A Distributed Private Information Storage Protocol Built on CPIR and Reed Solomon Codes

Anurag Kashyap Sarisht Wadhwa Grayson York

April 2021

Abstract

In this paper we describe a system for sharing files anonymously. Our system is a peer to peer network in which n storage bounded peers can share K files of size s using space only $O(MsK/n)$ for some security parameter M. These peers can then access the files anonymously in each of a set of communication rounds. Depending on the frequency of messages each file request will require a blowup of between $O(n)$ and $O(n^2)$ to the number of messages that must be sent between parties, but a blowup of only between $O(1)$ and $O(n)$ in the amount of data that must be sent across the network. This protocol can be secured against an unreliable network by storing the files using Reed-Solomon codes to stripe the data across a large subset (or all) the nodes and allow for some number of the nodes storing the file to drop without damaging the integrity of the file.

1 Previous Work

To the best of our knowledge, no protocol which accomplishes private information retrieval on a distributed network currently exists. Protocols such as Popcorn exist for private information retrieval from a central server, but these protocols assume that a large central or set of central servers exist which are able to store all of the data and disseminate it efficiently. Our protocol does not make this assumption, so it would be more efficient in situations where users were sharing a large amount of content between their personal machines. In addition, Popcorn provides no guarantee of hiding the identity of the user accessing the file. While an adversary may not know which file a user is accessing, the adversary will be able to determine which user is accessing the database and when.

Distributed file storage systems such as BitTorrent exist as well, but these systems provide little to no guarantee of privacy, and also cannot provide any space savings. Our protocol can provide a similar guarantee of reduced network bandwidth usage, but does not require replication of the entire file onto each node, and provides a privacy guarantee which BitTorrent cannot provide.

Tor provides anonymity to users in a distributed network, but this anonymity falls apart in the presence of a whole network adversary. In addition, Tor provides no possibility of reduced bandwidth usage or file storage overhead for nodes in the network, and does nothing to hide which files are being accessed on a given server.

2 Problem and Security Definition

We would like to create a peer to peer system where n parties can share K files between them, and no party must store a large fraction of the files. We want any given party i to be able to access any file j, without revealing to any other party that i is accessing j. We will define security using a trusted third party to present an idealized situation.

This system would contrast to previous work such as the Popcorn protocol, which requires a large central server to store the data. Our protocol assumes that no such server exists, and all storage must be on machines with bounded space.

Let us say we have a trusted third party T which stores each of the K files. If a party i wishes to access file j, i can query T and not reveal j. In addition, we will assume that each of the n parties queries T in a given communication round, potentially with a dummy request, and T will respond to each party with either the requested file or a dummy file. This way, any adversary snooping on the network will be unable to determine which nodes issued a request in a given communication round and which files were requested. We would like to create a system which is indistinguishable from this idealized trusted third party situation to an adversary which is watching traffic on node i, or more strongly on the entire network.

3 Background

3.1 Reed Solomon Codes

In this protocol we use Reed Solomon Codes, which are defined on a finite Galois Field $\mathbb{F} = GF(2^a)$. We can use an (n, b) RS code over $\mathbb{F}$ to encode messages of length b into code words of length $n < 2^a$ such that any b symbols of a given codeword can be used to recover the original code word. An (n, b) RS code is equipped with two functions, $enc(m_1) = s_1$ and $dec(s_2) = m_2$ where m_1, m_2 are strings of length b, s_1 is a codeword of length n, and s_2 is a codeword of length n with at most $n - b$ errors. Therefore, if we break the codeword up among n parties, so long as at least b of those parties remain honest, they can always recover the original data.

3.2 CPIR

A PIR protocol allows a user u to access a file at a specified index b on another machine m without m learning b. The trivial PIR protocol involves m sending all of its files to u, and u only taking the file at b. In fact, this is the only information theoretically secure PIR protocol which does not require multiple non-colluding servers.

In this paper, we use a PIR scheme which is only effective against computationally bounded adversaries known as CPIR. The CPIR scheme which we will describe is based on an additively homomorphic encryption scheme (gen,enc,dec) due to Stern.

First, the user will generate a public key private key pair (p_k, s_k). Then, the user will compute $c_i = enc(p_k, f_i)$, where $f_i = 1$ iff. $i = b$ and $f_i = 0$ otherwise. Then the user will send $(p_k, f_1, \ldots, f_n)$ to m.

m will have a library of data L, and will represent this library as a matrix of y bit integers. We will assume that each element of the library has size l. m will calculate $r_j =_{i=1}^{n} c_i^{L_{i,j}}$ for each $j < l/y$. Then, m will return $(r_1, \ldots, r_{l/y})$ to u.

Finally, u must simply calculate $dec(r_j, s_k)$ for each j, and u will have recovered the original message.

4 Protocol Specification

4.1 Assumptions

- The content of files is not a secret, anyone in the network can download and view the file.
- Fully Connected network; any node can communicate with any other node in the network.
- All files have the same size s for some integer $s \geq n$ representing the number of bits in the file. We will assume that $s \approx Sn$ for some integer $S \geq 1$.
- We will assume that less than half of the nodes in the network are corrupted by the adversary.

4.2 Notation

- n is the number of peers participating in the network

- M is a parameter which can be adjusted to tolerate more adversaries at the cost of efficiency
- s is the size of files in the network
- K is the total number of files stored by the peers
- F will denote an arbitrary file stored on our network.
- $S = s/n$ we will assume is an integer representing the ratio of the file size to the number of peers.
- U is an arbitrary uploader
- D is an arbitrary downloader
- c is the number of rounds of communication we perform during a given CPIR request.

4.3 Protocol Overview

Let us say that we have n parties which wish to store K files. We will define a protocol which allows each party to access any of the files anonymously without alerting an adversary snooping the network that they are accessing a file or which file is being accessed.

4.4 Upload Protocol

Lets say a party A, wants to upload a file F. A will encode the file in a $(Mn, s) = (Mn, Sn)$ Reed-Solomon code for some parameter M representing the level of redundancy we desire We see that with no RS code integrity, each peer would need to store s/n bits of the original file. When we encode the file, it will have a blowup of $\frac{Mn}{Sn} = M/S$ bits. Therefore, we see that A will need to send each of the n peers in the network $\frac{M}{Sn} = \frac{M}{s}$ bits of the code. All the peers save this code on their machine at some index i which we ensure is consistent across all of the machines. We see that each file upload will take only n messages sent across the network, and each message will be size $O(\frac{M}{s})$.

Each peer who receives the coded file with a unique identifier(file name or hash, which is the same across all the peers) in one particular communication round, would put the file into the next available index on its database. If multiple files have been uploaded in the same communication round, then we store the file with smaller hash (of unique identifier), at the smaller index. This ensures that index of each file is the same across all the honest peers in the network, and that each peer knows the index of each file represented by its unique identifier. We see that we can tolerate up to $(Mn - Sn)/2$ bits begin controlled by an adversary, and still reconstruct the file. Therefore, as there are Mn total bits in the network, we can tolerate $\frac{Mn-Sn}{2Mn} = \frac{M-S}{2M}$ fraction of the nodes being adversarial. Therefore we have a tradeoff where a higher M leads to more tolerance against adversarial participants in the network, but also a higher storage overhead.

4.5 Download Protocol

We will assume that when a downloader D requests a file from uploader U, that each of the n parties sends a CPIR request to each other party. In order to ensure that each party sends these requests simultaneously, we will assume that requests are sent in communication rounds, each lasting for some fixed time t. The parties which do not request a file in a given round will send a dummy request, and receive some arbitrary data (of the same size as a code) in return. That way, an adversary watching the network will not know which party D initiated the request.

D will send out a CPIR request for index j to each of the n peers participating in the protocol. Assuming that the adversaries are computationally bounded, an adversarial node observing the request would be unable to determine which index was desired. Then, upon receiving the RS codes from each of those peers, D will reconstruct the desired file from those codes. For any RS code received which does not match its corresponding digital signature the code is discarded.

4.6 Security

4.6.1 Upload

During the upload of the file, there is no privacy guarantee required, and thus we consider the following adversarial actions:

- Uploader is adversarial and purposefully gives out different unique identifiers throughout the network for the file. This would lead to a denial of service attack for not only the file uploaded by the uploader but all the files uploaded in the same communication round.
  - To prevent this kind of attack, each peer would declare each unique id received with the digital signature of the sender. If more than $n/2$ proofs are received for a given unique id, the same would be assigned an index, irrespective of whether the peer had originally received the message from the uploader or not. In additionally these ($> n/2$) votes are broadcast in the next round of communication, thus confirming the index with each of the honest peer. Assuming synchronous setting, every honest peer would have the same indexes of files.
  - Additionally if 2 different unique ids have been sent by the same uploader, its file can be immediately dismissed.
  - **Correctness**: For all honest uploaders the code should be accessible for each peer at the same index. This would be true since for each unique id in the system we maintain the same index throughout the network as discussed above.

4.6.2 Storage

- During the storing of the code from an honest uploader, the adversary could modify the error code, corrupting the same. (Mn, s) Reed-Solomon codes, can tolerate up to

$\frac{Mn-s}{2}$ errors and $Mn - s$ erasures. Considering that we wish to tolerate approximately $Mn/2$ errors, we would need $Mn >> s$, which could be unfeasible. Thus the adversary could use this power to introduce errors to make the system infeasible.

- To ensure that the codes are not tampered, we could store a digitally signature of the RS Code from the uploader. If the code has been tampered, then it can be treated as erasure and therefore we could tolerate $Mn/2$ errors.

4.6.3 Download

In upload or storage phase there is no requirement of anonymity or security, since all the communication is open to all to read. However, during download, we need to ensure that no adversary can find out which file is getting uploaded, or if someone is accessing any file while maintaining the liveness of the file download

- **Liveness**: Given a file F, uploaded by an honest uploader, anyone should be able to download the file in presence of less than $\frac{Mn-s}{2M}$ adversaries. Since the Downloader requests for codes from all the peers in the network, it would receive at-least the s codes from honest peers, through which the complete file can be derived.

- **Correctness**: Given a file F, uploaded by an honest uploader, the codes derived from all users should correctly derive the actual file. (Mn, s) Reed-Solomon erasure codes [6] ensure that recovery of the file given $Mn - s$ erasures is possible. Given that errors are detected using a digital signature, they can be treated as erasures.

- **Anonymity**: For any honest peer downloading a file F, no adversary snooping the entire communication network should be able to obtain knowledge that the peer requested the file. This is ensured by the usage of the communication rounds and CPIR, in which even the uploader is unable to ascertain whether a file is being downloaded. Since in a communication round each node is requesting CPIR from every other node (dummy requests wherever required), no adversary can say whether the particular user ever requested a file.

- **File download anonymity**: Since the file was requested using CPIR scheme, it doesn't let any adversary including the peer who has been requested, to know which file was accessed across all the file codes stored on their storage.

5 Analysis

5.1 Storage

If the files have size s, the storage required on each individual machine is $O(\frac{MKs}{n})$ for some security parameter M. If all of the files were stored on one server, the storage required on each machine would be $O(Ks)$. Therefore, our system can reduce the load on a given machine on the network by a factor of M/n.

5.2 Communication Rounds

In each communication round, each node will need to send $O(n)$ PIR requests, for a total of $O(n^2)$ PIR requests sent during a given round. In addition, each PIR request will require $O(c)$ communication rounds for some parameter c. Therefore, we will need a total of $O(n^2c)$ messages in each communication round. If $O(n)$ of those requests were real, i.e. communication on the network is dense, then this is only an $O(nc)$ blowup in the number of messages that must be sent over the network. However, depending on the sparsity of the network this blowup could be significantly larger, up to $O(n^2c)$ if only $O(1)$ real requests are sent in a given communication round.

The increase in the number of messages sent would be offset by the fact that each message would have to be substantially smaller than if our protocol was not implemented. If our protocol were not used, the request to the server hosting the file and response from the server would need to have size $O(s)$, but using this protocol we can reduce the size of the message to $O(\frac{M}{Sn}) = O(M/s)$. If $M/S \approx 1$, then that is a $1/n$ improvement to the size of messages.

Combining these two results, we see that if $O(1)$ requests are sent in each communication round, then the total amount of data that must be sent over the network has an $O(n)$ blowup over the situation where our protocol is not used. If the total amount of requests sent in a given communication round is $O(n)$, then the blowup in data sent over the network is only $O(1)$. Since all the n communications are happening in parallel in a communication round, we could possibly achieve faster download speeds than in a single server no privacy setting.

6 Conclusion

6.1 Summary of Results

In this paper we outline a protocol for storing information in a distributed manner for private retrieval. Users of the protocol can save a significant amount of storage space by loading their data into this network rather than storing the files locally. In addition, users can download files from the network without their identity or the identity of the sought after file being revealed. Finally, our protocol can potentially accelerate file downloads by reducing the amount of work that must be done by the peer which originally uploaded the file by allowing all download requests to happen in parallel.

6.2 Future Work

This protocol has much room for improvement, and several promising avenues for future work exist.

- We have not yet implemented this protocol, and currently have no concrete information on its performance in a real world scenario.

- It is possible that the PIR requests on the network could be made more efficient by exploiting the fact that the index is constant across the peers.

- More work needs to be done on implementing this protocol on sparser network topologies. We describe a protocol based on TOR which could address this issue, but the details of the protocol still need to be fleshed out.

7 Bibliography

1. Polynomial Codes Over Certain Finite Fields, I.S. Reed and G Solomon, Journal of the Society for Industrial and Applied Mathematics, 8(2), 300–304. (5 pages)

2. Gupta, Trinabh. et al. "Scalable and Private Media Consumption with Popcorn."

3. Cohen, Bram. "BitTorrent.org." BitTorrent, 10 Jan. 2008, www.bittorrent.org/beps/bep_0003.html.

4. " Tor: Onion Service Protocol." TorProject, 2019.www.torproject.org/docs/onion-services.html.en.

5. J. P. Stern. A new and efficient all-or-nothing disclosureof secrets protocol. InInternational Conference on theTheory and Application of Cryptology and InformationSecurity (ASIACRYPT), 1998.

6. Kumar, Arvind. (2019). "Reed-Solomon Code: Principle , architecture and Implementation".

8 APPENDIX:Communication Protocol

Independent of the FTP protocol (attempt to remove the fully connected assumption)

This section describes a TOR like network to be used for communication. In the current TOR network, to communicate through the network, we need to store the whole graph connection (Since the real world connection graph is not fully connected) to establish a circuit to communicate to a peer/server in the network. This does not scale to real world communication graph since the size has just become too large. There are two major ways of overcoming this limitation: Query using PIR to establish the relays in the network; Random Walk Based information propagation.

8.1 Assumptions

- Number of peers in the network is N.
- List of peers is known to the network. This can be achieved through a broadcast of peer identity in the network, when entering the network.
- ENC is an encryption scheme, safe across variable size messages.
- The communication graph is connected, i.e. all nodes are reachable through some path in the network graph, from any node.

8.2 Ideal security

Lets say Alice (A) needs to send a message to Bob (B)

- No adversary should know that Alice sent a message, unless in collusion with Alice.
- No adversary should be able to tell that Bob received a message unless in collusion with Bob.

- No adversary should be able to decode the message sent unless in collusion with Alice or Bob.

8.3 Protocol

- Alice randomly selects a random number of nodes $r\epsilon[1, K]$, from the list of all nodes. These serve as relays in the network. Let these peers be $\{P_1, P_2, ..., P_r\}$
- Alice generates an encrypted message E_1 from message m as follows:

$$E_1 = (P_1, ENC_{PK_{P_1}}(E_2))$$

$$E_2 = (P_2, ENC_{PK_{P_2}}(E_3))$$

$$...$$

$$E_r = (P_r, ENC_{PK_{P_r}}(E_m))$$

$$E_m = (B, ENC_{PK_B}(m))$$

- Alice sends this message E_1 to x neighbors out of all the neighbors, who are directly connected to A. This starts a multi-random walk with destination as P_1, which is written clear-text in E_1.
- All peers who received the message would forward the message to its neighbors (Only to P_1, if directly connected, otherwise to x of the neighbors among its directly connected peers)
- When the walk reaches P_1, he will decrypt the message with his secret key to get E_2. P_1 re-initiates a walk with P_2 as destination written in clear-text.
- This continues till the message reaches the destination, i.e. Bob, who decrypts E_m to receive message m.

The above protocol is able to successfully deliver a message from Alice to Bob. It is trivial to show that the identity of Alice is not revealed to any peer in the network, as Alice could be a relay node or a random walk node.

For any node, the probability that the given clear-text destination node is Bob would be

$$P = 1/(r+1)$$

Since $r \in [1, K]$,

$$P = \frac{1}{K}(\frac{1}{2} + \frac{1}{3} + ... + \frac{1}{K})$$

$$P = \frac{\ln(K) + \gamma}{K}$$

.

Since the final message can only be decrypted by Bob, nobody except for Bob (and his colluders) and Alice (and her colluders) who sent the message can know about the message.

The above protocol is therefore secure in sending the message, but if Bob wants to reply to Alice, he doesn't have any way to contact or know who Alice is. Thus, if a two way communication needs to be initiated by Alice, she would need to send Bob a secret key, and a list of randomly shuffled nodes (of size K) out of which one is Alice. Bob would use these nodes to communicate the message back, with reply encrypted with secret key appended to the original scheme, so that any of the relays as chosen by Bob could be Alice. This reduces the security of Alice to be the same as Bob. The above is insecure against malicious adversary, since if Bob is an adversary he could ignore one or more of the nodes that Alice had sent to improve the probability of knowing who Alice is. We need to find a solution to receive back a message in malicious case

8.4 Security against snooping attack

An adversary could monitor the network to find the origin and destination of the message sent, and connect relays if the network is under continuous surveillance. To prevent this, we used a timed message sending protocol. Each peer sends a message to exactly k random neighbours (for multi-walk), after time t. The message contains y number of messages in a list. IF the number of messages that the peer needs to propagate is less than y, he sends dummy message forward, essentially making y messages. This way snooper cannot tell where the message started and/or where it ended

8.5 Route Capture Attack Security

In Random Walk based TOR algorithms, a major attack is Route Capture attack. In this attack adversaries only pass the message forward to other adversaries therefore stopping the walk, and through monitoring communication they can get who sent the message and to whom the message was sent, breaking anonymity. We get over this attack, by having no communication from the owner to the forwarding peer, thus no information gained about Alice. The adversary would get to know the next relay node in the circuit. This does not reduce the privacy of Bob as well. If the message passes through z adversarial relay nodes, the probability of the next node being Bob reduces to $\frac{ln(K-z)}{(K-z)}$

8.6 Benefits over TOR and related algorithms

- Deals with large graph, though compromising more on the speed of communication.
- Tor provides anonymity in the sense that there is no way to track communication between Alice and Bob, whereas in our scheme we are able to provide security that Alice ever sent a message and Bob ever received a message.