# Light Weight Security for Parallel Access to Multiple Mirror Sites

Bülent Yener *

## Abstract

Mirror sites approach has been proposed recently for reducing the access delay and providing load balancing in network servers. In the mirror site approach a file, such as a multimedia book, is replicated and dispersed over multiple servers and can be requested in parallel. However, to limit the bandwidth waste, each server maintains not the entire file but only a portion of it.

Current solutions to provide parallel access to multiple servers are based on breaking the file into $b$ "pieces" using Forward Error Correction (FEC) codes or their variants. In such techniques **any** $k \leq b$ pieces are necessary and sufficient to construct the file. Thus, protection of the file from unauthorized access has to be based on encryption. As a result, as the degree of parallelism increases a trade off occurs between the security overhead and access delay.

In this work, we propose new file dispersal and access control protocols to reduce the security overhead significantly. Our protocols are based on the combinatorial techniques which break a file into small pieces in a similar way to FEC code. Thus, at least $k$ pieces are necessary to construct the file. However, in contrast with the previous approaches, not every $k$ pieces are sufficient. Capitalizing on this property this work presents secure dispersal and access protocols that aim to minimize the overhead at the servers.

## 1 Introduction

Increasing demand for distributed information services and applications over the Internet (e.g., software distribution, World Wide Web (WWW), Distributed Interactive Simulations (DIS)) requires new algorithms to reduce server load and access delay. Recently, the *Mirror Site* approach is proposed to maintain multiple copies of a file in the network for (i) load balancing at the servers, (ii) faster response to a user request, and (iii) increasing fault-tolerance. In the mirror site approach a client is provided by a number of servers and chooses a single server for its request. Selection of a server is not trivial and two approaches are proposed for improving the performance: (1) static approach based on statistical information [9], and (2) dynamic approach based on probing to determine the "closest" site to direct the request [4].

In the mirror site model, the delay associated with downloading a file can be further decreased by polling multiple sites in parallel. However, this approach has the potential of causing congestion if the servers send back the entire file. Thus, to prevent congestion, new mechanism are needed to have a server to send only a subset of the packets. One solution would be to have the client instruct each server what a specific subset of the packets to be sent. Of course this approach would require negotiations and may not be scalable due to increased overhead. A better solution is to disperse the file into fixed size *pieces* at each server in a predetermined way. Some redundancy is introduced during the dispersal such that a client can recover the file upon receiving a limited number of pieces. Solutions based on this approach use erasure or Forward Error Correction (FEC) codes. For example, in his landmark paper [7] Rabin introduces an Information Dispersal Algorithm (IDA) which represents the common theme in FEC based approaches. IDA partitions a file of length $F$ into $b$ pieces such that (i) each piece has length $F/m$, (ii) no $m-1$ pieces are sufficient, and (iii) any $m$ pieces are sufficient to reconstruct the file. Most recently, a new parallel access scheme is proposed to increase the download speed from multiple mirror sites [3, 2]. Their work is based on Tornado Codes (TC) [6] which is a relaxation of Forward Error Correction codes (FEC) with reduced complexity.

The FEC based dispersal schemes have the property that *any* $m$ pieces would be sufficient to reconstruct the file. Although it is attractive for high availability, these schemes present a need to protect the file since as the Internet becomes commercialized, protection of the information from unauthorized users is needed. There are several protocols proposed for secure directory access (e.g., X.500 DAP [5] and LDAP [10]). In the event of introducing mirror sites, security complexity increases, since if client polls $r$ servers then $r$ authorizations and authentications must be performed for a single file. Thus, as the information (file) becomes redundant, replicated, and distributed the number of queries (requests) to be processed by the servers will grow. If each such request comes with a security overhead, then servers may become bottlenecks.

In this work we investigate a new approach to dispersal and parallel access of a file which aims to reduce the cryptographic overhead. Similar to the FEC based approaches, we perceive the file as a collection of a fixed number of *elements*. An element of the file is

*Information Sciences Research Center Bell Laboratories, Lucent Technologies 600 Mountain Avenue, Room 2T-314 Murray Hill, NJ 07974, email: yener@research.bell-labs.com, phone: (908) 582 7087 fax: (908) 582 1239.



0-7695-0722-0/00 $10.00 © 2000 IEEE

a logical unit of information. However, our approach is a combinatorial one and provides the following properties: (i) no $m-1$ pieces are sufficient and $m$ pieces are needed, (ii) however, **not any $m$ pieces are sufficient** to recover the file, (precisely there are only $m+1$ combinations which we call **configurations** that can reconstruct the file by using $m$ pieces), and finally (iii) there is no redundancy information within a piece. We design protocols that capitalize on (ii) to reduce the security overhead. In a network where each message is charged a fixed price $\$c$, a *budget-bounded* adversary which has a finite budget of $\$B$ and cannot make infinite number of attempts is considered. It is shown that if the adversary (unauthorized user) does not know the "right" configurations then it has exponentially low probability in the number of pieces by guessing a combination of the blocks to construct the file. As a result paralel access protocols which eleminates server-client authentication are proposed.

This paper is organized as follows: In Section 2 we introduce our model and assumptions. In Section 3 we show how to disperse a file into subsets and present two variants of the combinatorial approach are considered: (1) **cloning-based** which uses *random permutations* of *elements*, and (2) **bibd-based** which uses *deterministic permutations* of the elements. The main difference between these two algorithms is that bibd-based algorithm uses *Balanced Incomplete Block Design* (BIBD) of combinatorial design theory [1]. BIBD ensures that each subset (piece) is unique and elements in a piece have different ordering, while the cloning algorithms allows duplicate pieces. In Section 4 access methods are presented. In Section 5, we discuss the security properties of our schemes. Finally the work is concluded in Section 6.

# 2 Model and Assumptions

We consider a large network which is composed of clusters (domains). Each cluster has $s$ servers and there are total of $S$ servers in the network. Within each cluster, a subset of the servers are associated with a file $f$ which is composed of $|f| = v$ *elements*. For example, if the file contains a multi-media book, each element may correspond to a chapter. In order to provide parallel access in the network, the file is replicated [†]. Let $r$ be the number of copies of file $f$ thus $f$ has a total of $vr$ elements. We assume that $S > vr$ thus not every server has a piece of the file.

[†]The optimal granularity of the information contained in an elemen, and demand based replication and allocation of the file over the network is an optimization problem which will not be address in this paper.

The file is valuable as a whole and obtaining parts of it has no value due to the Quality of Service (QoS) requirement of the application requesting this file (e.g., obtaining some parts of the 9'th Symphony is considered to be worthless from QoS point of view). There is a price (denoted by $\sigma$) for accessing to the file. Furthermore, it is assumed that there is a usage-based charging mechanism associated with accessing to the network. Thus, a network service provider can identify and charge different user traffic flow (e.g., http or ftp) at the network entry points.

## 2.1 Trusted Authority

Each cluster has a Trusted Authority (TA) which is responsible from partition and dispersal of the file to multiple servers. Thus, TA maintains the dispersal and reconstruction information for each file that it manages. We assume that each TA is equipped with tools to perform (i) user authentication and authorization, and (2) charging to users for the file access. (We omit the elaboration on these cryptographic tools and assume that they are well understood security mechanism such as Digital Signatures, MACs [8]).

In order to provide a scalable solution and prevent TAs becoming bottlenecks, we limit the amount of information maintained at each TA. However, in order to increase fault-tolerance, the same file can be managed by more than one TA. Finally, a cluster leader performs routing and forwarding operations in a similar way to a gateway (border router) and can filter out messages.

## 2.2 Clients and Servers

For a given file $f$, a client is called *honest* if it is not corrupted by the adversary. It is called *legitimate* if it has paid for the service. We assume that all legitimate clients are also perfectly honest. Furthermore, a legitimate client does not give out neither any secret nor the file itself to any other client. Thus, we do not worry about second hand sale of the file.

Servers are assumed to be trusted. The identity of servers is not known to the clients and their knowledge is limited to the address of their TAs. For file $f$, a server is called *holder* if it keeps a piece of the file. We assume a generic access mechanism to a server called *polling*. As mentioned before each polling message has a fixed cost $\$c$.

## 2.3 Adversary

Adversary is assumed to be polynomial time bounded. We define two variants of the adversary. The **weak adversary** can corrupt up to $d$ other clients for collaboration. However, the adversary cannot change the set of corrupted clients at each time interval (i.e.,

it is not mobile).Adversary tries to "guess" the servers that keep a copy of the file. We assume that adversary is not adaptive and it makes $d$ guesses at once (i.e., not one guess at a time). The **strong adversary** has eavesdropping capability to all in-out traffic of any node in addition to the powers of the weak adversary explained above.

The adversary has a finite budget $B$ which is the sum of the budgets of $d$ collaborators. Since each message has a cost $c$, the maximum number of tries an adversary can effort is $X$ for $Xc \geq B$. We assume that $X < S$ so that adversary cannot effort trying each server. Adversary's objective is to pay less than the price of the service (i.e., accessing to the file). Adversary tries to "guess" the servers that keep a copy of the file. We assume that adversary is not adaptive and it makes $d$ guesses at once (i.e., not one guess at a time). We assume that this is the only goal of the adversary. Our objective is to ensure a scheme that adversary will not obtain the file less than the price of it. However we note that the adversary is able to obtain *partial* information about the file.

# 3 Combinatorial File Dispersal

In this section we present two combinatorial file dispersal algorithms (CFDs): (i) cloning based, and (ii) bibd-based. These algorithms share the following two main steps: (1) a permutation of the indices of the element for reordering of the information, and (2) assignment (distribution) of elements to blocks (pieces).

The proposed algorithms differ from each other at step 1 where the permutation of the elements of a file is performed. The cloning-based algorithm permutes the elements randomly and uses the same random permutation for each copy of the file. In contrast bibd-based algorihm uses the combinatorial design theory [1] for the permutation of the elements. It uses a unique permutation for each copy of the file and maintains well defined properties between the blocks of different copies.

## 3.1 Cloning Based Dispersal

Let $e_1, e_2, \cdots, e_v$ be the set of elements of a given file $f$. Let $\pi_i$ be a random permutation of the indices of the elements to be used for the dispersal of $i'th$ copy of the file. Since the file is to be replicated to $r$ copies, one can either use the same permutation $\pi$ for each copy or choose a different random permutation $\pi_i$ for each copy $i = 1, 2, \cdots, r$. In cloning algorithm we use the same permutation for each copy.

The algorithm first packs the permutation of the indices into $k$ pieces (blocks). We perceive a block as a one-dimensional array that will contain at least $v/k$ elements. The choice of parameter $k$ depends on the file size and number of site. For example consider a file with 9 elements with indices $0, 1, \cdots, 8$ and set $k = 3$. Permutation $\pi_1 = 0, 4, 7, 2, 3, 5, 1, 6, 8$ indicates that elements 0, 4, 7 will be packed into the first block. The next three indices show the elements for the second block.

Dispersal of file $f$ is an assignment of these $k$ blocks to $k < s$ servers such that a server can get at most one block. Next we present a *deterministic* permutation technique using combinatorial block designs.

## 3.2 Block Design to File Dispersal

In this section we consider Balanced Incomplete Block Designs (BIBD) [1] for permutation of the elements. A BIBD is a collection of k-element subsets (called blocks) of a v-element set $S$, $k < v$, such that each pair of elements of $S$ occur together in exactly $\lambda$ of the blocks. In a $(v, b, r, k, \lambda)$ BIBD with $b$ blocks and $v$ elements, each element occurs in $r$ blocks where $bk = vr$; $\lambda(v-1) = r(k-1)$. As a short hand notation a BIBD can be represented with parameters $(v, k, \lambda)$.

In particular, we use block designs that are *resolvable*. A BIBD is resolvable if its blocks can be arranged into $r$ groups (called parallel classes) so that $\frac{b}{r} = \frac{v}{k}$ blocks of each group are disjoint and contain in their union each element exactly once. Resolvable BIBDs considered in this work are the ones with parameters $(n^2, n, 1)$. For example, in Figure 1 we show a BIBD with parameters (9,3,1) and its parallel classes [‡].

Parameter $v$ of a resolvable BIBD is associated with the number of elements in the file while parameter $k$ of the BIBD is the minimum number of pieces necessary to construct the file. Each parallel class of a resolvable design provides first a permutation of the indices of the elements and then packing them into blocks $k$ blocks each with $k$ elements. The order of the blocks within a parallel class is significant since it gives a different permutation of the elements. For example in Figure 1 the second parallel class which has the blocks $B_3, B_4, B_{11}$ will induce permutation $\pi_1 = 0, 4, 7, 2, 3, 5, 1, 6, 8$ if the blocks are ordered as $B_3, B_{11}, B4$, and permutation $\pi_2 = 1, 6, 8, 0, 4, 7, 2, 3, 5$ if the ordering is $B_4, B_3, B_{11}$.

Block design based dispersal can be perceived as a

[‡]Note that there are standard methods for actual construction of a BIBD based on finite fields. For example a (13,4,1) design can be constructed by 0,1,3,9(mod 13), PG(2,3). A resolvable design with parameters (9,3,1) can be obtained by deleting a block of the (13, 4, 1) design. The time complexity of such constructions are linear. There are several methods to scale the block designs (see [11] for scaling of BIBDs to achieve networks of arbitrary size).

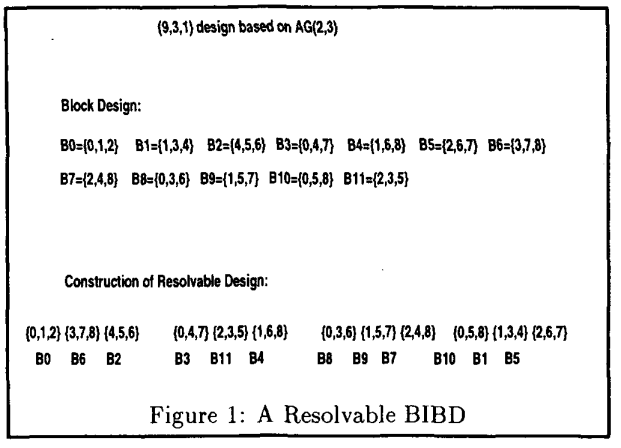


Figure 1: A Resolvable BIBD

special case of cloning algorithm such that each or the $r$ copies of the file have a unique permutation of the elements such that a pair of blocks from two different copies have exactly one common element. In contrast cloning ensure that some pair of inter-copy blocks have empty intersection. If the intersection is not empty then the cardinality is at least $k$. We will use a parallel class (PC) and a copy of the file interchangeably.

**Theorem 1** *Bibd-based CFD algorithm breaks a file with $n^2$ elements into $n^2 + n$ pieces such that*
*(i) any $n-1$ pieces are not sufficient for reconstruction*
*(ii) there are exactly $n+1$ ways for reconstruction by using only $n$ pieces.*

### 3.3 Combinatorial Key

A **combinatorial key** is used for dispersal and construction of each copy of the file. The key includes the following information: (1) FID, (2) v, (3) IP address of each mirror site (server), (4) element index in the block held by this mirror site. For example the following combinatorial key: <FID, SYMkey, 9, [$SID_1$, (0, 4, 7)], [$SID_2$, (2, 3, 5)], [$SID_3$, (1, 6, 8)] > indicates that file identified by FID has 9 elements and the server $SID_1$ contains a block which has the elements 0, 4, and 7. SYMkey for *symmetric* encryption is used only in case of strong adversary.

Servers have **junk blocks** which are composed of random information. For example a junk block my contain text, picked randomly, from several books or video clips mixed together. A junk block is used to send "false" information back to the adversary which polls the server for a file that the server is not a holder. Since we assume a usage-based charging scheme, the cost of junk blocks (e.g., increased network load) is billed to the adversary.

## 4 Access Protocols

This section considers two problems for obtaining a copy of the file. First, enough pieces should be obtained. Second, contents of the pieces must be sorted to obtain correct information.

### 4.1 Client-TA Communication

Client and TA performs authentication check using standard methods (i.e., MAC, digital signatures or appended authenticator) on each message that is exchanged between them. This is necessary to ensure that sender of the message is not the adversary (i.e., impersonation attack).

Two messages are exchanged: *request* (REQ) and *confirm* (CONF). The REQ message contains the < *Address(C)*, $K_C$, *FileName*, *PaymentInfo* > where $K_C$ is the public key of $C$, and the *PaymentInfo* is payment (e.g., credit card) information. Implicit is the assumption that the client knows the name and the price of the file. The CONF message contains combinatorial key of the file. In case of weak adversary (i.e., no eavesdropping) both messages can be in clear text as long as authentication is ensured. However, to protect against to a strong adversary, we adapt an asymmetric-key based scheme in addition to authentication. We assume that the public key ($K_{TA}$) of the TA is known to the clients. Client $C$ sends a REQ (request) message by encrypting it using the $K_{TA}$ : $E_{K_{TA}}(REQ) = eREQ$. The TA decrypts the message using its private key and then sends back a confirmation (CONF) message which is encrypted by $K_C$: $E_{K_C}(CONF) = eCONF$.

### 4.2 Client-Server Communication

Upon receiving the CONF message, a legitimate client $C$ contacts with each server in the combinatorial key. There are two special messages exchanged between a server and the client: *polling* (POLL) and *reply* (REP) messages.

The POLL message is sent by the client and contains the FID, and the client's return address. The REP message is a reply to the polling request by the server. We assume that the server's authentication is implicit from the REP message using appended authentication. The necessary information can be passed to the client with the CONF message from the server. REP can contain either a junk block (if the server is not a holder), or the block associated with the FID (if the server is a holder). Thus, a server always returns a reply message (REP) back to *any* client request. However, if the request is made for a file that the server is not a holder then a "false" reply message is returned. The motivation behind sending junk blocks is to force the adversary to process and to distinguish false replies from

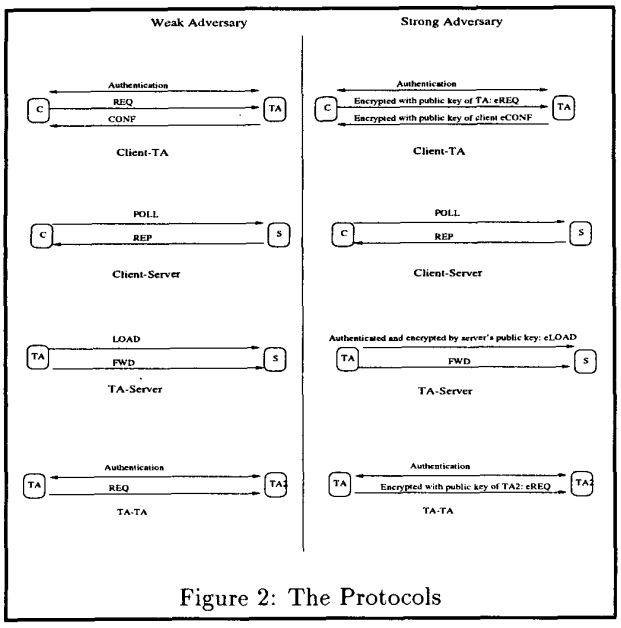


Figure 2: The Protocols

the valid ones. Note that the mirror site server does not perform any authorization and authentication check on the client POLLs. Furthermore, both POLL and REP messages are send in the clear for both adversary models.

### 4.3 TA-Server Communication

There are two messages used for this communication: *download* message (LOAD) and *forward* message (FWD) both sent by the TA. The LOAD message is used to distribute the pieces (block) to the servers. The FWD message is used to have a server transmit its blocks to a specified client. Motivation for FWD is to further reduce the delay by the TA and in this case the CONF does not contain the addresses of the servers. Upon sending a CONF message to a legitimate client, the TA polls each server associated with the requested file. The FWD message includes the following: (1) client's address, and (2) FID. Each server upon receiving such a FWD message sends the blocks associated with the FID. The FWD message is transmitted in clear in both adversary models while the LOAD message is secured using the public key of the server against to strong adversary in addition to authentication. Furthermore in strong adversary model the pieces in the LOAD message are encrypted using a symmetric key which is passes to a honest client by the CONF message.

### 4.4 TA-TA Communication

In our model each cluster leader behaves as a TA and for each file in its cluster it maintains a directory with the following information: (1) file identifier (FID), (2) combinatorial key for the file (3) list of servers holding the blocks of the parallel class. Since not every cluster has a copy of the requested file, a TA needs to check with the other TAs to determine which cluster has a copy of the file. Thus, upon receiving a REQ message, the TA checks if the file is in the cluster. If the request is for a remote file then the TA forwards the request hierarchically. We assume that TA-TA communication is also secure. Upon receiving a forwarded REQ message, a TA searches its directory using the FID as a key. If the requested file is found then the combinatorial key and the list of servers is sent back to the forwarding TA.

### Reconstruction of the File

Reconstruction of a file at the client has two passes. First, all the blocks specified in the combinatorial key must be received. Second, the elements within the received blocks must be sorted based on again the combinatorial key. In case of strong adversary, the blocks in the REP message will be decrypted using the symmetric key sent in the CONF message.

## 5 Analysis

The security is based on the combinatorial properties. For a file with $v$ elements, the first challenge is to obtain enough blocks such that the union of the elements in these blocks is $v$. Given the set of blocks, sufficient to construct the file, the second task is to determine the permutation of the elements to obtain the correct sorting of the information. In this section we show that achieving the first task by random polling is too expensive for an adversary with a bounded budged. This task also requires identifying the valid and false which is an additional computational overhead. The second task is a computationally difficult one. The adversary has to determine the permutation $\pi$ associated with that copy to construct the information which has cost $O(v!)$. Furthermore, for some applications adversary cannot be sure which ordering of the elements is the correct one even if it enumerates all of them.

### 5.1 Combinatorial Security

Consider the probability of obtaining the blocks of a file assuming that the adversary's can POLL any server in any cluster. Since the adversary can corrupt and collaborate with $d$ nodes it has the power to POLL $d$ servers in parallel. There are $vr$ pieces of the file and $S$ servers in the network. The minimum number of blocks

to construct $f$ is $k$. Elements in each minimum set must induce one of the $r$ permutations determined by the dispersal algorithm. Let's now compute the probability of obtaining a configuration or a copy of the file by random polling. We map the problem to the following combinatorial one. Suppose in a jar there are $M$ balls such that $K$ of them are green and $S-K$ are blue. We consider choosing $x$ balls without replacement and consider the probability of getting exactly $y$ greens. That is given by

$$P_x(y) = \frac{C(K,y)C(M-K,x-y)}{C(M,x)}. \quad (1)$$

$C(K,y)$ is the number of ways of selecting $y$ green among $K$ green ones. $C(M-K,x-y)$ is the number of ways of selecting $x-y$ red balls and $C(M,x)$ is the number of ways of selecting $x$ balls out of $M$. In this mapping $M = C(S,k)$ is number of selecting $k$ servers without replacement since the minimum number of servers needed for constructing the file is $k$. Out of $M$ (balls) only $r$ of them are green since there are $r$ k-server combinations can construct the file so $K = r$ (i.e., $r$ green balls). Remark here that for FEC based approaches $K = C(vr,k)$ which is much larger than $r$. Let's define the bound $S \leq Nvr$ so that $M = C(Nvr,k)$ and set $k \geq d$. It is sufficient to compute the probability of getting no green balls (i.e., y=0) with $x$ tries which is

$$P_x(0) = \frac{C(M-K,x)}{C(M,x)} \approx (\frac{(M-K)e/x}{Me/x})^x \approx (\frac{M-K}{M})^x. \quad (2)$$

$$P_x(0) \approx (\frac{(Nvre/k)^k - r}{(Nvre/k)^k})^x \approx [1 - r(\frac{k}{Nvre})^k]^x. \quad (3)$$

Bounding $x$ with $cx \leq \sigma + ck$ where $\sigma$ is the price of the file and $c$ is the cost of each polling message.

$$P_x(0) \approx [1 - r(\frac{k}{Nvre})^k]^{k+\sigma} \quad (4)$$

$$P_x(0) \approx (1 - r(k/Nvre)^k)^k . (1 - r(k/Nvre)^k)^\sigma \quad (5)$$

Which will converge to 1 since $k \leq v$ yields always $(k/Nvre) < 1$. Thus, probability of success for the adversary (i.e., $1 - P_x(0)$) decreases accordingly.

**Strong Adversary**

In case of strong adversary, a symmetric-key system needs to be deployed against to eavesdropping attacks. First the TA encrypts the blocks using a symmetric key and LOADs them to the server in encrypted form. Second the CONF message sent to a honest client contains the above symmetric key. Since CONF is done using an asymmetric key system and authentication is performed between TA and the client, the CONF is secured. The client has to first decrypt the blocks and then use the combinatorial key to construct the file.

# 6 Summary

Accessing to multiple mirror sites in parallel increases the number of requests per server as a function of redundancy. If authorization and authentication is necessary to protect the file then the servers may become bottlenecks. This work presented a combinatorial approach to dispersal of the file. In the proposed scheme at least $k$ pieces are necessary but not any $k$ pieces are sufficient to construct the file. Capitalizing on this property we presented access control schemes with reduced security overhead under two adversary models.

# References


[1] I. Anderson. *Combinatorial Designs: Construction Methods.* John Wiley Sons, New York, 1990.

[2] J. W. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel using tornado codes to speed up downloads. *Proc. of INFOCOM'99.*

[3] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. *Proc. of ACM SIGCOMM'98.*

[4] R. L. Carter and M. E. Crovella. Dynamic server selection using bandwidth probing in wide-area networks. *Proc. of IEEE INFOCOM'97*, 1997.

[5] International Organization for Standardization. Information technology osi- the directory. *ISO/IEC9594-3, ISO/IEC9594-8.*

[6] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. *Proc. of ACM STOC'97*, 1997.

[7] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Jrn. ACM*, 36-2:335–348, 1989.

[8] Bruce Scheier. *Applied Cryptography.* John Willey & Son Inc., 1996.

[9] S. Seshan, M. Stemm, and R. Katz. Spand: Shared passive network performance discovery. *Proc. of Usenix Symposium on Internet Technologies and Sysyems'97*, December 1997.

[10] M. Wahl, T. Howes, and S. Kille. Lightweight directory access protocol (v3). *RFC 2251.*

[11] B. Yener, Y. Ofek, and M. Yung. Combinatorial design of congestion-free networks. *IEEE/ACM Transactions on Networking*, 5(6):989–1000, December 1997.