 16 × 16 lookup table to find a replacement byte for a given byte in the input state array.

2. Shift rows

(called ShiftRows for shifting the rows of the state array during the forward process) (The corresponding transformation during decryption is denoted InvShiftRows for Inverse Shift Row Transformation.)The goal of this transformation is to scramble the byte order inside each 128-bit block.

3. Mix columns

(called MixColumns for mixing up of the bytes in each column separately during the forward process) (The corresponding transformation during decryption is denoted InvMixColumns and stands for inverse mix column transformation.) The goal is here is to further scramble up the 128-bit input block.The shift-rows step along with the mix-column step causes each bit of the ciphertext to depend on every bit of the plaintext after 10 rounds of processing.

4. Add round key

(called AddRoundKey for adding the round key to the output of the previous step during the forward process) (The corresponding step during decryption is denoted InvAddRound Key for inverse add round key transformation.)

Bilinear pairing based public auditing Schemes

The Bilinear pairing based public auditing for secure cloud has two schemes

Section 1: Signature generation

Section 2: Auditing

Section 1: Signature generation

- Parameter generation: The system selects parameters $\{G_1,G_2,e,q,P,H\}$
- Key generation: User randomly selects $x \in_R Z_q$ and computes

$P_{pub} = xP$. The public key is P_{pub} and the secret key is x.

- Signature generation: for a given file $F=\{ m_i \}$ user computes an authenticator $\sigma_i \leftarrow (1/(h(m_i)+x))*P$
- The encrypted file F and σ_i will be stored on cloud.

Section 2: Auditing Scheme

1) Retrieve the file tag f and verify its signature quit if not.Signature verification is done by $e(H(m_i)P+P_{pub},S)=e(p,P)$
2) TPA generates random challenge $Chal=\{(i,v_i)\}_i \in I_i$
3) Cloud server computes $\mu' = \sum_{i \in I} v_i m_i$ and $\sigma = \prod_{i \in I} \sigma_i^{Vi}$
4) Cloud randomly picks $r \leftarrow z_p$ and $R= e(u,v)$ and $r=h(R)$
5) Cloud then computes $\mu=r+r\mu'$ mod p
6) TPA computes r=h(R) then verifies $\{\mu,\sigma,R\}$ using equation. $R \cdot e(\sigma^r,g)=e((\prod_{i=s\,1}^{sc} H(m_i)^{Vi}) \cdot u^u ,v)$

HARDWARE REQUIRED

System	:	Pentium IV 2.4 GHz
Hard Disk	:	40 GB
Floppy Drive	:	1.44 MB
Monitor	:	15 VGA color
Mouse	:	Logitech.
Keyboard	:	110 keys enhanced
RAM	:	2 GB

SOFTWARE REQUIRED

Operating System

Independence of Operating system

Application Libraries

Java and J2EE, web services and cloud storage

Language

J2EE and Java

Front End

HTML DATASET

Table 6: performance of the system

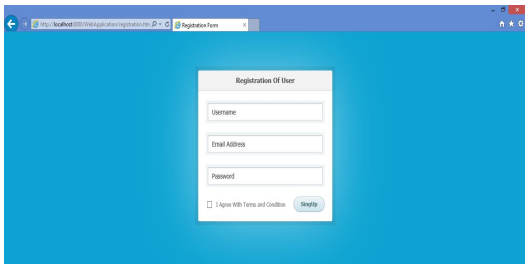
	Our scheme	Existing Scheme
No. Of files	1 or more	1or more
Server computation Time(ms)	Less than 300ms	335ms
TPA computation Time(ms)	Less than 500 ms	530ms

EXPERIMENTAL SETUP

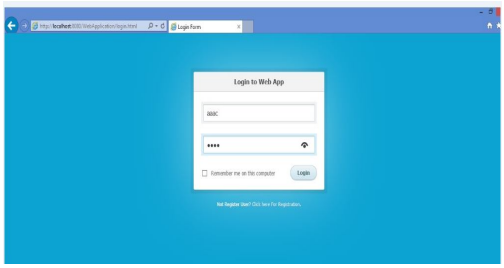
The Bilinear pairing based algorithm for learning public auditing is implemented in Java. the downloadable package available on Internet is used [16]. It is written in Java and runs on almost any platform. The front end used is HTML. Once the project is mounted on cloud then it can be run on any system but internet connection is required.

SNAPSHOTS

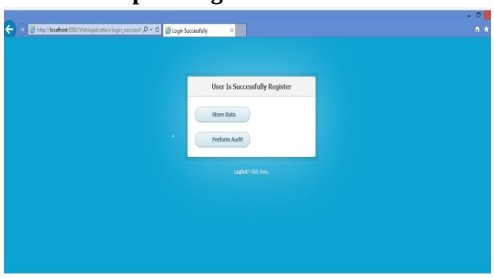
USER REGISTRATION



User Login

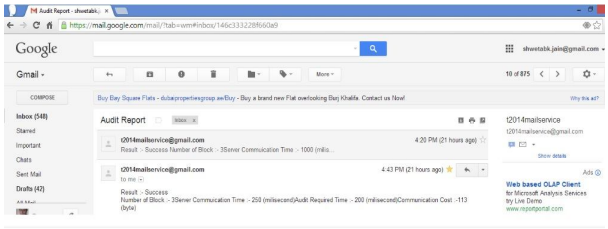


Uploading Files on Cloud



Performing Audit

Sending Email Notification



3. RESULT

In this section the performance of Bilinear Pairing based algorithm for implementing public auditing system discussed. The results provided are based on server computation time and TPA computation Time. Table 10.2 shows average values of the performance parameters such as server computation Time and TPA time. From Table 10.2 it can be seen that, over all the problems, Bilinear Pairing based algorithm performs significantly better than all the other auditing algorithms like MAC and HLA algorithms. On these datasets, the performance of the Bilinear Pairing algorithm is comparable with the best other auditing method. The average accuracy of Bilinear Pairing algorithm is better than other auditing algorithms. Thus, overall, performance of Bilinear Pairing algorithm is better than or comparable to any other auditing algorithms.

4.CONCLUSION

In this paper, we propose a privacy-preserving public auditing system for data storage security in cloud computing. We utilize the homomorphic linear authenticator and random masking to guarantee that the TPA would not learn any knowledge about the data content stored on the cloud server during the efficient auditing process, which not only eliminates the burden of cloud user from the tedious and possibly expensive auditing task, but also alleviates the users' fear of their outsourced data leakage. Considering TPA may concurrently handle multiple audit sessions from different users for their outsourced data files, we further extend our privacy-preserving public auditing protocol into a multiuser setting, where the TPA can perform multiple auditing tasks in a batch manner for better efficiency.The verification technique can be done by further techniques to improve the efficiency and security. System can be enhanced to verify different type of data.

5. REFERENCES

[1.] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," Proc. 14th ACM Conf. Computer and Comm. Security (CCS '07), pp. 598-609, 2007.
[2.] M.A. Shah, R. Swaminathan, and M. Baker, "Privacy- Preserving Audit and Extraction of Digital Contents," Cryptology ePrint Archive, Report 2008/186, 2008.
[3.] A. Juels and J. Burton, S. Kaliski, "PORs: Proofs of Retrievability for Large Files," Proc. ACM Conf. Computer and Comm. Security (CCS '07), pp. 584-597, Oct. 2007.
[4.] Cloud Security Alliance, "Security Guidance for Critical Areas of Focus in Cloud Computing,"
[5.] H. Shacham and B. Waters, "Compact Proofs of Retrievability," Proc. Int'l Conf. Theory and Application of
[6.] Cryptology and Information Security: Advances in Cryptology (Asiacrypt), vol. 5350, pp. 90-107, Dec. 2008.
[7.] C. Wang, K. Ren, W. Lou, and J. Li, "Towards Publicly Auditable Secure Cloud Data Storage Services," IEEE Network Magazine, vol. 24, no. 4, pp. 19-24, July/Aug.2010.
[8.] M.A. Shah, M. Baker, J.C. Mogul, and R. Swaminathan, "Auditing to Keep Online Storage Services Honest," Proc. 11th USENIX Workshop 104th United States Congress, "Health Insurance Portability and Accountability Act of1996 (HIPPA).
[9.] R. Curtmola, O. Khan, and R. Burns, "Robust Remote Data Checking," Proc. Fourth ACM Int'l Workshop Storage Security and Survivability (StorageSS '08), pp. 63-68, 2008.
[10.] K.D. Bowers, A. Juels, and A. Oprea, "Proofs of Retrievability: Theory and Implementation," Proc. ACM Workshop Cloud Computing Security (CCSW '09), pp. 43- 54, 2009.
[11.] D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing," J. Cryptology, vol. 17, no. 4, pp. 297-319, 2004.
[12.] A.L. Ferrara, M. Green, S. Hohenberger, and M. Pedersen, "Practical Short Signature Batch Verification," Proc. Cryptographers' Track at the RSA Conf. 2009 on Topics in Cryptology (CT-RSA), pp. 309-324, 2009.
[13.] C. Wang, K. Ren, W. Lou, and J. Li, "Towards Publicly Auditable Secure Cloud Data Storage Services," IEEE Network Magazine, vol. 24, no. 4, pp. 19-24, July/Aug. 2010.
[14.] M.A. Shah, M. Baker, J.C. Mogul, and R. Swaminathan, "Auditing to Keep Online Storage Services Honest," Proc. 11th USENIX Workshop Hot Topics in Operating Systems (HotOS '07), pp. 1-6, 2007.

[15.] 104th United States Congress, "Health Insurance Portability and Accountability Act of 1996 (HIPPA)," http://aspe.hhs.gov/admnsimp/pl104191.htm, 1996.
[16.] R. Curtmola, O. Khan, and R. Burns, "Robust Remote Data Checking," Proc. Fourth ACM Int'l Workshop Storage Security and Survivability (StorageSS '08), pp. 63-68, 2008.
[17.] A.L. Ferrara, M. Green, S. Hohenberger, and M. Pedersen,"Practical Short Signature Batch Verification," Proc. Cryptographers'Track at the RSA Conf. 2009 on Topics in Cryptology (CT-RSA),pp. 309-324, 2009.
[18.] C. Wang, Q. Wang, K. Ren, and W. Lou, "Towards Secure and Dependable Storage Services in Cloud Computing," IEEE Trans. Service Computing, vol. 5, no. 2, 220-232, Apr.-June 2012.
[19.] M. Bellare and G. Neven, "Multi-Signatures in the Plain PublicKey Model and a General Forking Lemma," Proc. ACM Conf.Computer and Comm. Security (CCS), pp. 390-399, 2006.