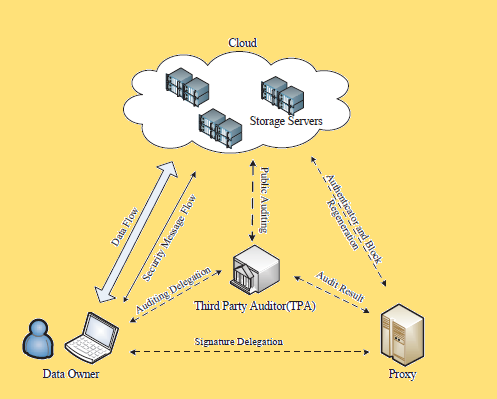
**Privacy-Preserving Public Auditing for Regenerating-**

**Code-Based Cloud Storage**

**Abstract:**

To protect outsourced data in cloud storage against corruptions, adding fault tolerance to cloud storage together with data integrity checking and failure reparation becomes critical. Recently, regenerating codes have gained popularity due to their lower repair bandwidth while providing fault tolerance. Existing remote checking methods for regenerating-coded data only provide private auditing, requiring data owners to always stay online and handle auditing, as well as repairing, which is sometimes impractical. In this paper, we propose a public auditing scheme for the regenerating-code-based cloud storage. To solve the regeneration problem of failed authenticators in the absence of data owners, we introduce a proxy, which is privileged to regenerate the authenticators, into the traditional public auditing system model. Moreover, we design a novel public verifiable authenticator, which is generated by a couple of keys and can be regenerated using partial keys. Thus, our scheme can completely release data owners from online burden. In addition, we randomize the encode coefficients with a pseudorandom function to preserve data privacy. Extensive security analysis shows that our scheme is provable secure under random oracle model and experimental evaluation indicates that our scheme is highly efficient and can be feasibly integrated into the regenerating code- based cloud storage.

**Architecture Diagram:**

****

**Existing System:**

Cloud storage is now gaining popularity because it offers a flexible on-demand data outsourcing service with appealing benefits: relief of the burden for storage management, universal data access with location independence, and avoidance of capital expenditure on hardware, software, and personal maintenances.

**Disadvantages:**

It is noted that data owners lose ultimate control over the fate of their outsourced data; thus, the correctness, availability and integrity of the data are being put at risk.

**Proposed System:**

The integrity of outsourced data without a local copy have been proposed under different system and security models up to now. The most significant work among these studies are the PDP (*provable data possession*) model and POR (*proof of retrievability*) model, which were originally proposed for the single-server scenario by Considering that files are usually striped and redundantly stored across multi-servers or multi-clouds, explore integrity verification schemes suitable for such multi-servers or multi clouds setting with different redundancy schemes, such as *replication*, *erasure codes*, and, more recently, *regenerating* *codes*.

**Advantages:**

We focus on the integrity verification problem in regenerating-code-based cloud storage, especially with the functional repair strategy.

**Implementation Modules:**

1. ***Regenerating Codes:***
2. ***Design Goals***
3. ***Definitions of Our Auditing Scheme***
4. **Enabling Privacy-Preserving Auditable**

**Regenerating Codes:**

Regenerating codes are first introduced for distributed storage to reduce the repair bandwidth. Viewing cloud storage to be a collection of *n* storage servers, data file *F* is encoded and stored redundantly across these servers. Then *F* can be retrieved by connecting to any *k*-out-of-*n* servers, which is termed the MDS2-property. When data corruption at a server is detected, the client will contact *ℓ* healthy servers and download *β′* bits from each server, thus regenerating the corrupted blocks without recovering the entire original file.

***Design Goals:***

To correctly and efficiently verify the integrity of data and keep the stored file available for cloud storage, our proposed auditing scheme should achieve the following properties:

*• Public Auditability*: to allow TPA to verify the intactness of the data in the cloud on demand without introducing additional online burden to the data owner.

*• Storage Soundness*: to ensure that the cloud server can never pass the auditing procedure except when it indeed manage the owner’s data intact.

*• Privacy Preserving*: to ensure that neither the auditor nor the proxy can derive users’ data content from the auditing and reparation process.

*• Authenticator Regeneration*: the authenticator of the repaired blocks can be correctly regenerated in the absence of the data owner.

*• Error Location*: to ensure that the wrong server can be quickly indicated when data corruption is detected.

***Definitions of Our Auditing Scheme***

Our auditing scheme consists of three procedures: Setup, Audit and Repair. Each procedure contains certain polynomial-time algorithms as follows:

Setup: The data owner maintains this procedure to initialize the auditing scheme. *KeyGen*(1*κ*) *→* (*pk, sk*): This polynomial-time algorithm is run by the data owner to initialize its public and secret parameters by taking a security parameter *κ* as input.

*Degelation*(*sk*) *→* (*x*): This algorithm represents the interaction between the data owner and proxy. The data owner delivers partial secret key *x* to the proxy through a secure approach.

*Sig And BlockGen* (*sk, F*) *→* (\_*,* *, t*): This polynomial time algorithm is run by the data owner and takes the secret parameter *sk* and the original file *F* as input, and then outputs a coded block set , an authenticator set \_ and a file tag *t*.

Audit: The cloud servers and TPA interact with one another to take a random sample on the blocks and check the data intactness in this procedure.

*Challenge*(*Finfo*) *→* (*C*): This algorithm is performed by the TPA with the information of the file *Finfo* as input and a challenge *C* as output.

*ProofGen*(*C,*\_*,*  ) *→* (*P*): This algorithm is run by each cloud server with input challenge *C*, coded block set and authenticator set \_, then it outputs a proof *P*.

*Verify*(*P, pk, C*) *→* (0*,* 1): This algorithm is run by TPA immediately after a proof is received. Taking the proof *P*, public parameter *pk* and the corresponding challenge *C* as input, it outputs 1 if the verification passed and 0 otherwise.

Repair: In the absence of the data owner, the proxy interacts with the cloud servers during this procedure to repair the wrong server detected by the auditing process.

**Enabling Privacy-Preserving Auditable:**

The privacy protection of the owner’s data can be easily achieved through integrating with the random proof blind technique or other technique . However, all these privacy-preservation methods introduce additional computation overhead to the auditor, who usually needs to audit for many clouds and a large number of data owners; thus, this could possibly make it create a performance bottleneck. Therefore, we prefer to present a novel method, which is more light-weight, to mitigate private data leakage to the auditor. Notice that in a regenerating-code-based cloud storage, data blocks stored at servers are coded as linear combinations of the original blocks Supposing that the curious TPA has recovered *m* coded blocks by elaborately performing *Challenge-Response* procedures and solving systems of linear equations], the TPA still requies to solve another group of *m* linearly independent equations to derive the *m* native blocks.

# Configuration:-

# H/W System Configuration:-

# Processor - Pentium –III

Speed - 1.1 Ghz

RAM - 256 MB(min)

Hard Disk - 20 GB

Floppy Drive - 1.44 MB

Key Board - Standard Windows Keyboard

Mouse - Two or Three Button Mouse

Monitor - SVGA

# S/W System Configuration:-

* Operating System :Windows/XP/7.
* Application Server : Tomcat5.0/6.X
* Front End : HTML, Java, Jsp
* Scripts : JavaScript.
* Server side Script : Java Server Pages.
* Database : Mysql 5.0
* Database Connectivity : JDBC.