Deniability - an Alibi for Users in P2P Networks

Ofer Hermoni, Niv Gilboa, Eyal Felstaine and Sharon Shitrit
Department of Information Systems Engineering,
Ben-Gurion University of the Negev, Beer-Sheva, Israel
{oferher,gilboan,eyalfe,shitrits}@bgu.ac.il

***Abstract*—Peer to peer file sharing is ’booming’, but meanwhile censorship of these networks and prosecution of users that share censored content are growing just as quickly. In this paper, we propose a novel notion of *deniability* as an easy and efficient method for users to avoid censorship and prosecution. The fundamental concept is that a given data element, which contains controversial or censored content, is also associated with neutral-content material. Hence, even a powerful adversary capable of monitoring all communication in the network and viewing the internal state of participating hosts is unable to prove a link between censored content and a user. The communication overhead required to retrieve a document is only four times greater than what is needed in a standard network. The storage required for a document is only twice as large as the document itself. Deniability is an elegant alternative to user anonymity in P2P file sharing networks. Systems that provide anonymity for users typically require greater overhead and do not guarantee anonymity against powerful, real-world adversaries.**

I. Introduction

Peer-to-peer (P2P) networks have spread dramatically in recent years. A prime application of these networks is file sharing, which allows users to share content easily with other users of a network. The decentralized architecture of file-sharing P2P networks makes them important for freedom of speech, since *any* network user can perform publication, distribution and retrieval of content.

Opposition to P2P file sharing networks has been growing just as quickly as the networks themselves. P2P networks excel at disseminating information among millions. It is therefore natural that individuals and organizations that prefer to limit the distribution of information would do their utmost to censor such networks. There are various reasons for censoring P2P file-sharing networks, ranging from protecting copyright to re-pressing political ideas. The standard method for implementing censorship is to prosecute users of a file sharing network in a court of law. Sometimes, even the threat of such prosecution is enough to deter people from publishing, serving or reading files in P2P networks. An analysis of what may or may not be legally censorable is beyond the scope of this paper.

Resisting censorship of P2P file sharing networks is an active area of research. One approach to achieving censorship resistance is to ensure *anonymity* for participants in a file-sharing network [1] – [20]. The rationale behind the *anonymity* approach is that if the publisher, server and reader of a document all remain anonymous, then legal action cannot be brought against them, and they are thus not affected by censorship.

This Research is Supported by the Institute for Future Defense Technologies Research named for the Medvedi, Shwartzman and Gensler families.

There are two disadvantages to most works that claim to provide anonymity in P2P file sharing networks. The first is that overhead in terms of communication is greatly increased compared to standard P2P file sharing networks. The second, more profound, disadvantage is that anonymity is achieved only against limited adversaries. A real-world opponent of P2P file sharing is often a representative of powerful interests such as government or large corporations. An adversary of this sort may very well use court orders to eavesdrop on traffic in the P2P network and operate nodes in the network. Most proposals for anonymous networks fail to ensure anonymity when faced with such attacks.

Another approach makes it difficult to remove censored content without destroying legitimate content. This approach, which was introduced in [21] and [22], does not protect the participants of the network against prosecution. Their identities and the content they serve or read can be determined by a real-world adversary.

Our contribution to P2P anonymity is twofold: we formally define the new concept of *deniability* in P2P file sharing networks, and we propose a practical network that implements this concept. Deniability is the property of having a solid alibi: a user of a P2P file sharing network has deniability if s(he) can claim that his/her actions are legitimate. In other words, a user who performs an illegitimate operation, such as downloading censored documents, cannot be distinguished from a user who performs a lawful operation, such as retrieving a legitimate document.

We propose a network architecture that ensures deniability to both reader and server in a file-sharing network. The main idea is as follows. Each document is divided into shares by a secret sharing scheme [23]. Each share is associated with at least one legitimate document and possibly with a censored document. Shares are then distributed among several servers and stored as one share per server. Serving and retrieval of a document is done in a way that guarantees deniability.

Our scheme ensures deniability against powerful adversaries who can monitor all the traffic in the network and can even view the internal state of every reader and server. The overhead in communication and storage incurred by scheme is small. The communication complexity is at most four times greater than the communication complexity of a P2P file sharing network that does not offer deniability. The amount of storage is twice as large as what is needed in a standard P2P file sharing network.

A. Organization

This paper is organized as follows. In Section II, we discuss related works. In Section III we give a simple model for a P2P file sharing network. In Section IV we define the new concept of deniability in P2P networks. In Section V we show our protocol and prove several of its properties. Finally, we cover the mathematical basis for our solution in Section VI.

II. Related work

Many works on P2P anonymity and censorship resistance have been published. In this section we discuss a small number of systems that are most relevant to our work.

A. Anonymous Systems

The core concept of providing Internet anonymity goes back to the early days of the public network and has been extensively studied since then. Chaum [1] proposed to use an intermediary proxy (relay server or Mix) whose aim is to hide the identity of the reader from the server. However, an opponent could still figure out the identity of the endpoints by eavesdropping on communication lines and by performing timing analysis of the messages going in and out of the relay. A common solution to this problem is to use thwarting techniques [1], which include the sending of dummy messages and the introduction of random delays to message forwarding. Trusting a single relay server proved to be dangerous, since that server could potentially be controlled by an adversary.

Later works enhanced Chaum's MixNet approach by deploying predefined or ad-hoc paths and in addition by using encryption. Crowds [19], for example, uses symmetric encryption between every pair of adjacent nodes in a path. Crowds is based on the idea that people can be anonymous when they blend into a crowd. The sender forwards requests to a randomly selected relay. Once the message is received by the relay, the latter ”flips a coin” and decides whether to forward the message to another, randomly chosen, relay or to send it directly to the end server. Neither the end server nor any of the intermediate relays can determine whether the message was received from the originator of the message or from a relay.

Onion Routing [8] uses a fixed, predefined path, which is essentially a list of intermediate proxies leading to the destination. The predefined path is established by the source and is attached to the message. The message is sent by the source to the destination containing layers of encryption that are peeled off at each step to reveal the address of the next relay on the path. The major advantage of onion routing is that relays cannot unravel the information received or determine destination address.

Another solution, Freenet [10] introduces a degree of server, document and publisher anonymity; this, at the cost of greater communication, computation, retrieval time and storage overhead. When a user requests a document, he uses a document identifier to send the query without being aware of the server's identity or location. Freenet places copies of files with similar identifiers in predetermined areas or clusters. The query is directed towards the 'area' that is most likely to hold the desired file. Responses to queries are routed in the same path traversed by the query, only in the opposite direction. In this way, Freenet achieves server anonymity, since no node along the path knows the real source of the document. Publisher anonymity is obtained by using the same mechanism for inserting documents into the network.

A different technique to provide anonymity employs secret sharing schemes to break data items into several parts and distribute them among different server. In Publius [17], the content is encrypted by a key and stored by a fixed set of servers. The encryption key is shared by Shamir's secret sharing and distributed to the servers. Reading is carried out by reconstructing the key, retrieving the encrypted document and decrypting it.

Another solution that uses secret sharing technique is SSMP [20]. Queries are divided using shamir's secret sharing scheme and spread over the network. A node that collects enough shares reconstructs the query and forward it to a server. Replies are sent using both Rabin's Information Dispersal Algorithm (*IDA*) [24] and Onion Routing.

Free Haven [18] is an anonymous publishing system. It is made up of a number of servers, known as servnets, which agree to store and provide documents for anyone. The identities of these servnets are publicly known. Communication is carried out over a Mix-based communication layer. When a publisher wants to publish a file he breaks it into a number of parts using *IDA* and sends each part to a different servnet. When a reader wants to retrieve a file she must first find the hash of the required file and send it to a servnet. The servnet broadcasts the request to the other servnets, which then send the pieces of the file to the reader. Free Haven provides a certain level of publisher, server, reader and document anonymity.

There are several problems common to all of the systems described in this section. Typically, they offer a low level of anonymity and result in a large communication overhead. Anonymity is achieved against a limited adversary, however it is compromised when the adversary may control nodes or even monitor a peer's traffic. Furthermore, all the solutions that use public key cryptography are vulnerable to Man-in-The-Middle Attacks, since there is no authentication mechanism in such distributed networks.

B. Censorship Resistant Systems

Censorship Resistance means that it is hard or even impossible to remove unwanted or censored information from the system's servers. Systems such as Publius, Freenet and Freehaven provide Censorship Resistance in addition to user anonymity.

Several Censorship Resistant systems have been proposed recently. Dagster [21] and Tangler [22] prevent removal of any single document from the system by entangling documents together. Removal of a censored document results in the deletion of other which are legitimate. In the network proposed by Serjantov [25], each peer in the system can act as a server, forwarder, or decrypter. Each stored document is divided into encrypted blocks and placed on multiple servers. A forwarder acts as an intermediary between the servers and a reader; only a forwarder knows the mapping between the data blocks and the servers that store them. A decrypter is responsible for decrypting data blocks but does not have knowledge of the data-server mapping.

Censorship Resistant systems are not designed to protect their users against prosecution. Therefore, integrating the system proposed in this paper with an existing Censorship Resistant system would provide the best of both worlds.

III. System model

A. Participants

In P2P file sharing networks, information is stored in units known as *documents*. The *publisher* of a document is the entity that places the document in the system. The *server* of a document is the entity that stores and distributes the document. Documents are retrieved by *readers* from the servers. The *readers* query the *index* so as to locate the particular server for a required document. Although real-world documents are of variable sizes, we assume that all documents in our system are of fixed size. Thus, real-world documents may have to be divided into several smaller documents or be padded into a larger document in our system.

B. The Adversary

The adversary's goal is to show that a censored document was published by a certain user, retrieved by a certain reader or provided by a specific server. In this paper, we do not consider either attacks on storage or denial of service attacks. An adversary may have a variety of capabilities. A limited adversary can log into the network and act as a participating user (reader or server) or as multiple users. A powerful adversary, such as a government or a major corporation, can monitor all links in the network. In Section V we describe the resistance of our scheme against any given adversary.

C. Performance issues

Integrating deniability within a P2P network introduces overhead. Our analysis takes into account three parameters: Communication complexity, Storage complexity and computational complexity. Given a document d, *Communication complexity* indicates the total number of Bytes sent within the system while inserting or retrieving d. *Storage complexity* is the number of Bytes that need to be stored for d within the system. The last parameter is *computational complexity*, which refers to the amount of computation needed to insert a new document, and to retrieve a document. In Section V we show that the overhead associated with our design gives rise to an efficient implementation of a P2P file sharing network.

IV. Deniability

A. Preface

In this paper we introduce a new concept in place of anonymity. *Deniability*, as we call this new concept is the ability of an entity to *deny* any connection to a particular document. Many attempts to provide anonymity fail to provide it against real world adversaries. In this paper, instead of providing anonymity we achieve deniability. *Reader deniability*, *Server deniability* and *Document deniability* can be viewed in formally as analogues of anonymity definition of Free Haven [18]. *Reader deniability* means that the reader can deny her link to a specific document received by her, even if an adversary controls all the servers in the system and monitors all the traffic. *Server deniability* means that the server can deny its link to a document served by it. *Document deniability* means that the server can deny storing any censored content. Formal definitions for deniability types are described in Subsection IV-C.

B. Anonymity

One may define several types of *anonymity* with respect to P2P networks. Each type of anonymity corresponds with a different element within the network. In [18] the authors have suggested definitions for the anonymity of the participants. *Reader anonymity* means that an adversary has no way of knowing which reader on the network has retrieved a particular document. *Server anonymity* means an adversary has no way of knowing which server on the network has served this document or currently stores it. *Document anonymity* means that a server does not know which documents it is storing.

C. Definitions

In a P2P file-sharing environment it is logical to divide all documents into two categories denoted X and L. Category X contains the documents that one may want to deny any relation to (e.g. censored or unwanted). Category L contains the remaining documents, that are legitimate and bear no controversial content. An example of a legitimate document is freeware code, while a song under copyright law is censored.

Definition 1: A *record* is a contiguous block of Bytes of fixed length.

Definition 2: A *database* is a set of records and is stored by a server in a P2P file sharing system. Two operations are defined on a database: *insertion* of a new record and *retrieval* of a given record.

Definition 3: A record r is *associated* with a document d if there exists a set of records R such that $r \in R$ and two conditions are satisfied:

- Given R it is possible to obtain the document d.
- Given $R \setminus \{r\}$ it is not possible to obtain d.

Definition 4: A set of records R is called a *minimal retrieval set* for a set of documents D if:

- Every document in D can be obtained from the records in R.
- There is no subset of records $R' \subset R$ such that all the documents in D can be obtained from R'.

Definition 5: A reader maintains *reader deniability* when retrieving a set of records R, if R is a *minimal retrieval set* for a set of legitimate documents $D \subseteq L$.

Definition 6: A database maintains *document deniability* if for every record r in the database, there is a legitimate document $D \in L$ such that r is associated with d.

Definition 7: A server maintains *server deniability* if every record r that the server provides in answer to a retrieval request, is associated with a legitimate document $d \in L$.

V. Proposed Architecture

A. Preface

In our system, each document is divided into pieces called *shares*. A share sh is associated with a document d and vice versa, if by using sh (along with other shares) it is possible to reconstruct d. Each share can be regarded as a *record* (Definition 1), since shares are the basic component in every server's database.

The main idea of our system is to affiliate each non legitimate document with a set of legitimate documents. In this way serving or retrieving the non legitimate document seems identical to serving or retrieving the affiliated set of legitimate documents. Therefore, an adversary cannot distinguish between the two transactions and deniability is preserved.

Table I contains notation that is used in this section.

B. Indexing

Retrieving a document in the system relies on the *indexing* mechanism. The *indexing* mechanism contains a database that includes information about the documents, the shares and their locations. The *index* enables a user to search a specific document within the system in order to get the set of all the shares associated with the document and the locations (servers) of those shares. The *index* also enables a user to search a specific share in order to get the set of all the documents associated with that share. An *index* in our system may be viewed as a bipartite graph, as shown is Fig. 1, the square nodes indicate the documents and the circular nodes indicate the shares. Each edge in the graph indicates that a particular share is associated with a particular document and vice versa. For example, in the figure, the document ℓ_1 is associated with the shares sh_1, sh_2 and sh_3 and the share sh_1 is associated with documents ℓ_1 and x_1.

Anonymous indexing, which means that the user's query is not revealed, is assumed in our design and is essential in order to preserve *reader deniability*. *Anonymous indexing* can be achieved by downloading the entire index or more efficiently by Private Information Retrieval (PIR) [26] – [30].

C. Deniability transformation

In order to implement our deniability scheme, we design a new function called *Deniable Secret Sharing*, DSS. Deniable Secret Sharing is a variation of Shamir's Secret Sharing [23], which may have several shares fixed before the sharing takes place.

Formally speaking, DSS_t takes as input a document d and k shares denoted by sh_1, sh_2,...,sh_k, where $0 \leq k \leq t-1$. The output of DSS_t is t shares denoted by sh_1, sh_2,...,sh_t. DSS_t is a t-threshold secret sharing scheme and the t output shares allow to reconstruct the original document d. An illustration of the input and output of DSS_t appears in the left side of Fig. 2. Note that the number of output shares is identical to the threshold and that the first k output shares are the same as the k input shares.

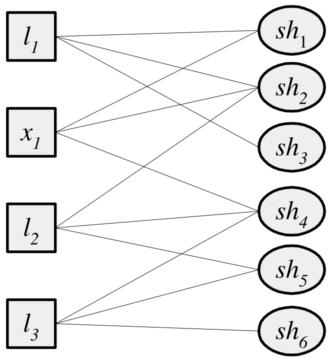

Figure 1. *Indexing* system; □ - document, ◯ - share

The document d, originally inserted into DSS_t is the result of entering the t output shares of DSS_t as the input of the reciprocal function DSS_t^{-1} (see the right side of Fig. 2). We explain the basis for DSS_t and DSS_t^{-1} in Section VI.

DSS_t is used in two ways. For the initialization phase (no shares exist yet), we use DSS_t with $k = 0$. The other way in which DSS_t is used, is with $k = t-1$, so that the only new share in the output is the last share sh_t, since $t-1$ shares are used as input.

D. Publishing - Insertion of a new document

The shares in the system are divided into three categories. The first category, *LL*, contains shares that are associated only with files from the legitimate set *L*. The second category, *XX*, contains shares that are associated with files from set *X* only. The last category, *XL*, contains shares associated with documents from both sets *X* and *L*. Shares from category *XX* are undesirable since a user that stores such a share can be implicated as serving censored content. In other words, the user can't claim that one of the legitimate documents in the network can be constructed with such a share. Hence, all shares in the system are put into a *'pool of shares'* that contains only shares from categories LL and XL.

The first document to be inserted by a publisher to the system is a legitimate document. In this *Initialization* phase, all the shares are associated only with legitimate content and can therefore be added into the *pool*.

The general algorithm for inserting a new document (see Algorithm 1), which is described below takes as input three parameters; d is the document to be inserted into the network, ℓ is an arbitrary legitimate document and *pool* is a pointer to the pool of shares.

Insertion (d, ℓ, pool): The publisher constructs a set of shares, Ψ, by randomly selecting $t-1$ shares from the pool. The document d and Ψ are put into DSS_t and the output is a set of shares $sh_d[1, t]$ (t shares that are associated with d). If d is a censored document, i.e. $d \in X$, then the new share $sh_{dt} \in XX$ and so it cannot be put into the *pool*. In order to eventually publish the share sh_{dt}, the publisher does the following. It constructs a set of shares, Θ, by randomly selecting $t-2$ shares from the pool and adding the share

TABLE I
NOTATION

Notation	Description
X	Set of censored documents
L	Set of legitimate documents
XX	Set of shares associated with X documents only
LL	Set of shares associated with L documents only
XL	Set of shares associated with both X and L documents
t	The number of shares in the output of DSS_t
k	The number of shares in the input of DSS_t
$sh_d[1,i]$	Set of i shares associated with document d
sh_{di}	The i-th share associated with document d
Λ_i	Set of all documents associated with share sh_i

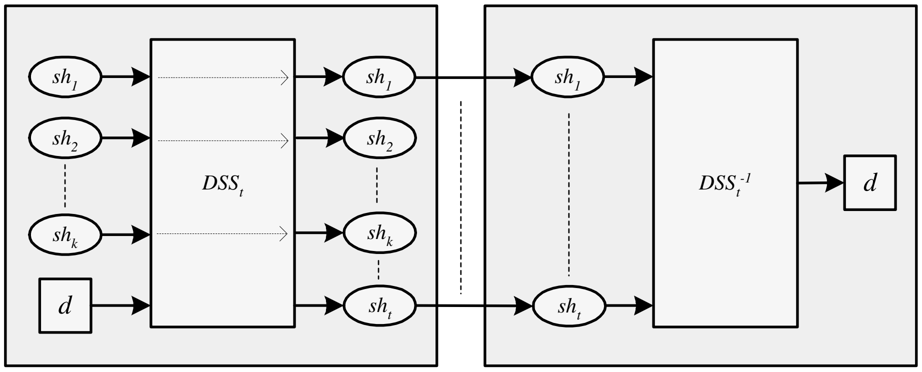

Figure 2. (left) $DSS_{n,t}$; (right) $DSS^{-1}_{n,t}$. $DSS_{n,t}$ - Input: k shares and document d; Output: n shares. $DSS^{-1}_{n,t}$ - Input: t shares; Output: document d

sh_{dt}. The legitimate document ℓ and Θ are put into DSS_t and the output is a set of shares $sh_\ell[1,t]$. Now, the share sh_{dt} is associated both with d and ℓ, therefore $sh_{dt} \in XL$ and can be put in the *pool* along with the new share that was created for ℓ, which is denoted by $sh_{\ell t}$.

An example of inserting a document, where $d \in X$ and $t = 3$ appears in Fig. 3. In this example two random shares sh_{R1} and sh_{R2} along with d are put into DSS_t. A new random share sh_{R3} along with sh_{dt} and a legitimate document ℓ are again put into DSS_t function. Now the share sh_{dt} is associated with ℓ and therefore $sh_{dt} \in XL$ and can be put into the pool, along with the share $sh_{\ell t}$. Note that it is possible to reconstruct d by putting the shares sh_{R1}, sh_{R2} and sh_{dt} into DSS^{-1} and reconstruct ℓ by putting the shares sh_{R3}, sh_{dt} and $sh_{\ell t}$ into DSS^{-1}.

Algorithm 1 $Insert\ (d, \ell, pool)$

```
$\Psi \leftarrow t-1$ random shares from pool
$sh_d[1,t] \leftarrow DSS_t(\Psi, d)$
**if** $d \in X$ **then**
    $\Theta \leftarrow t-2$ random shares from pool
    $\Theta \leftarrow \Theta \bigcup \{sh_{dt}\}$
    $sh_\ell[1,t] \leftarrow DSS_t(\Theta, \ell)$
**end if**
Put $sh_{dt}$ and $sh_{\ell t}$ (if $d \in X$) into the pool and update the index
```

E. Document Deniability

In this section we show that the database of every server in our system has the property of *document deniability*.

Theorem 1: The database of every server in the network ensures document deniability.

Proof: We prove *document deniability* by induction. An empty database trivially has the property of *document deniability*. We next show that adding a document to the database using Algorithm 1 maintains this property.

When a legitimate document d is inserted, only one new share is created - sh_{dt}. This share along with the $t-1$ shares that are randomly selected from the pool are necessary and sufficient to obtain d. Hence, according to Def. 3, sh_{dt} is *associated* with d. Therefore, according to Def. 6, if the system maintained *document deniability* before the insertion of d, it still maintains *document deniability* after the insertion.

When a censored document d is inserted, two new shares are added to the database: sh_{dt} and $sh_{\ell t}$. Both new shares together with $t-2$ shares that are randomly picked from the pool are necessary and sufficient to obtain ℓ. Hence, due to Def. 3 they are *associated* with a document $\ell \in L$. Therefore, according to Def. 6, if the system maintained *document deniability* before the insertion of d, it still maintains *document deniability* after the insertion. ■

F. Retrieval

A naive way in which a reader can retrieve a document d is as follows. First, the reader queries the index for all the shares of d and their locations. Then, the reader retrieves all the shares of d and reconstructs the document. The problem

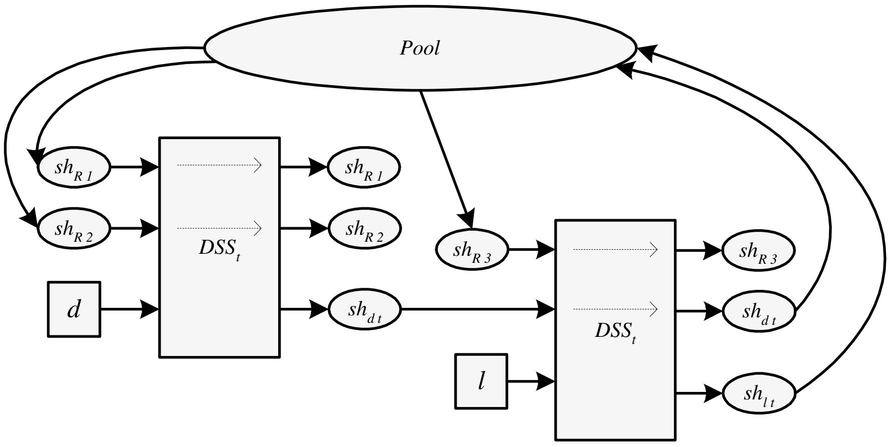

Figure 3. Example for shares creation in the process of inserting a censored document d

with this procedure is that if $d \in X$ then the reader does not preserve the property of *reader deniability*. In particular, an adversary that monitors all the traffic received by the reader can link the reader to the retrieved document.

In order to prevent the adversary from linking the reader to a censored document d, we suggest another way to download d (see Algorithm 2). If $d \in L$ then the reader simply retrieves all of the shares of d. However, if $d \in X$ then for each share of d the reader queries the index for a legitimate document associated with it, and retrieves all the shares of these legitimate documents. For example, if the reader wants to obtain the document x_1 in Fig. 1, the first step is to query the index for the shares associated with x_1, the result is $\{sh_1, sh_2, sh_4\}$. For each of those shares the reader queries the index for the list of documents associated with them. The result is $\{\ell_1, x_1\}$ for sh_1, $\{\ell_1, x_1, \ell_2\}$ for sh_2 and $\{x_1, \ell_2, \ell_3\}$ for sh_4. The reader picks one legitimate document from each list and retrieves it, e.g., ℓ_1 for sh_1 and ℓ_2 for sh_2 and for sh_4. In fact, using this method, the reader retrieves three documents: ℓ_1, ℓ_2 and x_1. No adversary can distinguish between this situation and the one where the reader retrieves only the legitimate documents ℓ_1 and ℓ_2.

We say that a share sh is *alive* if it is possible to retrieve a legitimate document associated with it, i.e., all of the shares of the legitimate document exist in the system. Not all of the shares in the system are necessarily *alive* because users may log out and shares may be deleted. If during retrieval one of the shares sh_i is not *alive* then the reader does not retrieve d.

Algorithm 2, which is executed by a reader, receives as input a document d to be retrieved and a pointer to the index. Two operations on the index are used: index.get-shares receives as input a document d and returns $sh_d[1,t]$, all the shares that are associated with that document, while index.get-documents receives as input a share sh_i and returns Λ_i, all the documents with which the share is associated.

Theorem 2: The network ensures reader deniability.

Proof: In Algorithm 2, the set of shares $sh_d[1,t]$ is retrieved when the reader wishes to obtain d. If $d \in L$ then $sh_d[1,t]$ is a minimal retrieval set of shares (equivalently, records) for the set of documents $\{d\} \subseteq L$. Thus by Definition 5, the reader maintains *reader deniability*.

If $d \in X$ then if one of the shares in $sh_d[1,t]$ is not alive then the algorithm halts and no retrieval is performed, trivially maintaining *reader deniability*. Otherwise, the reader retrieves for every $i = 1,\dots,t$ the set of shares $sh_{\ell_i}[1,t]$ in which every share is *associated* with the legitimate document ℓ_i. Indeed, since $sh_{\ell_i}[1,t]$ is a minimal retrieval set for $\ell_i \in L$ we have that $\bigcup_{i=1}^{t} sh_{\ell_i}[1,t]$ is a *minimal retrieval set* for $\{\ell_1,\dots,\ell_t\}$, where $\{\ell_1,\dots,\ell_t\} \subseteq L$. Hence, according to Definition 5 the reader maintains *reader deniability*. ■

Algorithm 2 $Retrieve\ (d, index)$

```
$sh_d[1,t]$ = index.get-shares($d$)
**if** $d \in L$ **then**
    retrieve $sh_d[1,t]$
**else**
    **if** $\exists i$, s.t. $sh_{di}$ is not alive **then**
        STOP
    **else**
        **for** i=1,...,t **do**
            $\Lambda_i$ = index.get-documents($sh_i$)
            Let $\ell_i \in \Lambda_i \bigcap L$
            $sh_{\ell_i}[1,t]$ = index.get-shares($\ell_i$)
            Retrieve $sh_{\ell_i}[1,t]$
        **end for**
    **end if**
**end if**
$d$ = $DSS_t^{-1}(sh_d[1,t])$
```

The reader can settle for *partial reader deniability* in case of a weak adversary. We say that an adversary is weak if it can intercept at most one retrieved share during the retrieval procedure. For example, it controls only one server. Note that it is possible to generalize this definition of weak adversary to any number of monitored shares m as long as $m < t$. *Partial reader deniability* means that the reader can deny that she retrieved d as long as at most one share is observed by an adversary. Retrieval achieving *partial reader deniability* is carried out by checking if all the shares of d are *alive*, and if they are, retrieving all these shares. All the shares must be *alive* because if a share is associated solely with a censored document, an adversary can link the reader with that document. In this way the communication overhead is decreased to t as discussed in Subsection V-G.

In order to maintain *server deniability*, when a server gets a request for a particular share sh it has to verify that sh is *alive* before serving it to the reader. Otherwise, if the server serves a non live share, an adversary can link between the server and the censored content associated with sh. If sh is not alive the server deletes the share and updates the index.

Theorem 3: The network ensures server deniability.

Proof: Since the server provides only shares that are associated with legitimate documents, according to Def. 7, the server maintains *server deniability* ■

G. Performance analysis

In this section we analyze the complexity of the system in terms of communication, computation and storage.

Given a document d, *Communication complexity* indicates the total number of Bytes sent within the system while inserting or retrieving d. Inserting d, if $d \in L$, involves retrieving $t-1$ random shares from the pool of shares and sending one new share to a server. Thus the communication complexity is $t|d|$ Bytes. If $d \in X$ the total communication complexity for insertion is $(2t-1)|d|$ Bytes. The *Communication complexity* for retrieving a document again depends on the type of the document. If $d \in L$, then the reader has to retrieve only t shares of size $|d|$ Bytes each, with total *Communication complexity* of $t|d|$ Bytes. Otherwise, if $d \in X$, then the reader has to retrieve one legitimate document for each share associated with d, i.e., the reader has to retrieve at most t legitimate documents, with total *Communication complexity* of $t \cdot t|d| = t^2|d|$ Bytes. If *partial reader deniability* is sufficient the reader has to retrieve only t shares, thus the total *Communication complexity* is reduced to $t|d|$ bits.

Storage complexity indicates the number of Bytes stored within the system for each document. In case of a legitimate document, only one share is added to the system, ($t-1$ shares are randomly selected and only one share is created). Therefore the storage complexity is $|d|$ Bytes. If a censored document is inserted, the process is repeated twice (see Section III), therefore, total of two new shares are created, yielding storage complexity of $2|d|$ bits.

The *Computation complexity* is based on Shamir's secret sharing [23]. The complexity for dividing a document of size of $|d|$ into t shares, or reconstructing d from its t shares is $O(t|d|)$.

Improving the performance of the system can be achieved by decreasing the threshold t as much as possible. From the point of view of *deniability*, the value of the threshold t can be as small as two; all deniability types (reader, server and document) are preserved. The results of performance analysis using $t = 2$ can be found in table II. These results compare favorably with the overhead in various P2P file sharing networks that offer anonymity for users.

VI. Deniable Secret Sharing

In [23] Shamir has first shown how to share a secret. In this section we explain how to use Shamir's secret sharing scheme to perform the deniable secret sharing presented as a black box in Section V.

A. Shamir's Secret Sharing

In this section we explain Shamir's secret sharing in brief; for further details see [31]. The idea of secret sharing is to start with a secret, and divide it into pieces called *shares*, which are distributed amongst users in a way that allows reconstruction of the original secret. The mathematical basis is as follows. To share a secret s among n entities and ensure that no less than t ($t \leq n$) participants are required to recover the secret, a trusted party T creates a random polynomial $f(z)$ of degree $t-1$:

$$f(z) = a_0 + a_1 z + ... + a_{t-1} z^{t-1} \tag{1}$$

This polynomial is constructed over a finite field $GF(q)$, which is known to all participants. The coefficient a_0 is the secret s and all other coefficients are random elements in the field. T publicly chooses n random distinct evaluation points: z_i, and secretly distributes the shares $sh_i(s) = (z_i, f(z_i))$, $i = 1...n$ to each participant. We can prove that the secret s can be reconstructed from every subset of t-shares. Using Lagrange interpolation, given t points (z_i, y_i), $i = 1...t$, we have:

$$f(z) = \sum_{i=1}^{t} y_i \prod_{j=1, j \neq i}^{t} \frac{z - z_j}{z_i - z_j} (mod p) \tag{2}$$

Thus,

$$s = f(0) = \sum_{i=1}^{t} y_i \prod_{j=1, j \neq i}^{t} \frac{-z_j}{z_i - z_j} (mod p) \tag{3}$$

Therefore, each group of t shares is sufficient to compute the secret s.

B. Deniable secret sharing - Details

In this subsection we show how to implement DSS_t, (see Fig. 2) using Shamir's secret sharing. The inputs of the DSS_t function are the shares $sh_1, ...sh_k$, where $0 \leq k \leq t-1$ and a document d. The output of DSS_t is a set of t shares. We use the $k+1$ input arguments in order to construct an interpolation polynomial of degree $t-1$: $f(0) = d$, $f(1) = sh_1$, $f(2) = sh_2$,...,$f(k) = sh_k$. If $k+1 < t$ we randomly select values sh_{k+1},...,sh_t and thus the remaining points are: $f(k+1) = sh_{k+1}$,...,$f(t) = sh_t$. These shares now represent the same interpolation polynomial that appears in Equation 2. The inputs of the DSS_t^{-1} function are the shares $sh_1, ...sh_t$ and the output is $f(0)$, where f is the interpolation polynomial.

Fig. 4 shows a typical example of the usage of DSS in our network. It includes three lines (polynomials of degree one), representing three documents x_1, ℓ_1 and ℓ_2. The values of the polynomial f_{x_1} are $f_{x_1}(0) = x_1$, $f_{x_1}(1) = sh_1 = f_{\ell_1}(1)$ and $f_{x_1}(2) = sh_2 = f_{\ell_2}(2)$, i.e., the first share of x_1 is the same as the first share of ℓ_1 and the second share of x_1 is the same as the second share of ℓ_2.

TABLE II
PERFORMANCE [BITS]; $t = 2$, $|d|$-DOCUMENT SIZE

	Legitimate Doc	Censored Doc
Communication - insert	$2\|d\|\ bits$	$3\|d\|\ bits$
Communication - retrieve	$2\|d\|\ bits$	$4\|d\|\ bits$ partial - $2\|d\|\ bits$
Storage	$\|d\|\ bits$	$2\|d\|\ bits$
Computation (insert / retrieve)	$O(\|d\|)$	$O(\|d\|)$

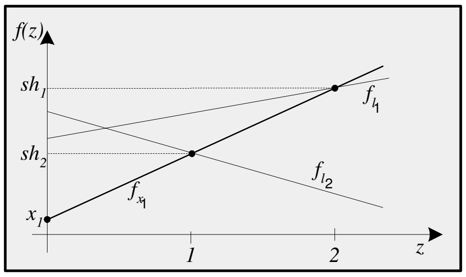

Figure 4. Example to the polynomial presentation of DSS

REFERENCES

[1] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 4, no. 2, February 1981.

[2] O. Berthold, H. Federrath, and S. Köpsell, "Web-MIXes: A system for anonymous and unobservable Internet access," in *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, vol. LNCS 2009, 2000, pp. 115–129.

[3] C. Gülcü and G. Tsudik, "Mixing e-mail with Babel," in *Proceedings of the Symposium on Network and Distributed System Security (SNDSS '96)*, IEEE, February 1996, pp. 2–16.

[4] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a type III anonymous remailer protocol," in *IEEE Symposium on Security and Privacy*, May 2003, pp. 2–15.

[5] M. Freedman and R. Morris, "Tarzan: A peer-to-peer anonymizing network layer," in *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, D.C.

[6] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman, "Mixmaster protocol version 2," June 2005, on-line: http://www.ietf.org/internet-drafts/draft-sassaman-mixmaster-03.txt.

[7] M. Rennhard and B. Plattner, "Practical anonymity for the masses with morphmix," in *Proceedings of Financial Cryptography (FC '04)*, A. Juels, Ed. Springer-Verlag, LNCS 3110, February 2004, pp. 233–250.

[8] P. Syverson, D. Goldsclag, , and M. Reed, "Anonymous connections and onion routing," in *Proceedings of the IEEE 18th Annual Symposium on Security and Privacy*, Oakland, California, 4–7 1997, pp. 44–54. [Online]. Available: citeseer.ist.psu.edu/syverson97anonymous.html

[9] V. Scarlata, B. Levine, and C. Shields, "Responder anonymity and anonymous peer-to-peer file sharing," in *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, 2001.

[10] I. Clarke, O. Sandberg, B. Wiley, and T. Hong., "Freenet: A distributed anonymous information storage and retrieval system," in *Workshop on Design Issues in Anonymity and Unobservability*, 2000, pp. 46–66.

[11] A. Beimel and S. Dolev, "Buses for anonymous message delivery," *Journal of Cryptology*, vol. 16, no. 1, pp. 25–39, 2003.

[12] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untranceability," *Communication of the ACM*, vol. 24, no. 2, 1988.

[13] S. Dolev and R. Ostrovsky, "Xor-trees for efficient anonymous multicast and reception," *ACM Transactions on Information and System Security*, vol. 3, no. 2, pp. 63–84, May 2000.

[14] R. Sherwood and B. Bhattacharjee, "A protocol for scalable anonymous communication," 2002. [Online]. Available: citeseer.ist.psu.edu/sherwood02protocol.html

[15] The Anonymizer, on-line: http://anonymizer.com/.

[16] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[17] A. R. Marc Waldman and L. Cranor, "Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system," in *Proceedings of the 9th USENIX Security Symposium*, August 2000, pp. 59–72.

[18] R. Dingledine, M. J. Freedman, and D. Molnar, "The free haven project: Distributed anonymous storage service," in *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, July 2000.

[19] M. K. Reiter and A. D. Rubin, "Crowds: anonymity for Web transactions," *ACM Transactions on Information and System Security*, vol. 1, no. 1, pp. 66–92, 1998. [Online]. Available: citeseer.ist.psu.edu/article/reiter97crowds.html

[20] L. X. R. X. Jinsong Han, Yunhao Liu and L. M. Ni., "A mutual anonymous peer-to-peer protocol design," *ipdps2005*, vol. 1, p. 68, 2005.

[21] A. Stubblefield and D. Wallach, "Dagster:censorship-resistant publishing without replication," Rice University, Dept. of Computer Science, Tech. Rep. TR01-380, July 2001.

[22] M. Waldman and D. Maziéres, "Tangler: a censorship-resistant publishing system based on document entanglements," in *ACM Conference on Computer and Communications Security*, 2001, pp. 126–135. [Online]. Available: citeseer.ist.psu.edu/waldman01tangler.html

[23] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[24] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *J. ACM*, vol. 36, no. 2, pp. 335–348, 1989.

[25] A. Serjantov, "Anonymizing censorship resistant systems," in *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. London, UK: Springer-Verlag, 2002, pp. 111–120.

[26] A. Beimel, Y. Ishai, E. Kushilevitz, and J.-F. Raymond, "Breaking the o(n1/(2k-1)) barrier for information-theoretic private information retrieval," in *FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 261–270.

[27] B. Chor and N. Gilboa, "Computationally private information retrieval (extended abstract)," in *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM Press, 1997, pp. 304–313.

[28] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *IEEE Symposium on Foundations of Computer Science*, 1995, pp. 41–50. [Online]. Available: citeseer.ist.psu.edu/article/chor95private.html

[29] C. Gentry and Z. Ramzan, "Single-database private information retrieval with constant communication rate." in *ICALP*, 2005, pp. 803–815.

[30] E. Kushilevitz and R. Ostrovsky, "Replication is NOT needed: SINGLE database, computationally-private information retrieval," in *IEEE Symposium on Foundations of Computer Science*, 1997, pp. 364–373. [Online]. Available: citeseer.ist.psu.edu/kushilevitz97replication.html

[31] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. Boca Raton, Florida: CRC Press, 1996.