IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 3 Issue 6, June 2016
ISSN (Online) 2348 – 7968 | Impact Factor (2015) - 4.332
www.ijiset.com

A Secured Public Auditing for Regenerating-Code-Based Cloud Storage

Miss. Nirupamashree C [1], Mrs. Pushpa R [2]

[1]PG Student, Department of Computer Science and Engineering,
[2]AssistentProfessor, Department of Computer Science and Engineering,
Sri Siddhartha institute of technology, Tumkur, Karnataka, India

***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.

Index Terms—Cloud storage, regenerating codes, public audit, privacy preserving, authenticator regeneration, proxy, privileged, provable secure.

I. INTRODUCTION

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, etc., [1]. Nevertheless, this new paradigm of data hosting service also brings new security threats toward users data, thus making individuals or enterprisers still feel hesitant. 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. On the one hand, the cloud service is usually faced with a broad range Of internal/external adversaries, who would maliciously delete or corrupt users' data; on the other hand, the cloud Service providers may act dishonestly, attempting to hide data loss or corruption and claiming that the files are still correctly stored in the cloud for reputation or monetary reasons. Thus it makes great sense for users to implement an efficient protocol to perform periodical verifications of their outsourced data to ensure that the cloud indeed maintains their data correctly. Many mechanisms dealing with 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 Ateniese *et al.* [2] and Juels and Kaliski [3], respectively. Considering that files are usually striped and redundantly stored across multi-servers or multi-clouds, [4] explore integrity verification schemes suitable for such **multi-servers** or **multi-clouds** set with different redundancy schemes, such as *replication*, *erasure codes*, and, more recently, *regenerating codes*.

In this paper, we focus on the integrity verification problem in **regenerating-code-based cloud storage**, especially with the functional repair strategy. Similar studies have been performed by Chen *et al.* [7] and Chen and Lee [8] separately and independently. [7] extended the single-server CPOR scheme to the regenerating-code-scenario; [8] designed and implemented a data integrity protection (DIP) scheme for FMSR based cloud storage and the scheme is adapted to the thin-cloud setting.1 However, both of them are designed for private audit, only the data owner is allowed to verify the integrity and repair the faulty servers. Considering the large size of the outsourced data and the user's constrained resource capability, the tasks of auditing and reparation in the cloud can be formidable and expensive for the users . The overhead of using cloud storage should be minimized as much as possible such that a user does not need to perform too many operations to their outsourced data (in additional to retrieving it) . In particular, users may not want to go through the complexity in verifying and reparation. The auditing schemes in [7] and [8] imply the problem that users need to always stay online, which may impede its adoption in practice, especially for long-term archival storage. To fully ensure the data integrity and save the users' computation resources as well as online burden, we propose a public auditing scheme for the regenerating-code-based cloud storage, in which the integrity checking and regeneration (of failed data blocks and authenticators) are implemented by a third-party auditor and a semi-trusted proxy separately on behalf of the data owner. Instead of directly adapting the existing public auditing scheme to the multi-server setting, we design a novel authenticator, which is more appropriate for regenerating codes. Besides, we "*encrypt*" the coefficients to protect data privacy against the auditor, which is more lightweight than applying the proof blind technique in and and data blind method. Several challenges and threats spontaneously arise in our new system model with a proxy , and security analysis shows

that our scheme works well with these problems. Specifically, our contribution can be summarized by the following aspects We design a novel homomorphic authenticator based on BLS signature, which can be generated by a couple of secret keys and verified publicly. Utilizing the linear subspace of the regenerating codes, the authenticators can
be computed efficiently. Besides, it can be adapted for data owners equipped with low end computation devices (e.g. Tablet PC etc.) in which they only need to sign the native blocks.

• To the best of our knowledge, our scheme is the first to allow privacy-preserving public auditing for regenerating code-based cloud storage. The coefficients are masked by a PRF (Pseudorandom Function) during the Setup phase to avoid leakage of the original data. This method is lightweight and does not introduce any computational overhead to the cloud servers or TPA.

• Our scheme completely releases data owners from online burden for the regeneration of blocks and authenticators at faulty servers and it provides the privilege to a proxy for the reparation.

• Optimization measures are taken to improve the flexibility and efficiency of our auditing scheme; thus, the storage overhead of servers, the computational overhead of the data owner and communication overhead during the audit phase can be effectively reduced.

• Our scheme is provable secure under *random oracle model* against adversaries

II.PRELIMINARIES AND PROBLEM STATEMENT

Notations and Preliminaries

1) Regenerating Codes: Regenerating codes are first introduced by Dimakis *et al.* 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 termed the MDS2-property. When data corruption at a server is detected, the client will contact *L* healthy servers and download *B* bits from each server, thus regenerating the corrupted blocks without recovering the entire original file. Dimakis *et al.* showed that the repair bandwidth Y= *LB* can be significantly reduced with $L \geq k$. Furthermore, they analyzed the fundamental tradeoff between the storage cost X and the repair bandwidth Y, then presented two extreme and practically relevant points on the optimal tradeoff curve: *the minimum bandwidth regenerating* (MBR) point, which represents the operating point with the least possible repair bandwidth, and *the minimum storage regenerating* (MSR) point, which corresponds to the least possible storage cost on the servers

Moreover, according to whether the corrupted blocks can be exactly regenerated, there are two versions of repair strategy: exact repair and functional repair . Exact repair strategy requires the repaired server to store an exact replica of the corrupted blocks, while functional repair indicates that the newly generated blocks are different from the corrupted ones with high probability. As one basis of our work, the functional repair regenerating codes are non-systematic and do not perform as well for read operation as systematic codes, but they really make sense for the scenario in which data repair occurs much more often than read, such as regulatory storage, data escrow and long-term archival storage [7].

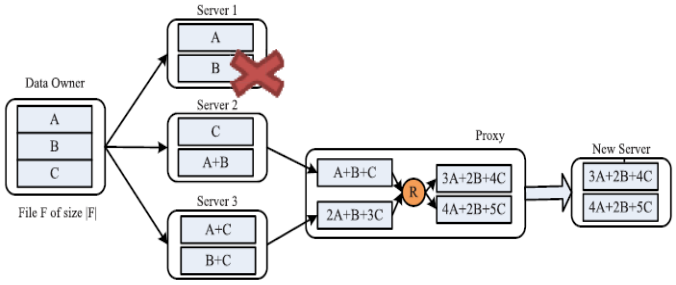


Figure 1

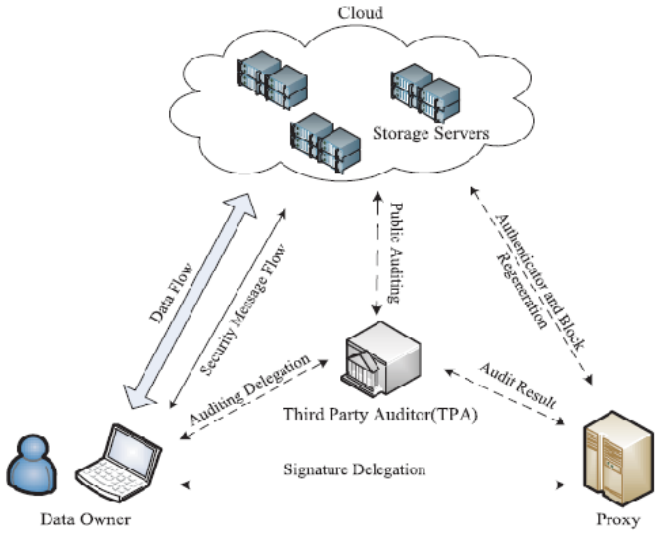


Figure 2

III SYSTEM MODEL

We consider the auditing system model for Regenerating- Code-based cloud storage as Fig.2, which involves four entities: *the data owner*, who owns large amounts of data files to be stored in the cloud; *the cloud*, which are managed by the cloud service provider, provide storage service and have significant computational resources; *the third party auditor* (TPA), who has expertise and capabilities to conduct public audits on the coded data in the cloud, the TPA is trusted and its audit result is unbiased for both data owners and cloud servers; and *a proxy agent*, who is semi-trusted and acts on behalf of the data owner to regenerate authenticators and data blocks on the failed servers during the repair procedure. Notice that the data owner is restricted in computational and storage resources compared to other entities and may becomes off-line even after the data upload procedure. The proxy, who would always be online, is supposed to be much more powerful than the data owner but less than the cloud servers in terms of computation and memory

capacity. To save resources as well as the online burden potentially brought by the periodic auditing and accidental repairing, the data owners resort to the TPA for integrity verification and delegate the reparation to the proxy. Compared with the traditional public auditing system model, our system model involves an additional proxy agent. In order to reveal the rationality of our design and make to be more clear and concrete, we consider such a reference scenario: A company employs a commercial regenerating-code-based public cloud and provides long-term archival storage service for its staffs, the staffs are equipped with low end computation devices (e.g., Laptop PC, Tablet PC, etc.) and will be frequently off-line. For public data auditing, the company relies on a trusted third party organization to check the data integrity; Similarly, to release the staffs from heavy online burden for data and authenticator regeneration, the company supply a powerful workstation (or cluster) as the proxy and provide proxy reparation service for the staffs' data.

THREAT MODEL

We assume that some blocks stored in server S_i are corrupted at some time, the adversary may launch the following attacks in order to prevent the auditor from detecting the corruption:

- *Replace Attack:* The server S_i may choose another valid and intact pair of data block and authenticator to replace the corrupted pair, or even simply store the blocks and authenticators at another healthy server S_j , thus successfully passing the integrity check.
- *Replay Attack:* The server may generate the proof from an old coded block and corresponding authenticator to pass the verification, thus leading to a reduction of data redundancy to the point that the original data becomes unrecoverable.
- *Forge Attack:* The server may forge an authenticator for modified data block and deceive the auditor.
- *Pollution Attack:* The server may use correct data to avoid detection in the audit procedure but provide corrupted data for repairing; thus the corrupted data may pollute all the data blocks after several epoches. With respect to the TPA, we assume it to be honest but curious. It performs honestly during the whole auditing procedure but is curious about the data stored in the cloud.

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$(1^k)\rightarrow$ *(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) $\rightarrow$ *(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.

SigAndBlockGen(sk, F) $\rightarrow (\omega, \Psi, t)$*:* 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 Ψ, an authenticator set ω and a file tag *t*.

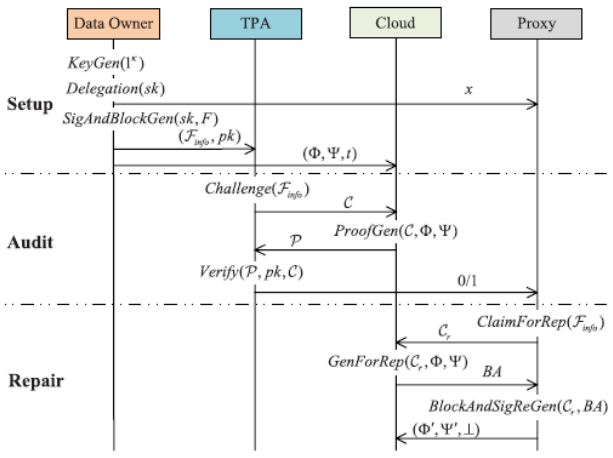
Audit: The cloud servers and TPA interact with one another to take a random sample on the blocks and check the data intactness in this procedure.

Challenge(F_{info}*)* $\rightarrow$ *(C):* This algorithm is performed by the TPA with the information of the file F_{info} as input and a challenge *C* as output.

Proof Gen(C,ω *,*Ψ*)*$\rightarrow$*(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*.

Veri f y(P, pk, C)$\rightarrow$ *(*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.



IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 3 Issue 6, June 2016
ISSN (Online) 2348 – 7968 | Impact Factor (2015) - 4.332
www.ijiset.com

III. THE PROPOSED SCHEME

In this section we start from an overview of our auditing scheme, and then describe the main scheme and discuss how to generalize our privacy-preserving public auditing scheme. Furthermore, we illustrate some optimized methods to improve its performance.

A. Overview

Although [7], [8] introduced private remote data checking schemes for regenerating-code-based cloud storage, there are still some other challenges for us to design a public auditable version.

First, although a direct extension of the techniques in [2], [12], and [15] can realize public verifiability in the multi-servers setting by viewing each block as a set of segments and performing spot checking on them, such a straight forward method makes the data owner generate tags for all segments independently, thus resulting in high computational overhead. Considering that data owners commonly maintains limited computation and memory capacity, it is quite significant for us to reduce those overheads. Second, unlike cloud storage based on traditional erasure code or replication, a fixed file layout does not existing the regenerating-code-based cloud storage. During the repair phase, it computes out new blocks, which are totally different from the faulty ones, with high probability.

Thus, a problem arises when trying to determine how to regenerate authenticators for the repaired blocks. A direct solution, which is adopted in [7], is to make data owners handle the regeneration. However, this solution is not practical because the data owners will not always remain online through the life-cycle of their data in the cloud, more typically, it becomes off-line even after data uploading. Another challenge is brought in by the proxy in our system model (see Section II-C).

The following parts of this section shows our solution to the problems above. First, we construct a BLS-based [17] authenticator, which consists of two parts for each segment of coded blocks. Utilizing its homomorphic property and the linearity relation amongst the coded blocks, the data owner is able to generate those authenticators in a new method, which is more efficient compared to the straightforward approach. Our authenticator contains the information of encoding coefficients to avoid data pollution in the reparation with wrong coefficients. To reduce the bandwidth cost during the audit phase, we perform a batch verification over all α blocks at a certain server rather than checking the integrity of each block separately as [7] does. Moreover, to make our scheme secure against the replace attack and replay attack, information about the indexes of the server, blocks, and segments are all embedded into the authenticator. Besides, our primitive scheme can be easily improved to support privacy-preserving through the masking of the coding coefficients with a keyed PRF.

All the algebraic operations in our scheme work over the field *GF(p)* of modulo *p*, where *p* is a large prime (at least 80 bits).

B. Construction of Our Auditing Scheme

Considering the regenerating-code-based cloud storage with parameters $(n, k, \_, \alpha, \beta)$, we assume $\beta = 1$ for simplicity. Let G and G_T be multiplicative cyclic groups of the same large prime order p, and $e : G \times G \to G_T$ be a bilinear pairing map as introduced in the preliminaries. Let g be a generator of G and $H(\cdot) : \{0, 1\}^* \to G$ be a secure hash function that maps strings uniformly into group G. Table I list the primary notations and terminologies used in our scheme description.

Setup: The audit scheme related parameters are initialized in this procedure.

KeyGen$(1^\kappa) \to (pk, sk)$*:* The data owner generates a random signing key pair (spk, ssk), two random elements $x, y \xleftarrow{R} Z_p$ and computes $pk_x \leftarrow g^x$, $pk_y \leftarrow g^y$. Then the secret parameter is $sk = (x, y, ssk)$ and the public parameter is $pk = (pk_x, pk_y, spk)$.

Delegation$(sk) \to (x)$*:* The data owner sends encrypted x to the proxy using the proxy's public key, then the proxy decrypts and stores it locally upon receiving.

SigAndBlockGen$(sk, F) \to (\quad ,, t)$*:* The data owner uniformly chooses a random identifier $ID \xleftarrow{R} \{0, 1\}^*$, a random symbol $u \xleftarrow{R} G$, one set $\_ = \{w_1, w_2, \ldots w_m\}$

TABLE –I
COMPARISON OF DIFFERENT AUDIT SCHEMES FOR REGENERATING CODE BASED CLOUD STORAGE

Items	Chen Bo [7]	Henry C. H. Chen [8]	Ours
Public Auditability	No	No	Yes
Privacy Preserving	Yes	Yes	Yes
Owners off-line support	No	No	Yes
Time for Faulty server localization	$O(1)$	$O(C_n^k(n-k)\alpha)$	$O(1)$
Server Storage Overhead	$(s\alpha + m\alpha + \alpha)\lvert p\rvert$	$(nm\alpha + s(\frac{n^r - k^r}{k^r})\alpha + \alpha)\lvert p\rvert^2$	$(s\alpha + m\alpha)\lvert p\rvert$
Communication Overhead(Audit)	$(m+2)\alpha\lvert p\rvert$	$(nm + c)\alpha\lvert p\rvert$	$(m+2)\lvert p\rvert$
Communication Overhead(Repair)	$(m\alpha + s + 1)\lvert p\rvert$	$(nm\alpha + s + 1)\lvert p\rvert$	$(2s+m)\lvert p\rvert$

IV. SECURITY ANALYSIS

In this section, we first elaborate on the correctness of verification in our auditing scheme and then formally evaluate the security by analyzing its fulfillment of soundness, regeneration-unforgeable and secure guarantee against replay attack.

A. Correctness

There are two verification process in our scheme, one for spot checking during the **Audit** phase and another for block integrity checking during the **Repair** phase.

Theorem 1: Given a cloud server i storing data blocks $_i$ and accompanied authenticators $_i$, TPA is able to correctly verify its possession of those data blocks during audit phase, and the proxy can correctly check the integrity of downloaded blocks during repair phase.

B. Soundness

Following from paper [10], we say that our auditing protocol is sound if any cheating server that convinces the verification algorithm that it is storing the coded blocks and corresponding coefficients is actually storing them. Before proving the soundness, we will first show that the authenticator as Eq.(10) is unforgeable against malicious cloud servers (referred to as *adversary* in the following

definition)under the *random oracle model*. Similar to the standard notion of security for a signature scheme ,

C. Regeneration-Unforgeable

Noting that the semi-trusted proxy handles regeneration of authenticators in our model, we say our authenticator is *regeneration-unforgeable* if it satisfies the following theorem.

Theorem 4: The adversary (or proxy) can only regenerate a forgery of authenticator for invalid segment from certain coded block (augmented) and pass the next verification with negligible probability, except that it implements the Repair procedure correctly.

Proof: See Appendix E.

D. Resistant to Replay Attack

Theorem 5: Our public auditing scheme is resistant to replay attack mentioned in [7, Appendix B], since the repaired server maintains identifier $\eta\_$ which is different with the corrupted server η.

CONCLUSION

In this paper, we propose a public auditing scheme for the regenerating-code-based cloud storage system, where the data owners are privileged to delegate TPA for their data validity checking. To protect the original data privacy against the TPA, we randomize the coefficients in the beginning rather than applying the blind technique during the auditing process. Considering that the data owner cannot always stay online in practices, in order to keep the storage available and verifiable after a malicious corruption, we introduce a semi-trusted proxy into the system model and provide a privilege for the proxy to handle the reparation of the coded blocks and authenticators. To better appropriate for the regenerating code-scenario, we design our authenticator based on the BLS signature. This authenticator can be efficiently generated by the data owner simultaneously with the encoding procedure. Extensive analysis shows that our scheme is provable secure, and the performance evaluation shows that our scheme is highly efficient and can be feasibly integrated into a regenerating-code-based cloud storage system.

REFERENCES

[1] M. Armbrust *et al.*, "Above the clouds: A Berkeley view of cloud computing," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2009-28, 2009.

[2] G. Ateniese *et al.*, "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2007, pp. 598–609.

[3] A. Juels and B. S. Kaliski, Jr., "PORs: Proofs of retrievability for large files," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 2007, pp. 584–597.

[4] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-replica provable data possession," in *Proc. 28th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2008, pp. 411–420.

[5] K. D. Bowers, A. Juels, and A. Oprea, "HAIL: A high-availability and integrity layer for cloud storage," in *Proc. 16th ACM Conf. Comput. Commun. Secur.*, 2009, pp. 187–198.

[6] J. He, Y. Zhang, G. Huang, Y. Shi, and J. Cao, "Distributed data possession checking for securing multiple replicas in geographicallydispersed clouds," *J. Comput. Syst. Sci.*, vol. 78, no. 5, pp. 1345–1358, 2012.

[7] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote data checking for network coding-based distributed storage systems," in *Proc. ACM Workshop Cloud Comput. Secur. Workshop*, 2010, pp. 31–42.

[8] H. C. H. Chen and P. P. C. Lee, "Enabling data integrity protection in regenerating-coding-based cloud storage: Theory and implementation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 2, pp. 407–416, Feb. 2014.

[9] K. Yang and X. Jia, "An efficient and secure dynamic auditing protocol for data storage in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 9, pp. 1717–1726, Sep. 2013.

[10] Y. Zhu, H. Hu, G.-J. Ahn, and M. Yu, "Cooperative provable data possession for integrity verification in multicloud storage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 12, pp. 2231–2244, Dec. 2012.