RESEARCH ARTICLE OPEN ACCESS

Efficient Data Integrity Algorithm for Outsourced Data in Cloud Environment

Pramod Kumar*, MMS Rauthan**, Vikram Kapoor***
**(Department Of Computer Science, Uttarakhand Technical University, India*
*** (Department Of Computer Science And Engineering, HNB Garhwal University, India*
*** (Department Of Computer Science, Uttarakhand Technical University, India*
Corespondind Auther : Pramod Kumar

ABSTRACT
With The Rapid Advances In Communication Services And Increasing Growth In Outsourcing Of Data To Cloud Environment; Concerns Of Data Integrity Of Outsourced Data Is On The Rise. The Evidence Of Data Integrity Being Tampered And Up-To Date, Seem To Be Of Immediate Concern. Current Data Integrity Techniques Are Inefficient To Provide Fast User Data Verification Or To Check The Correctness Of Data Without Additional Storage Overhead At The Data Owner (DO). For Data Owners With Large Data Files, Existing Techniques Are Not Feasible Solutions. In This Paper, We Propose An Efficient Alternative Technique For The Integrity Of Outsourced Data Using Optimal Bloom Filters (OBF). We Propose The Basic Data Integrity Method And Discuss Various Alternatives To Design The Optimal Data Integrity Model. We Present A Detailed Analysis And Experimental Results For One Of The Alternative. These Results Are Compared With The Current Data Integrity Algorithms Such As SHA-1 And MD5. The Proposed Technique Using Bloom Filter Implementations Is Highly Space-Efficient At The Expense Of Extra Computational Overhead And The Multicore Implementations Have Significantly Reduced The Execution Time. The Presented Results Demonstrate That Employing Bloom Filters To Enforce Data Integrity For Outsourced Data In Cloud Environments Is Feasible And Efficient Than Traditional Techniques.

Keywords - Data Integrity, Hashing, Outsourced Data, Cloud.

Date of Submission: 21-02-2018 Date of acceptance 8-03-2018

I. INTRODUCTION

Rapid Developments In Communication And Networks Have Made Huge Data File Sharing More Effective. Consequently, The Demand For Rich Media Applications, Such As Multimedia Mails, High Definition Audio/Video Sharing Has Grown Tremendously. The Volume Of Data Being Used By Those Applications Have Also Grown Exponentially. As A Result, The Costs Of IT Infrastructure And IT Support Staff Is Rising. Therefore, Cloud Computing Has Become An Attractive Technology Of IT Service. In Cloud Computing Users Can Use Infrastructures, Applications, Storage, Network, Servers And Other Computing Resources, Which Is A Shared Pool Of Computing Resources That Can Be Easily Accessed Through Network Connections. This Cloud Computing Service Model Offers Users Seemingly Unlimited Computing Resources Without Acquisition And Maintenance Costs. Storage, As One Of The Most Demanding Computing Resources Among The First Being Moved Into The Cloud. This Type Of Cloud Computing Services Is Known As Cloud Storage And In This Model The Service Provider Rent Spaces In Their Large-Pool Of Storage Infrastructure To Organizations And Individuals Such As Google Drive And Amazon Cloud. Cloud Computing Provides:

I) On-Demand Self-Service
Ii) Broadband Network Access
Iii) Resource Multiplexing
Iv) Rapid Elasticity

Besides The Key Advantages Of Cost Saving, Cloud Storage Can Facilitate Storage Of Sensitive Data, Such As Financial, Personal, Or Medical. Since Storage Is One Of The Core Infrastructure In Clouds, So Security And Privacy Of Data In Cloud Storage Is One Of The Key Issue. The Consequences Of Security Breaches Could Be Seriously Damaging To Both Service Providers And Users. The Essential Requirement Of Cloud Computing In Data Storage And Data Communication Is Data Integrity. The Parity Checking Codes, Error Detection Codes, And Error Correction Codes Have Long Been Used In Data Communication [1]. Cyclic Redundancy (CRC) Is One Such Code Used As A Checksum In The Computer Network Since Two Decade [2].

In Main Memory Technologies Hash Codes Have Been Used To Ensure Data Integrity [3]. A RAID Technology Have Been Used To Store The Parity Corresponding To Each Disk Block To

Detect/Correct Any Data Integrity Violations [4]. However, This Threat Model Comprised Of Unintentional Corruption Of Data Rather Than Intentional Altering [1-4]. Today, Cloud Computing Seems To Be The Fastest Evolving Technology With The Availability Of Higher Network Bandwidths [5]. In Addition To Several Private Clouds, Companies Such As Google (Gmail, Google Drive) And Microsoft (Hotmail, Onedrive) Offer These Services To The Public At Low Cost. The Rate At Which Data Is Being Collected By Cloud Service Provider (E.G., Facebook, Whatsapp) From Data Owners Is Also Growing [6]. This Has Resulted In Data And Process Outsourcing To The Clouds With Rapid Rate. However, Outsourcing Of Data Has Several Concerns Regarding Data Integrity, Security, And Privacy To The Data Owners. In Terms Of Data Integrity And Currency, The Concerns Of The Owner Are:

(I) How It Will Be Ensure That Data File Has Not Been Tampered.
(Ii) Assurance That All Updates Sent To It Have Been Carried Out Exactly Once.
(Iii) In Case, The Data Owner Is Different From The Data User, How Can The Data User Be Assured That The Provided Data Has Not Been Tampered With And Is Current?
(Iv) How Can An Owner Will Recover Its Tampered Files.

In This Paper, We Focus On Issue (I). In Particular, We Focus On File Integrity.

The Use Of Hashes Has Been The Most Practical Approach To Check The Integrity Of Files. While The Simplest Data Integrity Schemes Suggest Creating One Hash Value For Each Data Block Of File, A More Complex Mechanisms Create And Store The Hashes As A Merkel Tree [7]. In A Merkle Hash Tree, The Parent Node Is A Hash Of The Data Obtained By Concatenating The Hashes Of Its Two Children Data And This Process Is Continued Till The Root Hash Is Evaluated And This Root Hash Is Again Authenticated By The Client. Now For Every Request From The Client, The Server Generates A Verification Object (VOB) Which Is A Collection Of All The Necessary Hashes Required By The Client To Re-Compute The Root Hash And Verify Its Authenticity. Some Data Integrity Techniques Have Uses A Universal-Hash MAC Tree To Improve The Performance [8]. All These Schemes Involve Significant Processing Due To Complex Structure Of Hash Functions And Storage Overhead As Hash Value For Each Block At The Owner Side When Data Is Stored And Retrieved From The Cloud To Verify Data Blocks. In This Paper, We Propose An Efficient Data Structure Called Bloom Filters That Is Used To Check Whether A Data Block Is A Member Of A Set [10]. The Availability Of Multicore Processors Makes The Additional Computing Overhead Of Hashes Insignificant Since Bloom Filters Can Easily Programmed To Parallel Implementation Using Shared Memory Programming. Several Systems Such As Google's Bigtable [11], High-Speed Traffic Measurement [12] And Network Forensics For Iptraceback [14] Are Using Bloom Filters For Integrity Purpose. Zhang Et Al Have Implemented Bloom Filters For Data Integrity In Cloud Database Environments [17].

In This Paper, We Have Used Bloom Filters To Replace The Individual Hashes For Data Blocks To Ensure Integrity And We Show That Parallel Bloom Filter Performs Better Than The Existing Hash Based Solutions For Data Integrity. In This Technique, For Each Data Block Of File, We Compute A Set Of Hash Functions On The Data Block Of File, The Block Number, And The Version Number, And Include Them In A Bloom Filter. The Bloom Filter Can Then Be Either Stored At The Client Or Stored Along With The Data In Cloud Storage. We Can Consider To Store The Bloom Filter At The Server In An Encrypted Form Or As It Is.

The Paper Is Organized As Follows. In Section 2, We Briefly Summarize The Previous Work In Data Integrity And Bloom Filters. Section 3 Describes The Proposed Work. Section 4 Summarizes The Results. Finally, Section 5 Concludes The Paper With The Contributions Of This Paper.

II. LITERATURE REVIEW

The Essential Requirement Of Cloud Computing In Data Storage And Data Communication Is Data Integrity. Several Systems Have Uses Hashes For Data Integrity In The Last Two Decade [7-8]. However, Computing And Storing Hashes Has Significant Cost For Each Data Block In Resource-Constrained Systems. In 2005, Opera Et Al [19] Suggested A Method Using Tweakable Encryption Scheme (TES) And Perform Hashing Based On The Randomness Of The Data. The Data Is Divided Into Fixed Size Blocks And The Data Blocks Are Encrypted Using TES. Data Owner (DO) Is Responsible For Maintaining The File Integrity And For This It Stores A Block Identifier And A Hash Of The Block. To Reduce The Local Storage Overhead, The Entropy Of Each Block Is Calculated And No Hash Is Stored Locally For Low Entropy Blocks. In 2009, Yun Et Al. [8] Proposed A Novel Method That Uses MAC Tree Data Structure To Verify Data Integrity. In This Technique Each Encrypted Data Block Of File Is Stored At The Untrusted Server. A Message Authentication Code MAC Is Computed For Each Block Using The Block Cipher And The Nonce And Further A MAC Tree Is Constructed For All The Nonce Hashes Where Each External Node In The Order Contains The Nonce Used For The Corresponding Block To Calculate The Cipher

Block And The Hash. Kumar Et Al. [12] And Li Et Al. [13] Has Been Use Merkle Hash Trees As Data Structure To Store The Hashes Of The Blocks Of Data Files. In These Techniques Each External Node Of The Tree Contains The Hash Of The Corresponding Block. Opera Et Al's Method [19] Significantly Reduces The Space Overhead But It Compromises On The Strength Of Data Integrity. Mykletun Et Al. [20] Introduces A Model For Authentication In Outsourced Data. They Developed Techniques To Ensure That The Results Received For A Query From An Outsourced Database Are Not Tampered With And Are Authentic. Our Proposed Technique Uses Bloom Filters That Are Efficient And Provide Security With Data Integrity. This Technique Is Also Computationally Fast On The Server.

III. PROPOSED SCHEME

This Technique Employed Bloom Filters Which Is A Vector Of M Bits And All Bits Are Set To Zero Initially. Bloom Filters Has A Set Of K Basic Hash Methods, Each Of Which Can Take Any Key As Input And Return An Index (0, M-1) Into The Vector As Result. In This Data Structure Two Operations Are Defined:

(I) To Insert Message We Compute K Hash Values Of The Message And Setting To 1 The Corresponding Bits In The Vector;

(Ii) To Query A Message We Compute K Hash Values Of The Element And Test If The Corresponding Bits Are 1 In The M-Bit Vector.

When We Use Hash Functions, There Is A Chance Of Getting A Positive Answer While Querying For An Element Which Was Not Stored Into The Bloom Filter. This Scheme Is Block-Oriented Like [7-9, 20-21]. In This Technique, The Data File Is Divided Into Fixed Size Data Blocks. Let D1, D2, D3,…,Dn Be The Data Blocks And Bi Be The Data Block Number And Vi Be The Version Number For Data Block Di. A Set Of Hash Functions Used For Calculating The Bloom Filter Is Denoted By BF( ) And Hi Denotes The Set Of Hashes Corresponding To The Block Di And It Is Defined As-

$$Bi= BF(Hi(Di||Bi||Vi))$$

We Concatenate The Data Block Number And Version Number To The Block Of Data And Compute The Hashes And Store It Into The Bloom Filter Vector. In Proposed Scheme We Have Considered 8- Bit, 16-Bit, 24-Bit, And 32-Bit Bloom Filters Vector. The Resulting Bloom Filter Vector Hi Can Be Either Stored Locally On The Data Owner Computer Or Appended With The Data Block Itself On The Storage Server. The Data Owner (DO) Has Several Options In Storing The Data And The Corresponding Bloom Filter Vector Depending On Its Storage, Integrity, And Confidentiality Requirements. In This Technique, The Encrypted (ENC) And Unencrypted (UENC) Option Are Specified For Data And The Bloom Filter Vector. The BFV Location Option Is Specified For The Bloom Filter Vectors: At Server (SRV), At Data Owner (DO), Or At Both (SRV & DO).

In Options 1-4 Data Owner (DO) Keeps No Record At Client Side Node, But The Storage Server May Neglect The Recent Changes To A Data Block, Yet Maintaining Correct Bloom Filter Vector (BFV) From The Last Update. Neither The Data Owner (DO) Nor A Third Party Client May Detect This Inconsistency. The Options Are Listed In Table 1.

Table 1: Bloom Filter Vector Location

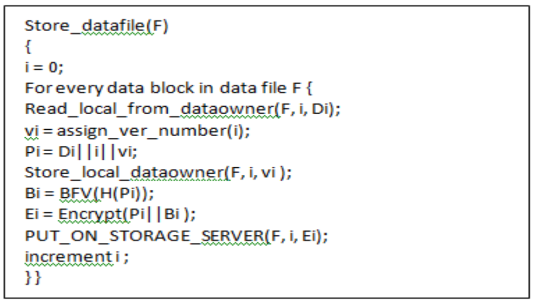
Scheme No	Data Block	Bloom Filter Vector (BFV)	Bloom Filter Vector Location BFV(Location)	Remarks
1	ENC	ENC	SRV	No Currency check
2	ENC	UENC	SRV	
3	UENC	ENC	SRV	
4	UENC	UENC	SRV	
5	ENC	ENC	DO	Data user to contact data owner for integrity/ currency check
6	ENC	UENC	DO	
7	UENC	ENC	DO	
8	UENC	UENC	DO	
9	UENC	ENC	SRV & DO	
10	ENC	UENC	SRV & DO	
11	ENC	ENC	SRV & DO	
12	UENC	UENC	SRV & DO	

In These Options No Storage Cost Is On The Data Owner (DO). In Serial No 1 And 3 Where The Bloom Filter Vector (BFV) Is Encrypted, It Is Possible To Rectify This By Data Owner (DO). This Will Enable It To Check The Currency When A Data Block Is Retrieved And The Bloom Filter Vector (BFV) Recomputed.

In Schemes 5-8, Data Owner Keeps The Bloom Filter Vector (BFV) And So The Additional Storage Cost At The Server Side Is Nothing And Whenever A Data Block Is Accessed, The Data Owner Computes The Hash And Test It For Data Integrity And Currency. However, Schemes 5-8 Have An Additional Storage Overhead On The Data Owner Machine And Any Clients Accessing The Data From The Storage Server Need To Contact The Data Owner For Integrity Test. It Will Bring An Additional Processing And Communication Cost For The Data Owner Machine.

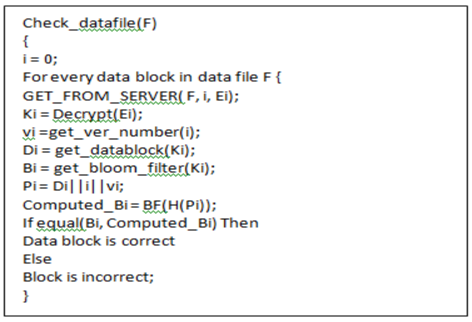
Finally, In Schemes 9-12, Both Storage Server And Data Owner Machine Keep The Bloom Filter Vector (BFV). This Is Helpful When Other Customer/Clients Access The Data Blocks And Hence Can Test The Data Integrity Directly Without Contacting The Data Owner Machine. The Data Owner Also Can Carry Out Currency Checks. While The Encrypted Bloom Filter (EBFV) At The Owner Is Not Useful, The Encrypted Bloom Filter At The Server Is Useful. Now The Storage Server Can Tampers The Data, But Can't Change The Bloom Filter Vector Since It Is Encrypted And The Client Will Be Able To Decrypt The Bloom Filter Vector And Can Check For Data Integrity Without Contacting The Data Owner. It Still Needs To

Contact The Data Owner For The Currency. In This Paper, For We Assume Option 11 With A Modification For Storage Efficiency. We Store The Version Number Of Data Block Rather Than BFV.



```
Store_datafile(F)
{
i = 0;
For every data block in data file F {
Read_local_from_dataowner(F, i, Di);
vi = assign_ver_number(i);
Pi = Di| |i| |vi;
Store_local_dataowner(F, i, vi );
Bi = BFV(H(Pi));
Ei = Encrypt(Pi| |Bi );
PUT_ON_STORAGE_SERVER(F, i, Ei);
increment i ;
}}
```

Figure 1a: Procedure at data owner to store a file at server



```
Check_datafile(F)
{
i = 0;
For every data block in data file F {
GET_FROM_SERVER( F, i, Ei);
Ki = Decrypt(Ei);
vi =get_ver_number(i);
Di = get_datablock(Ki);
Bi = get_bloom_filter(Ki);
Pi = Di| |i| |vi;
Computed_Bi = BF(H(Pi));
If equal(Bi, Computed_Bi) Then
Data block is correct
Else
Block is incorrect;
}
```

Figure 1b: Procedure at data owner to verify a file

Whenever A Data Block Is Modified In A Data File And Rewritten, That Version Number Is Incremented And The Corresponding Bloom Filter Vector (BFV) Is Recomputed For The Modified Block And Stored At The Server.

Fig. 1a And 1b Give A Detailed Information On The Data Storage And Data Retrieval/Verification At Data Owner. Since The Underlying Hash Functions Make A Significant Impact On The Effectiveness Of A Hash Filter, We Have Looked At Several Hashing Functions.

In This Experiment We Have Chosen The Basic Hash Functions Shown In Table 2 From [22-23]. SDBM Hashing Is The Simplest Of All And Its Code In C++ Is Shown In Fig. 2. Here, Data Is The Input String For Which Hash Is Being Computed And Size Is The String Size. The Resulting Hash Value Is An Unsigned Int.

Table 2: Basic Hash Functions Used In The Experiments

Hash Function	Source
RS	R. Sedgwick
SDBM	Open Source SDBM Project
JS	J. Sobel
PJW	P. J. Weinberger
BKDR	B. Kernighan And D. Ritchie

IV. RESULTS

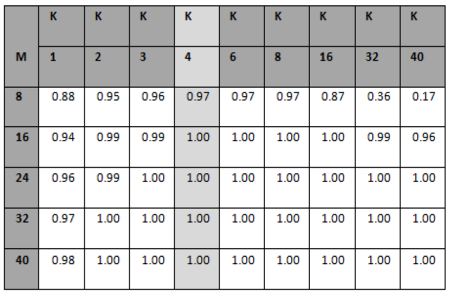
In This Experiment We Have Considered A Bloom Filter Vector Of Size M And K Independent Hash Functions, Each With Range {0,1,…,M-1}. For Each Element X (I.E A Data Block), The Bit Positions At H1(X), H2(X), …, Hk(X), Are Set To 1.

Given A Retrieved Data Block Y And Its Bloom Filter Vector We Have To Check If The Data Block Has Been Tampered With. If H1(Y), H2(Y), …, Hk(Y), Are All Set To 1, And No Other Bits Are Set To 1. In This We Case The Data Block Y Is Not Tampered With. However, There Is A Finite Probability Of False Positive (FP). Fan Et Al. [24] Shown That The Probability Of A False Positive (FP) Test Result Is-

$[1-(1-1/M) K]^{K}$

But The Important Assumption Underlying This Formula Is That The K Hash Functions Are Independent. This Is Empirically Shown In Table 3.

Table 3: Comparison With Variation Of Hash Fucntions



	K	K	K	K	K	K	K	K	K
M	1	2	3	4	6	8	16	32	40
8	0.88	0.95	0.96	0.97	0.97	0.97	0.87	0.36	0.17
16	0.94	0.99	0.99	1.00	1.00	1.00	1.00	0.99	0.96
24	0.96	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00
32	0.97	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
40	0.98	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

It Has Been Observed That (I) Accuracy Improves With Increase In Bloom Filter Vector (BFV) Size M; (Ii) When $M/K < 0.5$, Increase In K, For A Given M, May Actually Decrease The Bloom Filter Accuracy. (Iii) When $M/K \geq 0.5$, The Accuracy Improves With Increase In K For A Given M.

We Can Achieve Storage And Computational Efficiency By Using Four Hash Functions I.E. K=4 With Filter Size Of 8 Bits I.E. M=8 With An Accuracy Of 0.97. We Can Use A Filter Size Of 16 Bits I.E. M=16 With An Accuracy Of 1.

Here, Table 3 Shows Theoretical Accuracy Measures But The Underlying Hypothesis May Not Always Be Valid. So, To Check Validation And To Compare The Performance Of Bloom Filters Vector We Have Test Several Experiments. The Results And Conclusions From These Experiments Are Discussed Here.

A. Storage Requirements Of BFV

Bloom Filters Require Much Less Space Than Secure Hashes As:
SHA-1: It Generates A 20-Byte Digest
SHA-2: It Generates Either 32-Byte Or 64-Byte Digest
MD5: It Generates A 16- Byte Digest.

Bloom Filters Vectors That We Consider Use 1-Byte, 2-Byte, 3-Byte Or 4-Byte Filters Which Are Far Smaller Than The Secure Hashes.

B. Storage Overhead At The Server

Techniques That Use Merkle Hash Trees [7], The Storage Server Send All Hashes From The Requested Data Block Element's Root To The Root Of The Tree To The Client. For Example, For A 1 Mega Byte File With A Data Block Size Of 1K Bytes, The Height Of The Merkle Hash Tree Is About 10. This Is Ten Times More Than The Bloom Filter Vector. Here, The Storage Cost Is Number Of Data Blocks When Both Merkel Hash Trees And Bloom Filters Vector Are Used. It Is Proved That Bloom Filters Vectors Have A Major Advantage Over The Merkle Hash.

C. Effect Of The BFV Size And The Hash Functions

If A Storage Server Tampers With The Data, The BFV Of The Modified Data Should Not Match With The Original BFV When Verified By The DO Or Other Clients. If There Is A Match In BFV's, Then It Is A False Positive. So The Accuracy Of The BFV Can Be Measured By The Number Of False Positives Obtained From Various Modified Data Blocks. It Is Clear That Even With 8-Bit Bloom Filters And As Few As 3 Hash Functions, One Can Obtain 96% Accuracy. This Accuracy May Be Further Increased To Almost 100% By Increasing The Bloom Filter Size To 24 Bits And Increasing The Number Of Hashes To 12.

D. Computational Overhead

We Have Employed Multiple Hash Functions And The Number Of Hash Functions Used Clearly Dictated The Execution Times. Surprisingly, The Computational Overhead Is Not Linearly Proportional To The Number Of Hashes. The Execution Time Is Directly Proportional To The Number Of Blocks In The Data File.

V. CONCLUSION

In This Paper, We Have Investigated Efficiency Of Bloom Filters For Integrity Of Data Blocks. Here, We Discussed The Merkle's Hashing Methods For Data Integrity And Explained Their Additional Storage Cost On The DO's. Proposed Method Uses Bloom Filters Vectors And Reduces This Overhead. In Order To Reduce The Computational Overhead We Have Implemented These Schemes On Dual Core And Quad Core Systems. The Experiments Clearly Show The Advantage Of Using BFV.

REFERENCES

[1] A. S. Tanebaum And D. J. Wetherall, Computer Networks, 5th Edition, Pearson Higher Education, 2011.

[2] J. Peng, Y. Zhou, And Y. Yang, "Cyclic Redundancy Code Checking Based On Small Lookup Table," IEEE Intl. Conf. On Communication Technology And Applications, Pp. 596-599, 2010

[3] B. Gassend, G. E. Suh, D. Clarke, M. Van Dijk, And S. Devdas, "Caches And Merkle Trees For Efficient Memory Integrity Verification," 9th Intl. Symp. High Performnace Computer Architecture, Feb. 2003.

[4] A. Silberschatz, P. B. Galvin, And G. Gane, Operting System Concepts, Eith Edition, Wiley, 2009.

[5] R. Gagliardi, F. Marcantoni, A. Polzonetti, B. Re, And P. Tapanelli, "Cloud Computing For Network Business Ecosystem," IEEE IEEM'10, Pp. 862-868, Dec. 2010.

[6] J. B. Gurman, "How Many Terabytes Was That? Archiving And Serving Solar Space Data Without Losing Your Shirt," Bulletinn Of The American Astronomical Society, Vol. 31, P. 955, May 1999.

[7] J. Li, M. N. Krohn, D. Mazieres, And D. Shasha, "Secure Untrusted Data Repository (SUNDR), " In OSDI, 2004, Pp. 121-136.

[8] A. Yun, C. Shi, And Y. Kim, "On Protecting Integrity And Confidentiality Of Cryptographic File System For Outsourced Storage," CCSW 2009, Nov 2009.

[9] M. T.Goodrich, C Papamanthou, R. Tamassia, And N. Triandopoulos, "Athos: Efficient Authentication Of Outsourced File Systems," ISC 2008, LNCS 5222, Pp. 80–96, Springer-Verlag Berlin, 2008.

[10] B. H. Bloom, "Space/Time Trade-Offs In Hash Coding With Allowable Errors," CACM, Vol. 13, No. 7, July 1970, Pp. 422-426.

[11] F. Chang, J. Dean, S. Ghemawat, W. C. Hseih, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, And R. Gruber, "Bigtable: A Distributed Data Storage System For Structured Data," Proc. 7th Symp. Operating Systems Design And Implementation (OSDI'06), 2006, Pp. 205-218.

[12] A. Kumar, J. Xu, L. Li, And J. Wang, "Space-Code Bloom Filters For Efficient Traffic Flow Measurement," IEEE J. Selected Areas In Communication, Vol. 24, No. 12, 2006, Pp. 2327-2339.

[13] Z. Li And G. Gong, "On Data Aggregation With Secure Bloom Filter In Wireless Sensor Networks," Technical Report, Dept. Of Electrical And Computer Engineering, Univ. Waterloo, Canada.

[14] A. Telidevara, V. Chandrasekaran, A. Srinivasan, R. Mukkamala, And S. Gampa,

"Similarity Coefficient Generators For Network Forensics," Proc. IEEE WIFS, 2010.

[15] F. Jian-Ming, X. Ying, X. Hui-Jun, And W. Wei, "Startegy Optimization For P2P Security Using Bloom Filter," Intl. Conf. Multimedia Information Netyworking And Security, MINES'09, 2009, Pp. 403-406.

[16] P. Williams, R. Sion, And B. Carbunar, "Building Castles Out Of Mud: Practical Access Pattern Privacy And Correctness On Untrusted Storage," ACM CCS'08, Oct. 27-31, 2008, Alexandria, Virginia, Pp. 139-148.

[17] M. Zhang, K. Cai, And D. Feng, "Fine-Grained Cloud DB Damage Examination Based On Bloom Filters," Proc. ACM Web-Age Information Management (WIAM 2010), Springer-Verlag LNCS 6184, 2010, Pp. 157-168.

[18] M. J. Quinn, Parallel Programming In C With MP And Openmp, Mcgraw-Hill, 2004.

[19] A. Oprea, M. K. Reiter, And K. Yang, "Space-Efficint Block Storage Integrity," Proc. 12th Annual Network And Distributed System Security Symposium, NDSS'05, 2005.

[20] E. Mykletun, M. Narasimha, And G. Tsudik, "Authentication And Integrity In Outsourced Databases," ACM Transactions On Storage , Vol. 2, No. 2, May 2006. Pp. 107-138.

[21] S. Halevi And P. Rogaway, "A Tweakable Enciphering Mode," Advances In Cryptology CRYPTO'03, Lecture Notes In Computer Science, Vol. 2729, Pp. 482-499, Springerverlag, 2003.

[22] A. Kirsch And M. Mitzenmacher, "Less Hashing, Same Performance: Building A Better Bloom Filter," ESA 2006, Springer-Verlag LNCS 4168, Pp. 456-467, 2006.

[23] A. Partow, Genral Purpose Hash Function Algorithms, Http://Www.Partow.Net/Programming/Hashfunctions/.

[24] L. Fan, P. Cao, J. Almeida, And A. Z. Broder, "Summary Cache: A Scalable Wide-Area Cache Sharing Protocol," IEEE/ACM Trans. Networking, 2000, Pp. 254-265

[25] X. Wang And H. Yu, "How To Break MD5 And Other Hash Functions," EUROCRYPT 2005, Pp. 19-35, 2005.

Pramod Kumar "Efficient Data Integrity Algorithm For Outsourced Data In Cloud Environment "International Journal of Engineering Research and Applications (IJERA) , vol. 8, no. 03, 2018, pp. 10-15