International Journal of Emerging Science and Engineering (IJESE)
ISSN: 2319–6378, Volume-2 Issue-11, September 2014

File Security using Homomorphic Hashing in Peer to Peer Networks

Nirmit Desai, Chitesh Tewani, Karl Elavia, Omkar Gawde, Kiran Joshi

Abstract- This paper focuses primarily with homomorphic hashing and the quality of peer-to-peer content distribution. Some systems using simple block-by-block downloading can verify blocks with traditional cryptographic signatures and hashes, but these techniques do not apply well to more elegant systems that use rate less erasure codes for efficient multicast transfers. This paper presents a practical scheme, based on homomorphic hashing, that enables a downloader to perform on-the-fly verification of erasure-encoded blocks. Peer-to-peer content distribution networks can suffer from malicious participants that intentionally corrupt content. Traditional systems verify blocks with traditional cryptographic signatures and hashes. However, these techniques do not apply well to more elegant schemes that use network coding techniques for efficient content distribution. Problems that occur with these techniques are that peers have no way of knowing which block was bad if a piece they download fails hash check, and if they're streaming data they can't display it until a full piece is downloaded for hash verification purposes. Also there is a huge waste of bandwidth when a piece does not pass hash check, in fact, the peer must discard all the blocks (even all the correct ones) and then re-download all the blocks within the piece. It is better to discard only bad blocks, and re-download only them which will save bandwidth. Identifying such bogus blocks is difficult and requires the use of homomorphic hashing functions. This paper deals with reducing the bogus blocks by implementing homomorphic hashes on the blocks and using Luby Transform Codes on peer to peer networks.

Keywords- Homomorphic Hashing, Peer-to-peer (P2P), Luby Transform Codes (LT codes), Erasure Codes, File security

I. INTRODUCTION

A Peer-to-Peer network is a distributed application architecture that subdivides the workload among all participating peers. The peers, also called nodes of the overlay network, are equally privileged, and each one shares a fraction of its resources with others, without a central coordination. This type of architecture is suitable for much kind of applications: file sharing, video-on-demand [5], live-streaming and distributed computing are few examples. Nowadays peer-to-peer (P2P) applications generate a substantial fraction of the total Internet traffic. The reasons for this widespread diffusion are not difficult to understand. From the perspective of the end user, P2P applications make possible to retrieve a large amount of multimedia contents, and in turn to share media with other people.

Manuscript Received on September 22, 2014.

Nirmit Desai, B.Tech Student, Computer Science & Information Technology Department, VJTI, Mumbai, India.

Chitesh Tewani, B.Tech Student, Computer Science & Information Technology Department, VJTI, Mumbai, India.

Karl Elavia, B.Tech Student, Computer Science & Information Technology Department, VJTI, Mumbai, India.

Omkar Gawde, B.Tech Student, Computer Science & Information Technology Department, VJTI, Mumbai, India.

Kiran Joshi, Assistant Professor, Computer Science & Information Technology Department, VJTI, Mumbai, India.

Most important, the benefit of using a P2P overlay with respect to the classical client/server paradigm is to shift the load from a central server to peers, with many advantages: saving server bandwidth, providing the service to more users with the same resources utilization, obtaining a more robust system due the error-resilient features of a distributed overlay. On the other side, a P2P overlay exploits the Internet infrastructure (core and access networks) and generates a huge amount of traffic. This represents a problem especially in situations with untrusted users, such as peer to peer networks. There doesn't seem to be a practical way to verify if a peer is sending you valid blocks until you decode the file. Using Fountain Codes [4], a clever probabilistic algorithm that allows us to break a large file up into a virtually infinite number of small chunks, such that you can collect any subset of those chunks - as long as you collect a few more than the volume of the original file - and be able to reconstruct the original file. We propose a method using Homomorphic Hashing [3] a construction that's simple in principle and one that you can compute the hash of a composite block from the hashes of the individual blocks. With a construction like this, we could distribute a list of individual hashes to users, and they could use those to verify incoming blocks as they arrive.

II. THEORETICAL ANALYSIS

1. Homomorphic Hash

Homomorphic Hashing [3] is a novel construction that lets recipients verify the integrity of check blocks immediately, before consuming large amounts of bandwidth or polluting their download caches. A file F is compressed down to a smaller hash value, H(F), with which the receiver can verify the integrity of any possible check block. Receivers then need only obtain a file's hash value to avoid being duped during a transfer. The function H is based on a discrete-log based, collision-resistant, homomorphic hash function, which allows receivers to compose hash values in much the same way that encoders compose message blocks. Unlike more obvious constructions, hash function is independent of encoding rate and is therefore compatible with rateless erasure codes [10]. It is fast to compute, efficiently verified using probabilistic batch verification, and has provable security under the discrete-log assumption. The implementation results suggest scheme is practical for real-world use. Homomorphic hashes are reasonably-sized and independent of the encoding rate and they enable receivers to authenticate check blocks on the fly. The proposed two possible authentication protocols based on a homomorphic collision-resistant hash function (CRHF) are:

a. Global hashing model: In the global hashing model, there is a single way to map F to H(F) by using global parameters. As such, one-time hash generation is slow but well-defined.

b. Per-publisher hashing model: In the per-publisher hashing model, each publisher chooses his own hash parameters, and different publishers will generate different hashes for the same file.

2. Luby Transform Codes

The quality of peer-to-peer content distribution can suffer when malicious participants intentionally corrupt content. Some systems using simple block-by-block downloading can verify blocks with traditional cryptographic signatures and hashes, but these techniques do not apply well to more elegant systems that use rateless erasure codes for efficient multicast transfers. Peer-to-peer content distribution networks (P2P-CDNs) are trafficking larger and larger files, but end-users have not witnessed meaningful increases in their available bandwidth, nor have individual nodes become more reliable. As a result, the transfer times of files in these networks often exceed the average uptime of source nodes, and receivers frequently experience download truncations. Multicast transmission of popular files might drastically reduce the total bandwidth consumed; however, traditional multicast systems would fare poorly in such unstable networks. Developments in practical erasure codes [12] and rateless erasure codes [13], [14] point to elegant solutions for both problems. Erasure codes of rate r (where $0 < r < 1$) map a file of n message blocks onto a larger set of n=r check blocks. Using such a scheme, a sender simply transmits a random sequence of these check blocks. A receiver can decode the original file with high probability once he has amassed a random collection of slightly more than n unique check blocks. In rateless codes, block duplication is much less of a problem: encoders need not pre-specify a value for r and can instead map a file's blocks to a set of check blocks whose size is exponential in n. A file F is compressed down to a smaller hash value H(F), with which the receiver can verify the integrity of any possible check block. Receivers then need only obtain a file's hash value to avoid being duped during a transfer. The function H is based on a discrete-log-based collision-resistant, homomorphic hash function, which allows receivers to compose hash values in much the same way that encoders compose message blocks. It is independent of encoding rate and is therefore compatible with rateless erasure codes. It is fast to compute, efficiently verified using probabilistic batch verification, and has provable security under the discrete-log assumption. Furthermore, our implementation results suggest this scheme is practical for real-world use. Typically, a file F is divided into n uniformly sized blocks, known as message blocks (or alternatively, input symbols). Erasure encoding schemes add redundancy to the original n message blocks, so that receivers can recover from packet drops without explicit packet retransmissions.

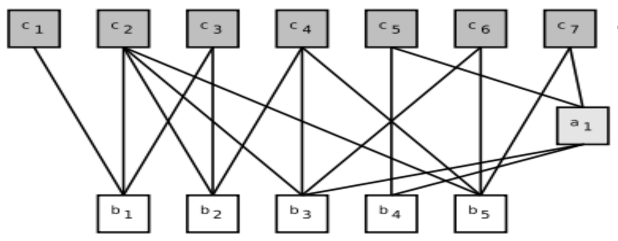


Fig. 1: Online Encoding of a Five-Block File, b_i are Message Blocks, a1 is an Auxiliary Block, and c_i are Heck Blocks

Edges represent addition (via XOR).

For example, $c_4 = b_2 + b_3 + b_5$,
$a_1 = b_3 + b_4$, and $c_7 = a_1 + b_5$.

Researchers have proposed a class of erasure codes with sub-quadratic decoding times such as Tornado Codes [17], LT Codes [13], RaptorCodes [16] and Online Codes [14].

These schemes output check blocks (or alternatively, output symbols) that are simple summations of message blocks. That is, if the file F is composed of message blocks b_1 through b_n, the check block c1 might be computed as b_1+b_2. The specifics of these linear relationships vary with the scheme. For multicast and other applications that benefit from lower encoding rates, LT, codes are preferable [18]. They feature rateless encoders that can generate an enormous sequence of check blocks with state constant in n.LT codes are decodable in time O(n ln(n)).

3. Peer-to-Peer (P2P)

Peer-to-Peer (P2P) networks have recently emerged as alternative to traditional Content Distribution solutions to deliver large files. Such P2P networks [7] [8] [9] create a fully distributed architecture where commodity PCs are used to form a cooperative network and share their resources (storage, CPU, bandwidth). By capitalizing the bandwidth of end systems, P2P cooperative architectures offer great potential for providing a cost-effective distribution of software updates, critical patches, videos, and other large files to thousands of simultaneous users both Internet-wide and in private networks.

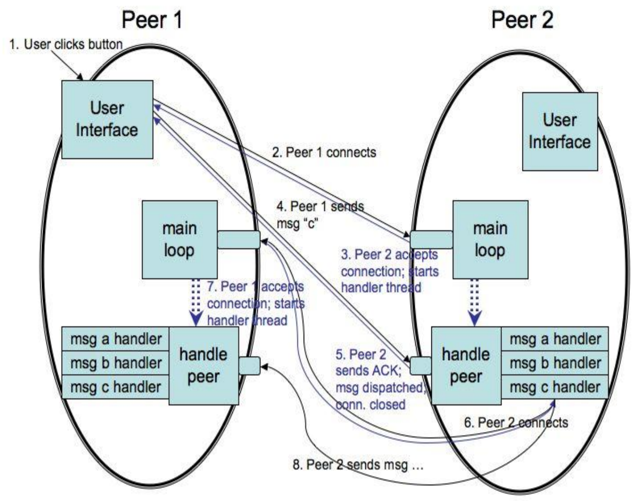


Fig. 2: Peer-to-Peer Model

.

Step 1: In figure 2, a scenario is diagrammed where the user on Peer 1 clicks a button, for example, a "Search" button, in the GUI interface. The interface somehow decides to send a "Query" message to another peer, in this case, Peer 2.
Step 2: The main loop of Peer 2 detects the incoming connection request.
Step 3: Peer 2 starts up a separate thread to handle the actual data of the request (A thread is a task that a program runs simultaneously, or pseudo-simultaneously, with other running tasks. The purpose of using threads here is to allow a peer to handle multiple incoming connections simultaneous.)
Step 4: Assuming, message type "c" refers to a "Query" message, Peer 1 sends the actual message once it has gotten a connection to Peer 2.
Step 5: The "handle peer" task (thread) of Peer 2 receives the message, sends an acknowledgment back to Peer 1, closes the connection, and then calls an appropriate function/method to handle the message based on its type.
Step 6: After processing the message, the "msg c handler" function decides that it needs to send a "Query Response" message back to Peer 1, so it attempts to connect.
Step 7: Peer 1's main loop, listening for such connections, accepts the connection and starts its separate handler thread to receive the actual message data from Peer 2 (step 8). Having received the message, Peer 1 does what Peer 2 did in step 5, and the process continues.

III. IMPLEMENTATION METHODOLOGY

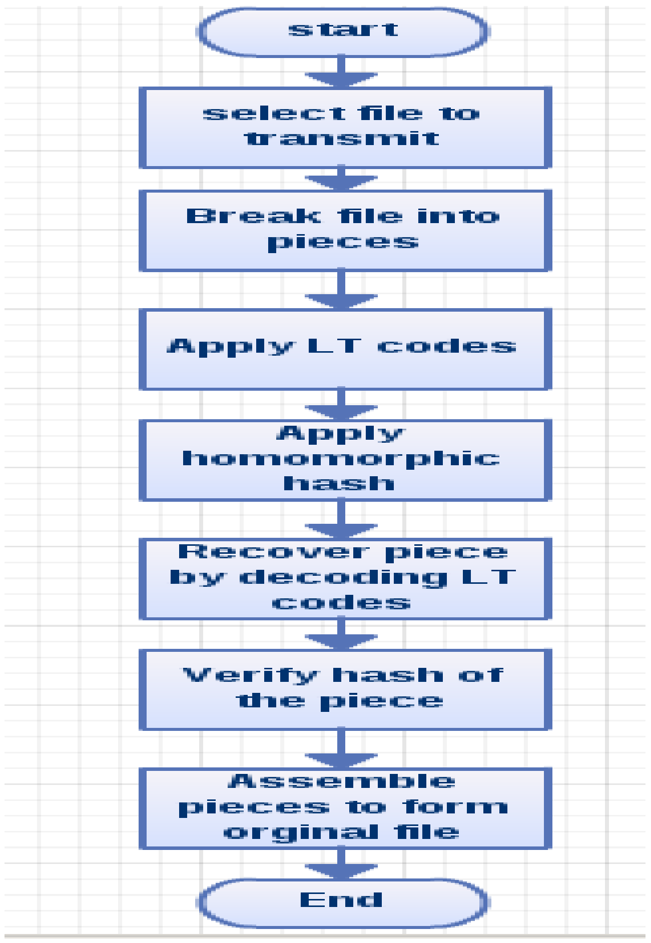


Fig. 3: Implementation Flowchart

Our implementation of secure file transfer using homomorphic hashing in peer-to-peer follows the approach outlined in Fig 3

A. LT Encoding

The encoding process begins by dividing the un-coded message into *n* blocks of roughly equal length. Encoded packets are then produced with the help of a pseudorandom number generator.

1. The degree *d*, $1 \le d \le n$, of the next packet is chosen at random.
2. Exactly *d* blocks from the message are randomly chosen.
3. If M_i is the i^{th} block of the message, the data portion of the next packet is computed as

$$(M_{i_1} \oplus M_2 \dots \oplus M_{i_d}) \qquad (1)$$

where $\{i_1, i_2, \dots, i_d\}$ are the randomly chosen indices for the *d* blocks included in this packet for equation (1).

4. A prefix is appended to the encoded packet, defining how many blocks *n* are in the message, how many blocks *d* have been exclusive-ored into the data portion of this packet, and the list of indices $\{i_1, i_2, \dots, i_d\}$.
5. Finally, some form of error-detecting code (perhaps as simple as a cyclic redundancy check) is applied to the packet, and the packet is transmitted. This process continues until the receiver signals that the message has been received and successfully decoded.

B. LT Decoding

The decoding process uses the "exclusive or" operation to retrieve the encoded message.

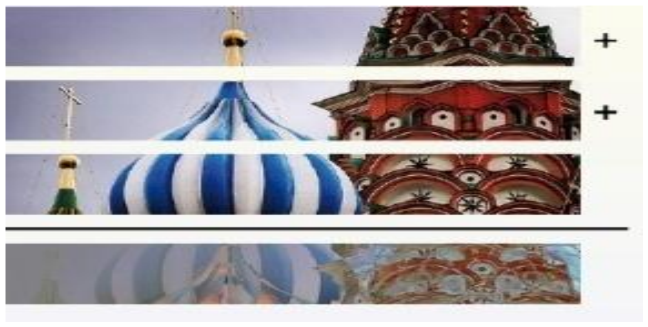
1. If the current packet isn't clean, or if it replicates a packet that has already been processed, the current packet is discarded.
2. If the current cleanly received packet is of degree $d > 1$, it is first processed against all the fully decoded blocks in the message queuing area (as described more fully in the next step), then stored in a buffer area if its reduced degree is greater than 1.
3. When a new, clean packet of degree $d = 1$ (block M_i) is received (or the degree of the current packet is reduced to 1 by the preceding step), it is moved to the message queuing area, and then matched against all the packets of degree $d > 1$ residing in the buffer. It is exclusive-ored into the data portion of any buffered packet that was encoded using M_i, the degree of that matching packet is decremented, and the list of indices for that packet is adjusted to reflect the application of M_i.
4. When this process unlocks a block of degree $d = 2$ in the buffer, that block is reduced to degree 1 and is in its turn moved to the message queuing area, and then processed against the packets remaining in the buffer.
5. When all *n* blocks of the message have been moved to the message queuing area, the receiver signals the transmitter that the message has been successfully decoded.

This decoding procedure works because $A \oplus A = 0$ for any bit string *A*. After $d - 1$ distinct blocks have been exclusive-ored into a packet of degree *d*, the original un-encoded content of the unmatched block is all that remains, as proved in equation 2. In symbols we have

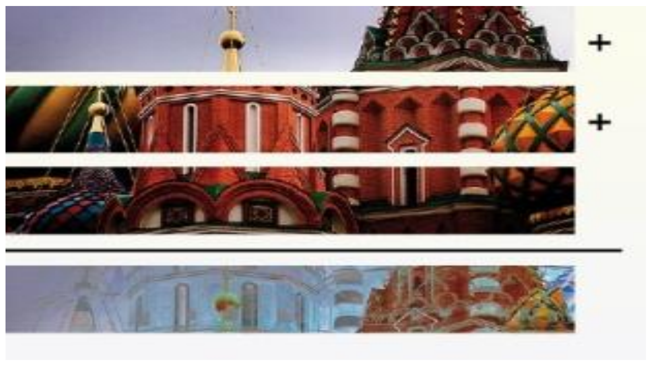
$$(M_{i_1}\oplus \ldots \oplus M_{i_d})\oplus(M_{i_1}\oplus \ldots \oplus M_{i_{k-1}} \oplus M_{i_{k+1}}\oplus \ldots \oplus M_{i_d})$$
$$= M_{i_1}\oplus M_{i_1}\oplus .. \oplus M_{i_{k-1}}\oplus M_{i_{k-1}}\oplus M_{i_k}\oplus M_{i_{k+1}}\oplus M_{i_{k+1}} .. M_{i_d}\oplus M_{i_d}$$
$$= 0\oplus .. \oplus 0 \oplus M_{i_k}\oplus 0 .. \oplus 0$$
$$= M_{i_k} \tag{2}$$

A

B



C



D

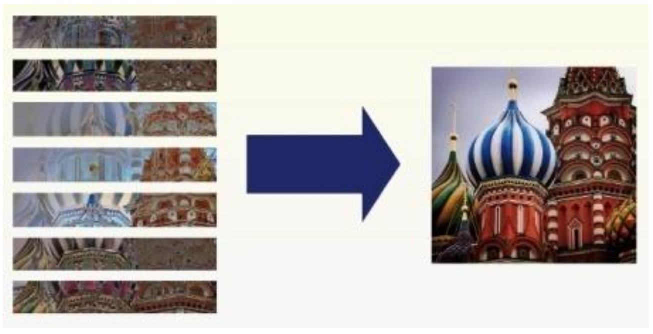

E

Fig. 4: Example of LT Encoding and Decoding

An example of LT encoding and decoding would work on an image as shown in Fig. 4. The steps are as follows:
Step1: Data (A)
Step2: Split into k = 6 parts (B)
Step3: One drop is generated by XORing parts 1, 2 and 3 (C)
Step4: Another drop is generated by XORing part 1, 5 and 6(D)
Step5: Drop generation can be repeated over and over again, until necessary
Step6: Decode (E)

C. Homomorphic Hashing

The two possible authentication protocols based on a homomorphic collision-resistant hash function (CRHF) are:

1. Global Homomorphic Hashing

In global homomorphic hashing [3], all nodes on the network must agree on hash parameters so that any two nodes independently hashing the same file F should arrive at exactly the same hash. To achieve this goal, all nodes must agree on security parameters λp and λq. Then, a trusted party globally generates a set of hash parameters G = (p, q, g), where p and q are two large random primes such that $|p| = \lambda_p$, $|q| = \lambda_q$ and $q \mid (p-1)$. The hash parameter g is a $1 \times m$ row-vector, composed of random elements of Z_p, all order q.

In particular, no node should know i, j, x_i, x_j such that

$$g_j^{x_i} = g_j^{x_j} \tag{3}$$

as one that had this knowledge could easily compute hash collisions. The generators might therefore be generated according to the algorithm PickGroup given in Figure 5. The input (λp , λq , m, s) to the PickGroup algorithm serves as a heuristic proof of authenticity for the output parameters, G = (p, q, g).

```
Algorithm PickGroup($\lambda_p$, $\lambda_q$ , m, s)
          Seed PRNG G with s.
          do
             $q \leftarrow qGen(\lambda_q)$
             $p \leftarrow pGen(q, \lambda_p)$
          while $p = 0$ done
          for $i = 1$ to m do
               do
            $x \leftarrow G(p-1)+1$
            $g_i \leftarrow x \pmod p$
          while $g_i = 1$ done
        done
      return (p, q, g)

Algorithm qGen($\lambda_q$ )
      do
             $q \leftarrow G(2\lambda_q)$
      while q is not prime done
      return q

Algorithm pGen(q, $\lambda_p$ )
      for $i = 1$ to $4\lambda_p$ do
          $X \leftarrow G(2\lambda_p)$
          $c \leftarrow X \pmod{2q}$
          $p \leftarrow X - c + 1$      // Note $p \equiv 1 \pmod{2q}$
      if p is prime then return p
      done
      return 0
```

Fig. 5: Pick Group Algorithm

1. File Representation

From [3], let β be the block size and let $m = [\beta/\lambda_q - 1$.Consider a file F as an m × n matrix, whose cells are all elements of Z_q. Selection of m guarantees that each element is less than $2^{\lambda_q - 1}$, and therefore less than the prime q. Now, the j^{th} column of F simply corresponds to the j^{th} message block of the file F, which we write $b_j = (b_{1,j} b_{2,j} b_{3,j} .. b_{m,j})$. Thus:

$$F = (b_1 b_2 .. b_n) = \begin{pmatrix} b_{1,1} & \cdots & b_{1,n} \\ \vdots & \ddots & \vdots \\ b_{m,1} & \cdots & b_{m,n} \end{pmatrix} \qquad (4)$$

We add two blocks by adding their corresponding column-vectors. That is, to combine the i^{th} and j^{th} blocks of the file, we simply compute:

$$b_i + b_j = (b_{1,i} + b_{1,j} \ldots b_{m,i} + b_{m,j}) \; mod\ q \qquad (5)$$

2. Precoding

The precoding stage produces auxiliary blocks that are summations of message blocks, and that the resulting composite file has the original n message blocks, and the additional n δ k auxiliary blocks as given in [3]. The precoder now proceeds as usual, but uses addition over Zq instead of the XOR operator.

3. Encoding

Like precoding, encoding is unchanged save for the addition operation. For each check block, the encoder picks an n'-dimensional bit vector x and compute $c = F'$. The output $\langle x, c \rangle$ describes the check block.

4. Hash Generation

To hash a file, a publisher uses a CRHF, secure under the discrete-log assumption. This hash function is a generalized form of the Pederson commitment scheme [19] (and from Chaum et al. [20]), and it is similar to that used in various incremental hashing schemes. Recall that a CRHF is informally defined as a function for which finding any two inputs that yield the same output is difficult.

For an arbitrary message block b_j, define its hash with respect to G from [3] [4]:

$$h_G(b_j) = \prod g_i^{b_{i,j}} \; mod\ p \qquad (6)$$

Define the hash of file F as a 1X n row-vector whose elements are the hashes of its constituent blocks using equation 6:

$$H_G(F) = (H_G(b_1)\, H_G(b_2) \ldots H_G(b_n)) \qquad (7)$$

5. Hash Verification

If a downloader knows (G, H_G (F)), he can first compute the hash values for the (n δ k) auxiliary blocks.

The precoding matrix Y is a deterministic function of the file size n and the pre-established encoding parameters δ and k. Thus, the receiver computes Y and obtains the hash over the composite file as:

$$H_G(F') = H_G\,(F).Y \qquad (8)$$

The hash of the auxiliary blocks is the last (n δ k) cells in this row vector. To verify whether a given check block $\langle x, c \rangle$satisfies

$c = F'x$, a receiver verifies that:

$$h_G(c) = \prod_{i=1}^{n} h_G(b_i)^{x_i} \qquad (9)$$

h_G functions here as a *homomorphic hash function*.

For any two blocks b_i and b_j,

$$h_G(b_i + b_j) = h_G(b_i).h_G(b_j) \qquad (10)$$

6. Decoding

XOR is conveniently its own inverse, so implementations of standard Online Codes need not distinguish between addition and subtraction. In our case, we simply use subtraction over q to reduce check blocks as necessary.

B. Per-Publisher Homomorphic Hashing

The per-publisher hashing scheme is an optimization of the global hashing scheme just described. In the per-publisher hashing scheme, a given publisher picks group parameters G so that a logarithmic relation among the generators g is known. The publisher picks q and p as in global hashing scheme, but generates g by picking a random $g \in Z_p$ of order q, generating a random vector r whose elements are in Z_q and then computing g = g^r.

$$H_G(F) = g^{rF} \qquad (11)$$

Given the parameters g and r, the publisher can compute file hashes with many fewer modular exponentiations. The publisher computes the product rF first using equation 11, and then performs only one modular exponentiation per file block to obtain the full file hash.

IV. CONCLUSION

Current peer-to-peer content distribution networks, such as the widely popular file-sharing systems, suffer from unverified downloads. A participant may download an entire file, increasingly in the hundreds of megabytes, before determining that the file is corrupted or mislabeled. Current downloading techniques can use simple cryptographic primitives such as signatures and hash trees to authenticate data. However, these approaches are not efficient for low encoding rates, and are not possible for rateless codes.

REFERENCES

1. P2P Streaming with LT Codes: a Prototype Experimentation. Andrea Magnetto, Rossano Gaeta, Marco Grangetto, Matteo Sereno
2. Cooperative Security for Network Coding File Distribution Christos Gkantsidis and Pablo Rodriguez Rodriguez
3. On-the-Fly Verification of Rateless Erasure Codes for Efficient Content Distribution"-
4. Capacity approaching codes design & Implementation " - D.J.C Mackay
5. An Approach for System Scalability For Video on Demand" By- V.B. Nikam , Kiran Joshi , B.B. Meshram.
6. Analyzing and Improving BitTorrent Performance" By- Ashwin R. Bharambe ,Cormac Herley ,Venkata N. Padmanabhan
7. An Analytic Framework for Modeling Peer to Peer Networks Krishna K. Ramachandran and Biplab Sikdar Rensselaer Polytechnic Institute, Troy NY 12180
8. Peer-to-Peer Research at Stanford Mayank Bawa, Brian F. Cooper, Arturo Crespo, Neil Daswani, Prasanna Ganesan, Hector Garcia-Molina, Sepandar Kamvar, Sergio Marti, Mario Schlosser, Qi Sun, Patrick Vinograd, Beverly Yang
9. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma and Steven Lim

10. L. Rizzo, "Effective erasure codes for reliable computer communication protocols," ACM Computer Communication Review, vol. 27, no. 2, Apr.1997.
11. S. Saroui, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, "An analysis of Internet content delivery systems," in *Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, Oct. 2002.
12. M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," in *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC)*, El Paso, TX, May 1997.
13. M. Luby, "LT codes," in *Proc. 43rd Annual Symposium on Foundationsof Computer Science (FOCS)*, Vancouver, Canada, Nov. 2002.
14. P. Maymounkov, "LT codes," NYU, Tech. Rep. 2002-833, Nov. 2002.
15. L. Rizzo, "Effective erasure codes for reliable computer communication protocols," *ACM Computer Communication Review*, vol. 27, no. 2, Apr.1997.
16. A. Shokrollahi, "Raptor codes," Digital Fountain, Inc., Tech. Rep. DF2003-06-001, June 2003.
17. J. Byers, M. Luby, and M. Mitzenmacher, "Accessing multiple mirror sites in parallel: Using Tornado codes to speed up downloads," in *Proc.IEEE INFOCOM '99*, New York, NY, Mar. 1999
18. P. Maymounkov and D. Mazi`eres, "Rateless codes and big downloads," in *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, Feb. 2003.
19. T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Advances in Cryptology—CRYPTO '91*, Santa Barbara, CA, Aug. 1991.
20. D. Chaum, E. van Heijst, and B. Pfitzmann, "Cryptographically strong undeniable signatures, unconditionally secure for the signer," in *Advancesin Cryptology—CRYPTO '91*, Santa Barbara, CA, Aug. 1991.

AUTHORS PROFILE

Nirmit Desai, Final Year B.Tech Information Technology from Veermata Jijabai Technological Institute, Mumbai, India.

Chitesh Tewani, Final Year B.Tech Information Technology from Veermata Jijabai Technological Institute, Mumbai, India.

Karl Elavia, Final Year B.Tech Information Technology from Veermata Jijabai Technological Institute, Mumbai, India.

Omkar Gawde, Final Year B.Tech Information Technology from Veermata Jijabai Technological Institute, Mumbai, India.

Kiran Joshi, Asst. Professor Veermata Jijabai Technological Institute, Mumbai, India.