

Secure Logistical Networking in Virtual Organizations

Lukáš Hejtmánek, Luděk Matyska and Michal Procházka

1 Abstract

In this paper, we propose an architecture that extends Logistical Networking to use Grid authentication and authorization services. Our architecture guarantees that user is authenticated to all services included in network storage stack, the authorization granularity is also at the service level and all authorizations can be revoked at any moment by service providers. We also support access policies. These can limit maximum amount of distributed storage space allocated to a user or group of users or they can limit the maximum amount of time the client can keep his data within the distributed storage. Advanced access control to files is supported, administrators can define access conditions. The prototype implementation has been used to evaluate overhead associated with the security enhancement. This overhead is negligible for Auth, Allocate, Load, and Store commands if only capabilities are encrypted, the Copy command has a penalty of some 10ms (rather negligible for large data transfers). When full data encryption is enforced, the Load, Store, and Copy command are bound by the speed of the used AES 128 cipher.

Keywords: IBP, PKI, X509, authentication, authorization

2 Introduction

The demands on the capacity, availability and performance of data storage are ever increasing and the requirements cannot be served with a centralized approach. Distribution of data storage is becoming very important aspect of data processing in many applications. Grids are an example of a distributed computing and data environment, where long term storage and retrieval of data in a distributed matter is the focus of interest, as demonstrated by many projects dealing with e.g., the high energy physics data (DataGrid, DataTag, EGEE, LCG etc.).

Within Grids, users are usually organized within Virtual Organizations (VO) [FKT01]. The VO serves as a unit for user membership and resource

management, including the data management. Different approaches to data management are employed in current Grids, like Storage Elements which communicate with the computing nodes in a stage-in/stage-out manner, and networked and distributed file systems (NFS, AFS, GPFS etc).

The Logistical Networking [BMP02] presents a novel approach to the problem of distributed data management in Grids. The Logistical Networking provides a universal storage fabric that provides a foundation for robust, scalable, and high performance data storage systems that can be tailored to particular needs (e.g., both distributed file systems and storage elements could be built on such a fabric). However, the legacy Logistical Networking is not ready to be used in a Grid environment where proper authentication and authorization are necessary to control access to the stored data.

The goal of the work presented in this paper is to extend the Logistical Networking with the features needed for its incorporation into the VO-based Grid environment. We focus on the security aspects, including the authentication and authorization mechanisms compatible with the security infrastructure used by contemporary Grids. Our approach is based on the PKI infrastructure for authentication and VOMS service [Alf04] for the authorization.

This paper is organized as follows: The Logistical Networking and VOs are briefly introduced in Section 3 and Section 4, respectively. Section 5 covers the abstract design of the proposed security enhancements to the Logistical Networking and Section 6 introduces our preliminary implementation as a proof of concept. Experimental results including performance evaluation are found in Section 7. The related work and conclusion are then covered in Section 8 and Section 9.

3 Logistical Networking

The Logistical Networking has been introduced to provide global network storage and data movement as a first class network service. Its major components are the IBP protocol and the corresponding network storage stack [BMP02], consisting of five layers: the IBP, L-Bone, exNode, LoRS, and application layers.

The IBP layer represents the raw storage layer, storing individual data blocks without any knowledge about the actual data structure. Access to the stored data is controlled by system of capabilities. Only the user possessing appropriate capability is allowed to read or write data blocks. Capabilities are represented by the hostname of the IBP server and random strings generated by the IBP storage server. The capabilities are issued to the client while storing data block. To prevent indefinite storage allocation and resulting capacity exhaustion, only time-limited data storage is usually provided by the IBP servers. The user can request to store data for some limited amount of time only. After expiration of

the time-limit, the storage server may remove the data block at any time. Users can renew the lease of the storage capacity.

The L-Bone layer acts as a register of resources and gathers information about IBP storage servers such as hostname, location, and availability. The client connects the L-Bone server to get hostnames of the IBP storage servers with the data blocks and then communicates directly with the servers.

The exNode layer maps data blocks into files. This mapping the file metadata is represented by an XML file. The metadata contains capabilities to access particular data blocks. On the conceptual level, the metadata is equivalent to the well known UNIX I-node.

The LoRS layer comprises API for storing and retrieving data, and creating and using the metadata. Users can upload, download, and manipulate data on the IBP storage servers using command line utilities that are part of the LoRS layer.

Application developers can access the data on the IBP servers using the `libxio` library that provides a standard UNIX I/O interface [HH03].

The major drawback of usability of the LoRS API and also the `libxio` in a Grid environment lies in the metadata handling. The metadata is always sent to the client which is responsible for its storage. If the metadata is lost or corrupted, the original data is lost, too (at least, the storage capacity is freed when the time limit expires). To overcome this problem, we have introduced a metadata manager that keeps the metadata on behalf of the client and ensures metadata availability. The metadata manager can be seen as an extension of the original network storage stack [Hej06].

4 Virtual Organizations and Logistical Networking

A virtual organization [FKT01] is a set of individuals and/or institutions that agree on sharing and access rules for some of their resources. The sharing is enabled by the appropriate policy and technical means to implement it. In this paper, we deal with sharing of storage resources based on the Logistical Networking. Each IBP depot has an owner with whom the particular VO negotiates the sharing rules. These rules must define sharing per VO (i.e., they must define access rights for VO members in general), and the depot owner may also define additional per-user based sharing rules. (e.g., to restrict access for individuals that misbehaved in the past). The VO general rules are usually based on a VO membership. Both the membership and the per-user individualized rules require some kind of user identification, based on some authentication mechanism. In this paper, we use PKI [Tue04] and VOMS (Virtual Organization Membership Service) [Alf04] to provide the authentication and authorization, respectively.

Virtual organization are usually not static entities, both resource providers and users can join and leave at any time. Also, a particular VO can become part of another VO or several VOs can merge together. The security infrastructure must be able to support even such dynamic behavior.

The IBP depots are registered with the L-Bone server. When dealing with Virtual organizations, the L-Bone sever should register only depots serving a particular VO. This restriction of resources registered with an L-Bone server can be seen as a natural enhancement of the VOMS server. Users, when connecting to the L-Bone, could easily see depots belonging to this VO, and the VO managers can easily open access to their IBP depot registering it to the appropriate VOs L-Bone server.

VOs are using some set of user names and group names for user authorization. These names are local and defined only within particular VO. This means that two distinct VOs can use the same user name or group name with different meaning. Consequently, services relying on these names cannot be directly shared among VOs, making merge of two or more VOs potentially difficult task. The basic authorization service within the Logistical Networking framework is provided by the metadata server. When storing the metadata, we can either use implicit VO identification or we can include the VO identification, making it part of the metadata itself. In the former case, the metadata is local and the authorization information can not be shared among different VOs (as there may be name clashes). In the later case, the metadata is unique (assuming we have a globally unique VO identifier), so sharing the metadata information is rather easy. We decided to use this approach, as it can be usable even if no globally unique VO identifier exists in such a case the merge of two VOs with the same identifier is not allowed (or, more precisely, we have to re-label one of these VOs and then do the merge). This way, we keep the naming system local (within a VO) while allowing two or more VOs to merge without need for (expensive) renaming.

4.1 Data Manipulation Requirements

The distributed storage system for a VO must fulfill the following requirements:

1. User must be authenticated to all services.
2. User must be authorized to use each service.
3. Authorization must be revocable.

Within the Logistical Networking context this means that user must be authenticated to all the metadata manager, L-Bone server, and IBP servers. This is

achieved via a proxy certificate with appropriate VOMS attributes. The metadata manager must provide the metadata to authorized users, the L-Bone server must issue list of IBP servers to authorized users, and the IBP servers serve only to the authorized users. We require that each authorization must be eventually revocable (there is no permanent right to use some service). Instead of using revocation servers (or lists) we rely on time limited authentication and authorization information that must be periodically refreshed.

The access policy is usually defined on a group and user levels. The access policy defines maximum amount of storage space that can be consumed by the client and maximum amount of time for which the client can keep stored data. We also offer advanced access control to files. Administrators can define access conditions such as time limit to the file access (e.g., the condition can limit users or groups to access particular file only for one day), or system load limit (e.g., the condition can limit users or groups to access particular file only if storage server is idle). The owner of a file can bind the file with these usage constraints.

5 Architecture Design

We can look at the whole schema of logistical networking from authentication and authorization point of view as some kind of workflow of services. The metadata manager, the L-Bone, and the IBP servers can poses services. Workflow looks like a string, a user have to be autenticated and authorized with all the services in predefined order. The metadata manager needs for processing the user request his valid credentials. The L-Bone server also requires valid user credentials to serve the user request. Target IBP storage server needs valid metadata from the metadata manager and correctly signed data from the L-Bone server. This implicates that the user must be successfully authenticated and authorized to the metadata manager and the L-Bone server.

Another feature of this schema lies in eliminating communication between user and additional services. When the user is authenticating or his authorization is verified by the metadata manager or the L-Bone or the IBP storage server there is no need to contact any additional services. Auhtentication and authorization is always made localy on the service.

An enhanced version of the Logistical Networking architecture is shown in Figure 1. It includes a certification authority and a VOMS server. While the actual access to data is analogous to the legacy Logistical Networking, at first each user must obtain a (proxy) certificate with VOMS attributes (steps 1 and 2). The user must be authenticated when contacting a service which verifies his/her authorization.

To make the architecture secure, none of the steps 1 through 5 can be bypassed.

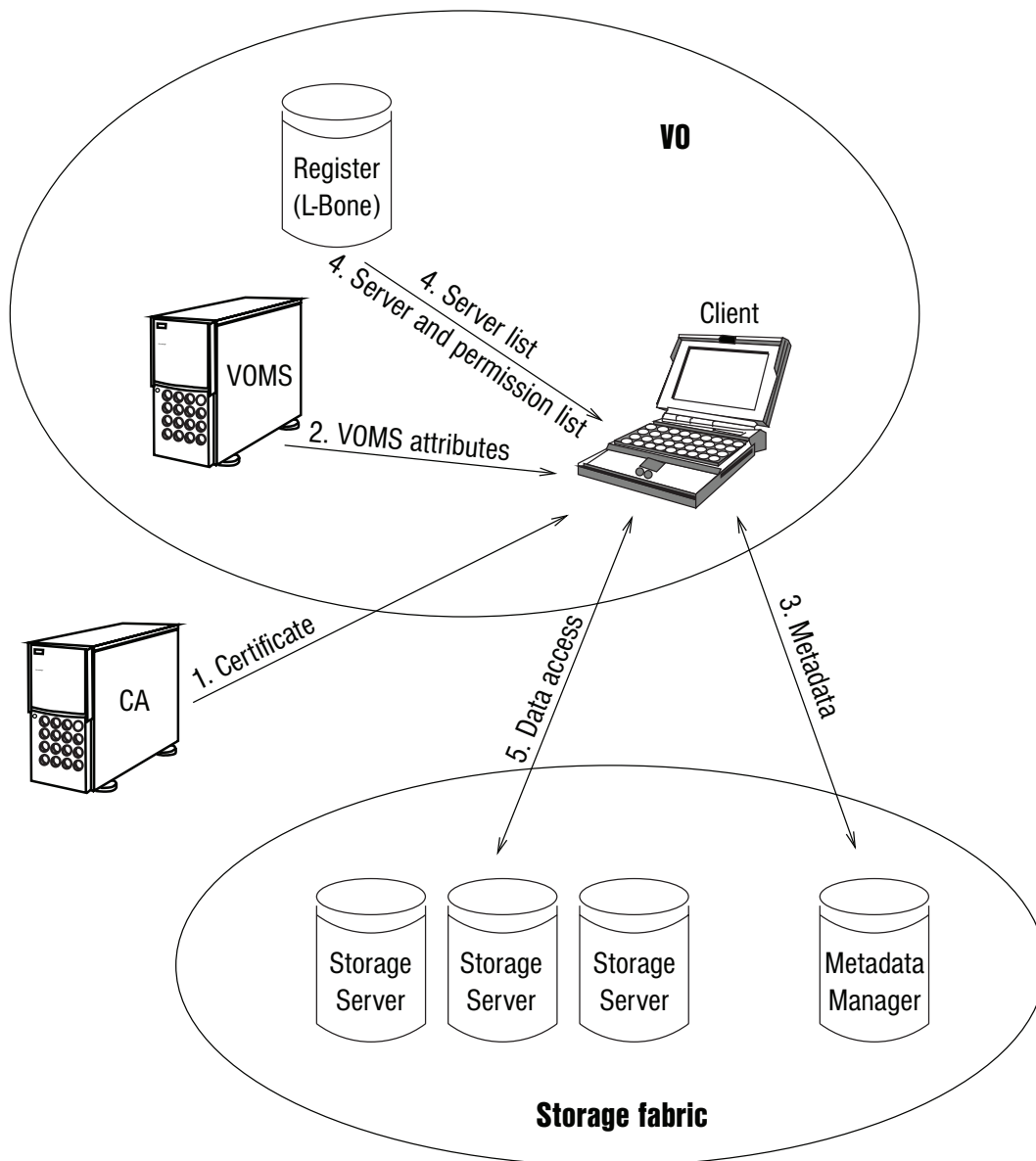


Figure 1: Enhanced schema of Logistical Networking within a VO.

While all the steps except step 3 are direct and they form a continuous sequence protected simply by time stamps, information obtained in the step 3 can be cached by a client. As discussed in Section 5.3, support of revocability of this step needs an enhancement to the time stamp approach.

Our extension of the Logistical Networking thus consists of proper user identification and authorization to each element of the network storage stack.

5.1 User identification

The user identification can not be based directly on the certificates as defined and used by the PKI system. User may possess multiple certificates and even the certificates issued by a particular certificate authority can change in time, so the certificates are not stable enough to provide permanent user identification to the system. Therefore, we have introduced user and group ID attributes that are part of the X509v3 certificate extensions. The user and group IDs are defined to be unique and permanent and are signed as a part of the proxy certificate by a trusted authority (e.g., the VOMS server). When the user approaches the system for the first time, a new unique user ID is generated and associated with the submitted user certificate. Using this initial certificate, user is able to add multiple certificates, associating them all with the same user ID. Thereafter, the user can provide any of his/her certificates to uniquely identify him/herself to the system.

5.2 Authorization Process

The authentication and the authorization has three parts:

1. The user obtains a *proxy certificate* with ID attributes (e.g., VOMS attributes or any other attributes defined within a particular VO). We suppose that the user has a certificate from a trusted certificate authority or a proxy certificate usable for this reason. The generation process of the proxy certificate with the VOMS attributes has two parts. First, the user creates a temporary proxy certificate from the personal certificate (or his proxy certificate), this temporary proxy certificate is used to authenticate with a VOMS server which issues VOMS attributes. Second, a new proxy certificate is created with the issued VOMS attributes. We suppose that the VOMS server issues the VOMS attributes only to authenticated and authorized clients, i.e., clients that are members of the particular VO. The proxy certificate has a limited time span that expires after couple of hours or days, thus making the authorization revocable.
2. The user obtains *metadata*. The metadata manager authorizes the user to access a file by issuing metadata specific to the file. Authorization is

based on the ID attributes from the proxy certificate and on the access control list to the file. In this part, the authentication relies only on the ID attributes in the proxy certificate. Validity of the issued metadata is time-limited. We will discuss this in Section 5.3.

3. The user obtains a *list of storage servers and signed permissions to use storage servers*. In case of data retrieval, the list of storage servers is not needed because the list of storage servers is included in metadata. The resource server issues the list of storage servers and the list of signed permissions to use storage servers to the authorized user (according to the ID attributes in proxy certificate). Validity of the issued signed permissions is also time-limited. The ID attributes from the user's proxy certificate are included in the signature making the signature valid for a particular user only.

The user accessing a storage server must provide the particular piece of metadata and particular item (i.e., the signed permission to use particular depot) from the list of signed permissions to use storage servers. The storage server checks validity of the metadata and the signature of permissions. Thereafter, the client can access stored data. In case of data storing, the user provides only particular item from the list of signed permissions.

We must enforce the users not to bypass any part of the authentication and authorization process, but we can see that this requirement is satisfied as the storage server requires authorization tokens from the register of resources and the metadata manager, and these two services require authorization tokens based on the users certificates issued by a trusted certificate authority. We also demand that the authentication and authorization tokens are revocable, which is provided by the time limited tokens that need to be refreshed and the refreshment may be rejected.

5.3 Time Limited Metadata

Splitting the metadata manager and storage servers into independent services poses security problem. If the metadata manager and storage servers are not online simultaneously, the storage server cannot verify the validity of metadata (i.e., whether access control information has not changed). The access control is not revocable as the client can keep issued metadata forever and thus has access to the data even if the access to metadata has been denied. However, revocable access control is one of the important features required by a VO. Therefore, the metadata manager must issue time limited metadata and the storage server must be able to verify validity of the metadata (i.e., the time limit). Basic idea about presenting time limited metadata is in signing metadata together with a time stamp by the metadata manager.

This problem is isomorphic to the following problem. We are given a book and a librarian. *How the librarian issues us a set of pages together with a signature that is valid only for us and this particular book?* The librarian is unable to put signature on every single page. The librarian cannot prepare the signed pages because the pages belong to our own copy of the book. We require that signature is good enough to prove that any single issued page belongs to the book without verifying the signature using the book or other issued pages. The isomorphism is as follows. Metadata is the book and pages from the book are the data blocks. Below we describe three possible solutions to this problem.

Signing complete metadata.

This approach basically means that the librarian issues a complete book together with pages. However, if we apply such approach to time limited metadata problem, it is unusable. That is because the book is too big as explained below.

Time limited metadata consists of metadata and signature which includes hash of metadata and time stamp. However, the size of metadata is not negligible and it is not feasible to provide metadata and the signature for each data request to the storage server. The client must provide whole metadata to the storage server as the storage server does not know the whole metadata and therefore it cannot guarantee the time limited usage of metadata. E.g., the size of stored file is 5 TB, using 50 MB data blocks, the size of metadata will be about 100 MB, this means that to obtain 50 MB of data, the client must provide 100 MB metadata and the signature.

We could use persistent connections and the client could send the metadata only at the time of the connection establishment. This approach requires the storage server to keep huge metadata and compare each data request whether the request is from issued metadata. This approach also requires one data connection per file, the data connections cannot be reused.

Signing per partes.

Other possibility is to split the book into chapters and issue always a complete chapter together with the pages. If we apply such approach to time limited metadata problem then it is unusable because the book may contain many chapters which leads to overloading the librarian as explained in the next paragraph.

We split metadata into small parts. The smallest possible division is into individual data blocks. However, such a division leads to overload of the metadata manager as it must sign many pieces of the metadata (e.g., making 1000 signatures takes about 8 seconds). However, an optimization is possible, we can compute hash of metadata for particular data blocks and then we can sign only hash values. This significantly reduces size of signed data but its size is still linearly dependent on the size of metadata (e.g., size of metadata for particular

data block is about 1000 bytes, the hash of this metadata can take only 18 bytes). The persistent connections can help also in this case but they have the same limitations as in the previous case.

Relation of data blocks and metadata.

We can build relation between a book and the pages. For example, we can put an ISBN number of the book to each page. Putting an ISBN number to each page differs from signing each page because it is sufficient to put the ISBN number only once for all clients whereas the signature must be done newly for each client. The signature of a book comprises the ISBN number using which we are able to prove that the pages belongs to the book.

We build relation between a data block and the corresponding metadata. This relation can be kept at no extra cost. At the time of creation a new metadata file, the metadata manager generates unique short identification of the metadata file. As the client (the creator of the file) stores data into the storage server, it provides the identification of the metadata to the storage server. The storage server binds the identification and the data persistently. When the client is obtaining metadata from the metadata manager, the metadata manager signs only identification of metadata and time stamp (or some additional usage conditions, such as maximum load of depot) for the whole metadata. The client must provide this signature when accessing any data block from the storage server. The storage server verifies the signature, time stamp and binding data block and identification of metadata. Such signed data is of constant size. Unique identification can be done using identification of metadata manager (e.g., hostname) and increasing sequence starting from zero and being local to a particular metadata manager. Such identification is always unique if increasing sequence per hostname is maintained. Identification does not need to be changed when the file is renamed or moved into different metadata manager.

This approach, however, changes a little bit semantics of storing data blocks in IBP servers. In the new architecture, a relation between the data blocks and the particular file is stored, and this mapping is not explicitly present in the legacy version of the IBP server. This relation gives higher chance to an attacker gaining access to IBP server to reconstruct files as the relation defines which blocks belong to the same file but the attacker does not know the order of data blocks in files (i.e., the attacker does not know offsets of data blocks). If this risk poses a real security problem, the user should encrypt data in an end-to-end manner.

6 Implementation

We suppose that there already exists certification authority that is able to issue X509 certificates and that the user is able to create proxy certificates. Together with certification authority, we use VOMS server to add identification attributes into the proxy certificate.

Implementation of the storage system consists of the resource register, the metadata manager, and storage servers. This system is based on the IBP network storage stack which consists of the L-Bone servers (resource registers), the IBP storage servers (IBP depots), and we have introduced the metadata managers as a part of the network storage stack.

6.1 Metadata Manager

The initial version of the metadata manager described in [Hej06] did not include any authentication and authorization, and thus any client could access any file. The current version already supports the authentication and authorization. Authentication is based on proxy certificates and authorization is based on VOMS attributes. We bind access control information to each file locally, authorization is performed by the metadata manager in user space, so that we do not rely on ACL scheme of the underlying file system. This model resembles approach used in EGEE storage elements [EGEE05]. The metadata manager does not need any other service online to be able to perform authorization which is important feature for robustness reasons. As mentioned above, the metadata manager also binds unique serial number to each file to allow signing the metadata. The serial number comprises identification of the metadata manager and increasing sequence number. We expect the number of concurrent metadata managers is kept low so 8 bits should be enough. Increasing sequence number is 56 bits long which is enough for 2 millions years if one file is created, on average, every 1 millisecond. The serial number is assigned during file creation; the users are enforced to create metadata file on the metadata manager first since they are only allowed to store data to the IBP depots providing the serial number.

6.2 L-Bone Server

We have reimplemented the legacy L-Bone server to support our security model. The clients use proxy certificate with VOMS attributes to authenticate with the L-Bone server within the particular VO. L-Bone checks user's certificate to be valid and performs authorization based on VOMS attributes. The L-Bone server uses its private key to sign a list of IBP storage servers issued to the client. The signature consists of an IBP server name, listening port, time stamp, and VOMS attribute representing user ID. Eavesdropper can steal this signature and also he can also have user's public key, but this is not sufficient to obtain

the capabilities (or data if full data encryption is required) as the symmetric encryption key generated by server is encrypted by user s public key and could not be obtained without the possession of the user s private key.

6.3 IBP Server

We have revamped the legacy IBP server. We have added one new command into the IBP protocol: `IBP_auth` to deal with the user authentication. Also, the IBP server can be configured to require an authentication as the first command in a communication protocol. During the authentication phase, the client supplies his/her certificate and signature of the particular IBP server to which authentication is being performed. The IBP server checks authenticity of the signature (i.e., the server checks whether signature has not expired and that signature belongs to this particular server and authenticating client) and generates random 128 bits key for the AES cipher. This key is sent back to the client encrypted using client s certificate. In the case of a stolen signature and/or certificate, eavesdropper will be unable to decrypt symmetric key and thus will be unable to encrypt and decrypt the capabilities. To make this approach secure, the IBP server must enforce encryption of capabilities if authentication was successful. All legacy commands of IBP protocol were changed so that they encrypt, using the generated symmetric key, all the capabilities strings whenever capabilities are sent over a network. The new IBP server also supports legacy protocol without authentication. A decision whether to encrypt capabilities is based on the availability of a symmetric key. If the key has been generated and sent to the client (i.e., if the `IBP_auth` operation has been successfully performed), all the capabilities are encrypted. Otherwise, they are sent as a plain text over the network. The behavior is configurable, owner of the IBP server can forbid the use of legacy unencrypted protocol completely. While the capability encryption is mandatory in our design, encryption of data transfers can be demanded by users independently for each data transfer. If data transfer encryption is required, data is encrypted using the same symmetric key.

The IBP protocol has an important ability to copy data directly between IBP servers. The user stores data to a particular IBP server (A) then allocates a new disk space on a different IBP server (B) and then requests the IBP server A to copy data directly to the IBP server B. In this case, the user sends read capability of the stored data and write capability of the new storage depot. The IBP server A uses the latter capability and performs a copy.

This basic scenario must be explicitly supported with the delegation of the authentication information, otherwise either the server A will not have the right to write to a pre-allocated space at the server B, or (worse) the server A will have a permanent right to write data to server B, effectively allowing any user to overcome any restrictions on the server B. However, the user may possess

the right to store data to the server B if and only if the user is authenticated and authorized with the server B. The user sends his right to store data on the server B to the server A. We only need to establish secure channel between the server A and the server B. We achieve this by presenting a new command `IBP_getcert` using which the server A can retrieve certificate from the server B. The server A generates symmetric key, encrypts it using a public key from the gained certificate and sends it to the server B. After this procedure, both servers possess symmetric key using which they may encrypt capabilities or data if data encryption was requested.

Legacy IBP library uses persistent TCP connections. This poses a problem to security enhancements as we have no control whether the connection is reused by the same client or by the same file, or whether connection was closed (due to a timeout). We decided to avoid this problem by designing and implementing a new API at the client side of the IBP library. Together with `IBP_auth` command, we implement equivalent secure commands to all legacy commands. The new API requires passing connection context, it means that a user pass his credentials to each IBP command. This approach allows to create a new secure connection when needed and thus persistent TCP connections can be used.

7 Preliminary Experiments

Our preliminary testbed consisted of a single client and a single server. The server has a dual Pentium Xeon@3.0 GHz processor, 4 GB RAM, 1 Gbps NIC and 7 TB SATA HW RAID 5 array, the client is 1.5 GHz Pentium-M, 768 MB RAM, 1 Gbps NIC with a local IDE disk. We are able to achieve 600 Mbps data transfer rate over a single TCP connection measured with the *iperf*¹ tool.

We use simple client calling IBP functions to measure performance impact of the encryption support in the IBP protocol. The calling pattern is as follows: *Allocate* function, *Store* function, *Load* function, new allocation on the same server, and the *Copy* function. This pattern is repeated 400 times. The block size (i.e., the size of a data transfer per called function) has been 1 MB, 5 MB, and 50 MB. We compare legacy IBP implementation, the new version that encrypts only capabilities (Crypto), and the new version that encrypts both capabilities and transferred data (Full crypto).

The timing for the *Auth* operation (which performs the `IBP_auth` command) is presented in Table 1. As this operation is required once per established connection between the client and the server, its duration is constant and is independent of the size of data transfers. *Allocate* function causes penalty about 0.5 ms, this penalty is independent of the size of the allocation and is

¹<http://dast.nlanr.net/Projects/Iperf>

	Auth	Allocate
Legacy	-	0.62 a 0.03
Crypto, Full crypto	14.5 a 1.6	1.10 a 0.04

Table 1: Auth and Allocate duration in milliseconds.

independent of the data encryption (the use of full encryption does not have any influence on its duration). As we can see in Table 2, 0.5 ms penalty is negligible compared to the duration of data transfers.

	Block size	Load	Store	Copy
Legacy	1 MB	22.2 a 0.4	17.7 a 0.1	3.4 a 0.1
Crypto	1 MB	22.4 a 0.3	17.7 a 0.1	13.6 a 0.2
Full crypto	1 MB	87.4 a 0.6	92.6 a 0.3	84.2 a 0.4
Legacy	5 MB	111.6 a 0.8	86.7 a 0.1	11.8 a 0.4
Crypto	5 MB	113.1 a 1.5	86.6 a 0.2	21.6 a 0.3
Full crypto	5 MB	546.3 a 34.8	460.4 a 1.9	376.1 a 1.2
Legacy	50 MB	1106.8 a 3.0	862.9 a 19.1	120.3 a 27.6
Crypto	50 MB	1109.5 a 3.2	864.9 a 22.3	129.2 a 21.6
Full crypto	50 MB	4427.3 a 4.4	4569.0 a 6.3	3630.1 a 9.4

Table 2: Load, Store, and Copy duration in milliseconds.

The timing for the *Load*, *Store*, and *Copy* operations are presented in Table 2. The basic *Load* and *Store* operations need comparable time to finish in the legacy and the new IBP version when only capabilities are encrypted. The new implementation of the *Copy* operations has a 10 ms penalty, caused by one additional round trip due to the certificate exchange, to the generation of a random key, and also to the preparation of the encryption layer. When the full encryption is used, the speed of the AES cipher (we use the *gcrypt*² library) becomes the performance limiting step.

8 Related Work

The IBP protocol has been already extended to work together with the SSL to provide encrypted communication between user and the IBP storage depot. The OpenSSL³ implementations can be used through the LoRS command line utilities. However, SSL support is not used for authentication or authorization, and no interaction with the L-Bone layer occurs.

²<http://www.gnupg.org/>

³<http://www.openssl.org/>

Authentication in the current grid environments is based on the same principle we use the proxy certificates and set of trusted CAs [EGEE05]. However, we differ in the way authorization is performed. Common authorization technique used to access local data in a grid environment is based on a local decision on the resource [EGEE05]. A user obtains authentication credentials (proxy certificate) and authorization credentials (VOMS attributes) and presents them to the resource. A resource has local mapping between the user identity in proxy certificate and local UID/GID. Authorization is then based on local file system permissions. This approach cannot guarantee consistent view on the file permissions when files are distributed across several resources with different UID/GID assignments.

9 Conclusion

We have designed a system for distributed data storage based on the Logistical Networking with security based on certificates and VOMS attributes. This system is suitable for use within Grid VOs and it also supports services provided simultaneously to different VOs. The major feature of our framework is the added security layer, that guarantees that each user is properly authenticated to every element of the network storage infrastructure, that its authorization is checked at each stage, and also that each authorization must be potentially revocable. The security model also anticipates sharing of the storage fabric between different VOs.

We have designed and implemented a prototype implementation and performed preliminary performance evaluation. This evaluation shows that even the prototype implementation exhibits very favorable performance comparable to the legacy Logistical Networking storage stack.

10 Acknowledgments

This research is supported by a research intent Optical Network of National Research and Its New Applications (MŠM 6383917201) and by CESNET Development Fund project 172/2005. We would also like to thank to Petr Holub for stimulating discussions and help with the work described in this paper.

References

- [Alf04] Alfieri R., Cecchini R., Ciaschini V., dell Agnello L., Frohner Á., Gianoli A., Lorente K., and Spataro F.: VOMS, an Authorization System

for Virtual Organizations. *Grid Computing, First European Across Grids Conference*, volume 2970/2004, pages 33–40. Berlin: Springer, 2004.

- [BMP02] Beck M., Moore T., and Planck J. S.: An end-to-end approach to globally scalable network storage. *SIGCOMM Comput. Commun. Rev.*, **32**(4):339–346, 2002.
- [EGEE05] EGEE: *Site Access Control Architecture*. Deliverable DJRA3.2, EGEE project, 2005. Available online⁴.
- [FKT01] Foster I., Kesselman C., and Tuecke S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Int. J. High Perform. Comput. Appl.*, **15**(3):200–222, 2001.
- [Hej06] Hejtmánek L.: Distributed Data Storage with Data Versioning. In G. Krčmářová and P. Sojka (editors): *CESNET Conference 2006*, pages 93–104. Praha: CESNET, 2006.
- [HH03] Hejtmánek L. and Holub P.: *IBP deployment tests and integration with DiDaS project*. Technical report 20/2003⁵, Praha: CESNET, 2003.
- [Tue04] Tuecke S., Welch V., Engert D., Pearlman L., and Thompson M.: *Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile*. RFC 3820⁶, IETF, 2004.

⁴<https://edms.cern.ch/document/523948>

⁵<http://www.cesnet.cz/doc/techzpravy/2003/ibpdidas/>

⁶<http://ietf.org/rfc/rfc3820.txt>