IJITKM *Volume 6 • Number 2 • July-December 2013 pp. 155-164*

Information Security in Grid Computing

Puja Gupta
C.M.J University, Meghallya, Shillong, India
pujababbar.29@rediffmail.com

Abstract — The spread of worldwide networks and the technological trend arc feeding the progress of network and distributed computing in different directions (Grid, Cloud, Autonomic, Ubiquitous, Pervasive, Volunteer, etc). With regard to information, great amount of data widely (geographically) spread over the network require adequate management, to ensure availability for authorized users only, confidentiality and integrity of information and data or, summarizing, security. In order to adequately address security problems such as insider attacks and identity thefts in network-distributed environments, in this work we propose a lightweight cryptography algorithm, combining the strong and highly secure asymmetric cryptography technique with the symmetric cryptography. The algorithm we propose implements a whole secure file system, which preserves and ensures the security of both data and file system structures (directory, links, etc). In the paper we describe in detail the secure distributed file system structure and the algorithms implementing its interface operations. In order to demonstrate the effectiveness of the proposed approach, we also describe its implementation into a Grid (gLite) environment.

I. INTRODUCTION

IT and network technology trends has supported and favored, especially in the last years, the proliferation of distributed computing systems. This stimulus has produced a plenty of network-distributed paradigms, infrastructures and architectures for dealing with the various problems arising from different environments and application fields. Grid, Cloud, Volunteer, Pervasive, Ubiquitous, Autonomic, Ad-Hoc, Green, Edge, Peer to Peer, etc., are different forms of network computing facing different problems in different ways and, sometimes, in different contexts. There are no formal definitions for such kinds of computing and consequently no clear separations among the corresponding contexts. Therefore, from a higher level point a network computing paradigm can be considered as an infrastructure that provides and/or shares network connected resources and services among a (large) population of users. Following this reasoning, problems common to the different network

Sharing resources in network-distributed environments triggers the problem of protecting such resources from malicious accesses and uses. A problem particularly felt by data. Several paradigms of network computing, such as Grid and Cloud, implement adequate resources management's capabilities and ensure security on accessing services and on communicating data, but usually they lack of data protection from direct malicious accesses, at system level. A problem usually underestimated: the access to data must be an exclusive prerogative of only authorized users. In other words, the fact that data are stored in remote nodes, directly accessible from their administrators, constitutes the main risk for data security in such environments, the so called insider abuse/attack problem. Also identity thefts and/or account hijacking problems have to be considered. It is therefore mandatory to introduce an adequate data protection mechanism, which denies data intelligibility to both unauthorized users and administrators.

The aim of this work is to provide such a mechanism capable to store data across network-distributed systems in a secure way and with acceptable performance. In order to protect the access to data, we propose to combine both the symmetric and the asymmetric cryptography. The main contribution of the paper is the specification of such a technique. In order to implement it, we choose to organize the data into a distributed file system, encrypted through symmetric key, also encrypting the file system structure. The proposed technique and the implementation guidelines we specify in the paper, are put into practice in Grid gLite environment, demonstrating the feasibility of the approach.

The remainder of the paper is organized as follows: after a short introduction of background concepts and state of the art in section II, we describe, from a high level point of view, the algorithm (section III), and therefore we provide requirements, specifications and guidelines for effectively implementing such algorithm (section IV). Finally, section V proposes some final remarks and possible future work.

II. STATE OF THE ART AND BACKGROUND

Information or data security helps to ensure privacy and to protect personal data. IEEE defines data security as "the degree to which a collection of data is protected from exposure to accidental or malicious alteration or destruction". The International Standard Organization in 2005 specified information security by the ISO/IEC 27002 standard. It provides best practice recommendations on information security management for use by those who are responsible for initiating, implementing or maintaining Information

Security Management Systems (ISMS). More specifically, information security is defined within the standard as "the preservation of confidentiality (ensuring that information is accessible only to those authorized to have access), integrity (safeguarding the accuracy and completeness of information and processing methods) and availability (ensuring that authorized users have access to information and associated assets when required)".

Several techniques and technologies have been specified in the specific literature in order to achieve data security, that can be classified and summarized into four classes: masking, backup, erasure and encryption. The masking of structured data is the process of obscuring specific data within a database table or cell to ensure that data security is maintained and sensitive customer information is not leaked outside of the authorized environment. Backup techniques refer to making copies of data so that the additional copies may be used to restore the original after a data loss event, in order to improve the reliability/availability of data and, consequently, their integrity. Data erasure is a method that ensures completely destroys of all data in order that no sensitive data is leaked in case of data deletion and/or when an asset is retired or reused. Data encryption refers to encryption technology that encrypts data on storage device. Encryption typically takes form in either software or hardware and it is based on the cryptography theory.

Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication. In recent times it is considered a branch of both mathematics and computer science, and it is affiliated closely with information theory, computer security, and engineering. There are two basic types of cryptography systems: symmetric (also known as conventional or secret key) and asymmetric (public key).

Symmetric ciphers require both the sender and the recipient to have the same key. This key is used by the sender to encrypt the data, and again by the recipient to decrypt the data. The most widely used symmetric cryptography algorithm is the advanced encryption standard (AES) , also known as Rijndael. It is a block cipher adopted as an encryption standard by the U.S. government and developed by two Belgian cryptographers: Joan Daemen and Vincent Rijmen.

With asymmetric ciphers each user has a pair of keys: a public key and a private key. Messages encrypted with one key can only be decrypted by the other key. The public key can be published, while the private key is kept secret. One of the most interesting asymmetric cryptography algorithm is the RSA , developed in 1977 by Ron Rivest, Adi Shamir and Lan Adleman at MIT.

Asymmetric ciphers are much slower, and their key sizes must be much larger than those used with symmetric cipher. At the moment, to break both the AES and the RSA algorithms only the brute force attack is effective, but it requires great power computing and long elaboration time to obtain the key, especially in the latter case.

An interesting technique that combines and synthesizes the high security of asymmetric cryptography algorithms with the efficiency of the symmetric approach is PGP (Pretty Good Privacy). In PGP data are encrypted by using a symmetric cryptography. Then, in order to secure the symmetric key, an asymmetric cryptography algorithm is applied, since this ensures high security. An algorithm similar to PGP has been developed by GNU in the open source project GPG (GNU Privacy Guard).

More specifically, in the literature, the problem of data security in network computing systems has been mainly faced as definition of access right, whilst the coding of the data is demanded to the user, since no automatic mechanism to access to a secure storage space in a transparent way has been defined.

Brunie et al. in studied in depth the problem of data access, and propose a solution based on symmetric keys. In order to prevent non-authorized accesses to the symmetric key the authors propose to subdivide it on different servers. A similar technique has been specified by Shamir in, used in PERROQUET to modify the PARROT middleware by adding an encrypted file manager. The main contribution of such work is that, by applying the proposed algorithm, the (AES) symmetric key, split in N parts, can be recomposed if and only if all the N parts are available.

HYDRA implements a secure data sharing service in gLite 3.0 medical environments, securing data by using the symmetric cryptography and splitting the keys among three keystore servers.

III. THE NETWORK SECURE STORAGE SYSTEM

In this section, we provide a logical description of the approach we propose to face the problem of data security in distributed environment, we named network secure storage system (NS^3). Thus, in subsection III-B, we describe from an high level point of view, the solution we propose to achieve our goal, while subsection III-A details the architecture that puts into practice such solution.

A. The Proposed Solution

As discussed in section II, the most successfully approach adopted for solving the problem of a secure data storage is the symmetric cryptography. The great part of the above mentioned techniques, starting from the symmetric cryptography, implement key splitting algorithms, i.e. the symmetric key (DataKey) is split among different key servers. The underlying idea of the key splitting approach is that at least a subset of the systems (the key servers) over w the keys are distributed will be trustworthy. The scheme parameterizable and, at the end, users can themselves determine how this should be parameterized, and where the key servers managing the splits are located, to obtain a solution that is sufficiently trustworthy to them. However this approach is weak from three points of views: the security, since the list of servers with key parts must be adequately secured, the system administrators can always access the keys and it is really hard to achieve trustworthy on remote and distributed nodes for users, moreover, this technique does not protect the owner of data from whose have (legally or illegally) the privileges of administrator; the reliability / availability, since if one of the server storing a part of the key is unavailable, the data cannot be accessed; the performance, since there is an initial overhead to rebuild a key, depending on the number of parts in which the key is split. A solution for improving reliability/availability is to replicate the key servers, but this contrasts with security challenges. For this reasons we retain it is necessary to use a more effective cryptography algorithm than the simple symmetric one.

An adequate solution for ensuring exclusive data accesses to users could be provided by the asymmetric cryptography, the public key infrastructure (PKI). The only condition to satisfy is that no other than the owner of data can access to the private key, that is the only exclusive way to decrypt data encrypted by the coupling public key.

But, on the other hand, a whole asymmetric cryptography algorithm could have a heavy impact on data access times, since this requires greater computational resources and therefore is slower than the symmetric algorithms. This usually does not allow to encrypt large amounts of data in acceptable time for users, considerably slowing down the overall system's performance. Thus it is necessary to implement a solution representing a trade-off between the two requirements of security and performance.

The one we propose combines both symmetric and asymmetric cryptography into a hierarchical approach. To make the system safe in accessing the symmetric key (DataKey) used in file encryption, this is encrypted by the public key of the user/owner, so that only the user that own the corresponding private key
can decrypt the symmetric key and therefore the data.

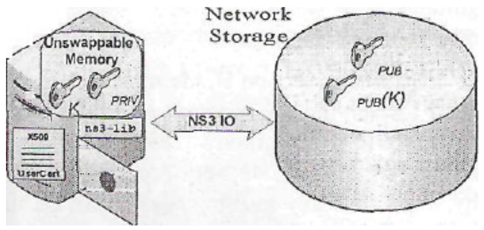


Fig. 1 NS^3 Logic Security Architecture

From an high level point of view such approach is pictorially depicted in Fig. 1. As can be observed, an authorized user, authenticated by his/her own X509 certificate through the user client host, contacts the network storage system where his/her data are located. The data in the distributed storage are encrypted by a symmetric cryptography algorithm whose symmetric DataKey (K) is also stored in the storage system, in its turn encrypted by the user/owner public key KPUB, obtaining the encrypted DataKey Kpub(k).In this way, only the user that has the matching private key Kp RIV can decrypt the symmetric DataKey and therefore the encrypted data.

The encrypted DataKey Kpub(K) is stored together the data in order to allow the user-owner to access them from any node that can access the distributed system. In this way, a user only needs the smartcard containing his/her private key to access the data. Notice that, in the proposed algorithm, the decryption is exclusively performed into the user node where the X509 certificate is stored, and the decrypted symmetric key is placed into an unswappable memory location of such node to avoid malicious accesses.

B, The Algorithm

From an algorithmic point of view, the high-level solution just described can be decomposed into two steps: 1) the symmetric DataKey K is encrypted through the user public key KPUB, and it is written in the distributed storage system; 2) K is ready to be used for data encryption.
The algorithm implementing such mechanism can be better rationalized in three phases; initialization, data I/O, and finalization/termination, detailed in the following subsections.

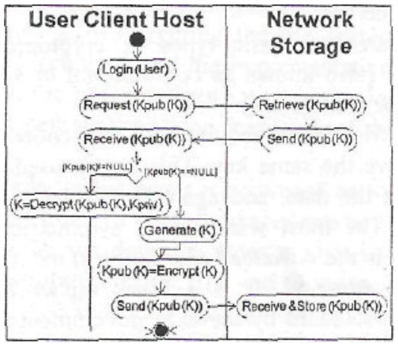


Figure 2. NS^3 Initialization phase algorithm.

1) Initialization: The first phase of the NS^3 algorithm is devoted to the initial setting of the distributed environment. The step by step algorithm describing the initialization phase is reported in the activity diagram of Fig. 2.
Once a user logs in the distributed environment trough the client host, the NS^3 algorithm requests to the storage system the symmetric DataKey K encrypted by the public key of the user KPUB- If the network storage has been already initialized, its answer contains the encrypted DataKey Kpub(K), that is decrypted by the user private key Kp RIV and then saved in a safe memory location of the user interface. Otherwise, at the first access to the storage, a DataKey K must be created by the user interface side of the algorithm and therefore encrypted and sent to the other side.

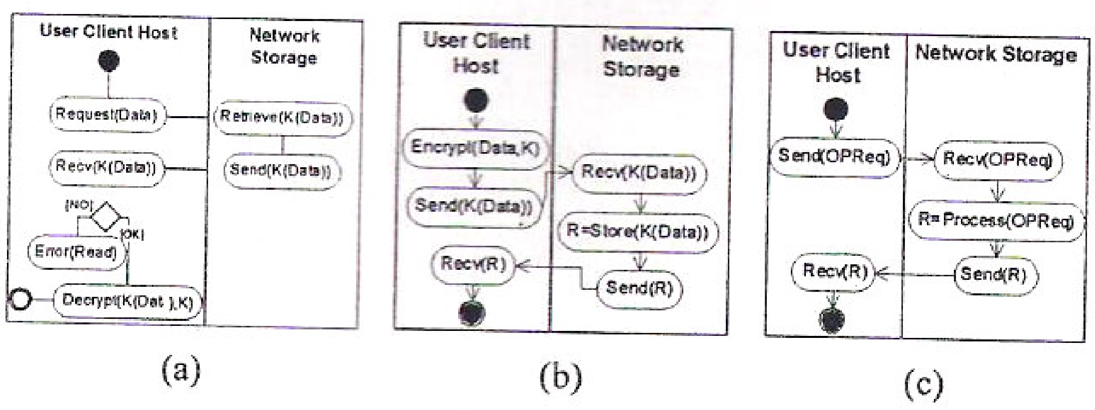


Figure 3. NS^3 data I/O primitives algorithm: read (a), write (b) and generic ops (c).

2) Data I/O: NS^3 organizes the data stored in the network storage system through a file system structured in directories. The data are managed and accessed by well-known primitives such as open, close, read, write, delete, list, rename, etc. In Fig. 3, the algorithms implementing read, write and generic operations (delete, rename, list, etc) are represented by activity diagrams. In particular the read algorithm of Fig.3(a) implies the decryption of data received by the network storage, while the write algorithm of Fig. 3(b) requires the encryption of data before they are sent to the storage system. A generic operation instead only sends a command or a signal, as shown in Fig. 3(c).

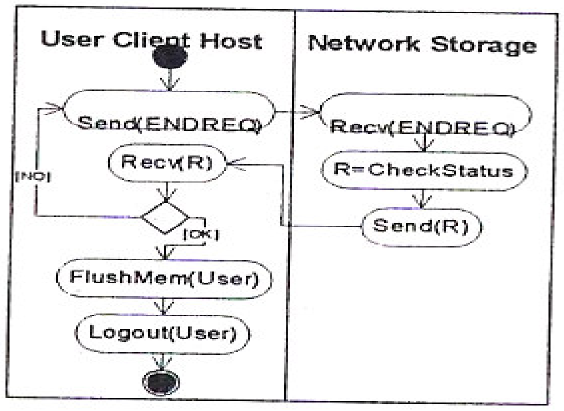


Figure 4. NS^3 termination phase algorithm.

3) Termination: The termination phase algorithm is described by the activity diagram of Fig. 4. Before the user logouts the system, it is necessary to remove the symmetric DataKey and the other reserved information from the user interface memory. But, since a user could still have one or more data I/O operations active/alive, it is possible he/she wants to know the status of such operations, and therefore asks to the network storage system about that. Then, evaluating the obtained answer he/she can choose to terminate the current session or to wait for the completion of some of them. Finally the user logouts the system.

IV. NS^3 IMPLEMENTATION

The idea of combining symmetric and asymmetric cryptography in the data security algorithm detailed in section III, can be really implemented by following the guidelines and the specifications provided in this section.

For the sake of simplicity and portability towards other paradigms a POSIX interface has been implemented for the NS^3 library, as shown in Fig. 5(a). Such idea allows to provide high generality to the NS^3 implementation. The only requirement imposed regards the network storage (NS) interface: in order to match with the actual NS^3 implementation, the NS device must only expose a standard, minimal, interface. Such an interface must be composed of four primitives, that we refer with the generic prefix dev_, implementing the corresponding, generic, NS device: dev_stat (BLK), clev__read (ELK), dev_write (BLK) and dev_unlink (BLK). Such primitives operate on a data block (BLK), checking its status, reading or writing the block or deleting it, respectively. They can be considered as a generalization, in network-distributed architectures, of canonical block device primitives. Therefore, if the considered network storage system provides and implements such four primitives interface, it is eligible to be integrated into the NS^3 file system.
Another minor requirement to take into account in the implementation of the NS^3 technique is that the secure storage should be available in interactive mode from the user client host (UCH). The best way to implement this requirement is to offer NS^3 as a service available from the network (Web service in case of Internet).

A. Storage Architecture

The architecture implementing the NS^3 algorithm satisfying requirements and specifications above described, is depicted in Fig. 5(a). According to this, NS^3 is implemented as a layer working on top of the NS device interface, providing a file service with the security/cryptography capability with a POSIX interface to end-users. With regard to the below layer (the NS device), the main improvement of NS^3 is the implementation of data security. To fulfill this task it is necessary to translate the NS^3 requests, incoming from users through the NS^3 -api interface, to lower level requests, passed on the NS device. This translation is made by the NS^3 -NSDev block, specifically conceived with this aim.
The NS^3 storage service creates a virtual file system structuring the data in files, directories and subdirectories without any restrictions on levels and number of files per directory. Since we build this architecture on top of a NS device, in NS^3 all data objects are seen as files stored into the NS accessible by users through the NS interface. Thanks to storage architecture and internal organization, this N implementation provides all the benefits of a (secure) File system.

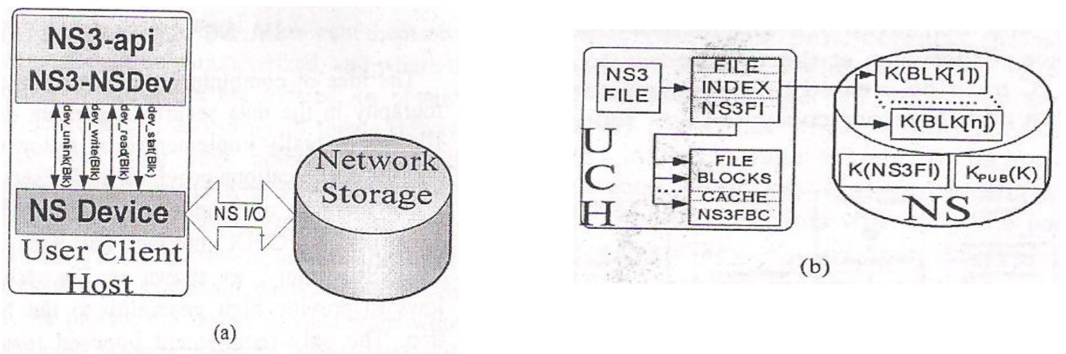


Figure 5. NS^3 gLite Implementation Architecture (a) and Fiie System (b).

An NS^3 file can be entirely stored in the NS in one chunk with variable length or it can be split into two or more blocks with fixed, user defined length, specified in the NS^3 setup configuration, as reported in Fig, 5(b). To avoid conflicts among file names, we univocally identify each chunk of data stored on the NS by a UUID identifier. The file index (NS3FI) shown in Fig. 5(b), maps a file to the corresponding blocks in the NS. Such file index is encrypted through the symmetric DataKey and is kept in the UCH memory. In this way the user operates on a virtual file system whose logic structure usually does not correspond with its physical structure in the NS, since each file can be split into many blocks stored in the NS as files. But the main goal of file indexing is the optimization of the file I/O operations, since it reduces the data access time.

The file system is created and stored in the NS when the NS^3 initialization is performed. Each file referring to data stored in the NS is encrypted by a symmetric DataKey stored in the same NS and encrypted by the user public key.

In order to optimize the file I/O operations performance, a local cache (NS3FBC) of encrypted blocks/chunks is held in the UCH central memory. All the operations involving blocks/chunks already loaded in the UCFI cache are performed locally, varying the content of such blocks/chunks. When a file is closed, the blocks stored in cache are updated to the NS. A specific NS^3 primitive (ns3_flush) has been specified to force the flushing of data from the UCH cache to the NS storage. This remarkably speeds-up the performance of the storage system, reducing the number of accesses to the NS. Problems of cache coherence may arise if a user has more than one simultaneously active login on the network storage system working on the same data. Even if this condition is really uncommon, to avoid the problem we make the assumption a user can have at most only one working session at time.

B. Interface Library and API

Since the NS^3 library commands implement a POSIX.l interface, the access to a file on the virtual encrypted file system is similar to the access to a local file. All the NS^3 functions are prefixed by "ns3_" : ns3_read, ns3_write, ns3_unlink and generally ns3_<op>. The main difference between NS^3 and a POSIX interface is constituted by the initialization and the termination phases as described in section III-B. In the following we specify the NS^3 primitives starting from the same phases characterization above identified.

1) Initialization: The initialization phase is the most important phase of the NS^3 gLite implementation. In this phase the library context is initialized with the user preferences set on environment variables; NS3_PATH (URL base where the data files are stored), NS3_PUBKEY (user's public key used to encrypt), NS3_PRVKEY (user's private key used to encrypt).

A user needing to access the NS must invoke the ns3_init function in order to read from storage space the symmetric DataKey K encrypted by the user public key KPUB- As shown in Fig. 6, and also introduced in subsection III-B1, two cases distinguish the first from successive accesses. In the first initialization phase, ns3_init generates the symmetric key K as sequence of random numbers, returned by an OPENSSL function. In the following accesses, ns3_init loads K and the file index NS3FI from the storage elements. The algorithms in both cases are similar: firstly the UCH checks the presence of the encrypted key Kpub(K) in the NS by a dev_stat, then, in case it does not exist, a new key is created (Fig. 6(a)) and sent to the NS; otherwise the key and the file index are loaded in the UCH by two consecutive dev__read operations (Fig. 6(b)) The encrypted Datakey Kpub(K) is therefore decrypted by the user private key and placed into an unswappable memory location of the UCH to avoid malicious accesses.

2) Data I/O: NS^3 data I/O operations are implemented through I/O POSIX primitives such as: open, read/write and close. Files are always encrypted in memory, the encryption is performed at runtime. To improve the NS^3 performance and the usability of its library the accessed files' chunks are locally buffered into a cache in the UCH until the corresponding files are closed. At file closing, the UCH cache is synchronized with NS.

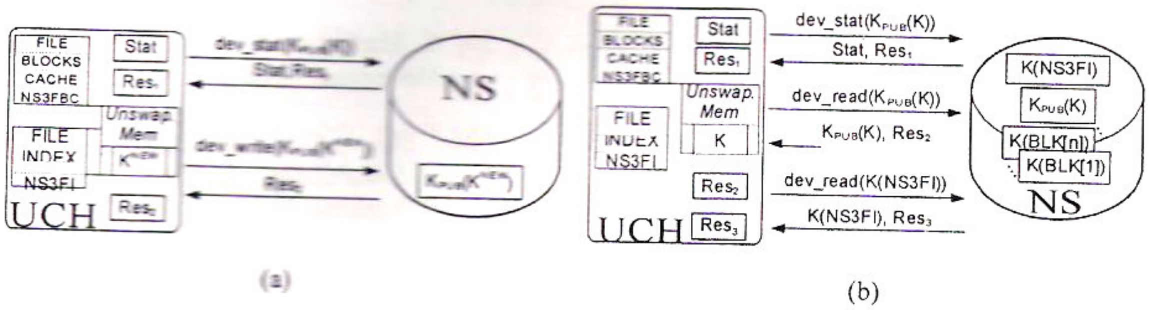


Fig. 6 NS^3 _ Init( ) Library Intialization : First use (a) and following uses (b)

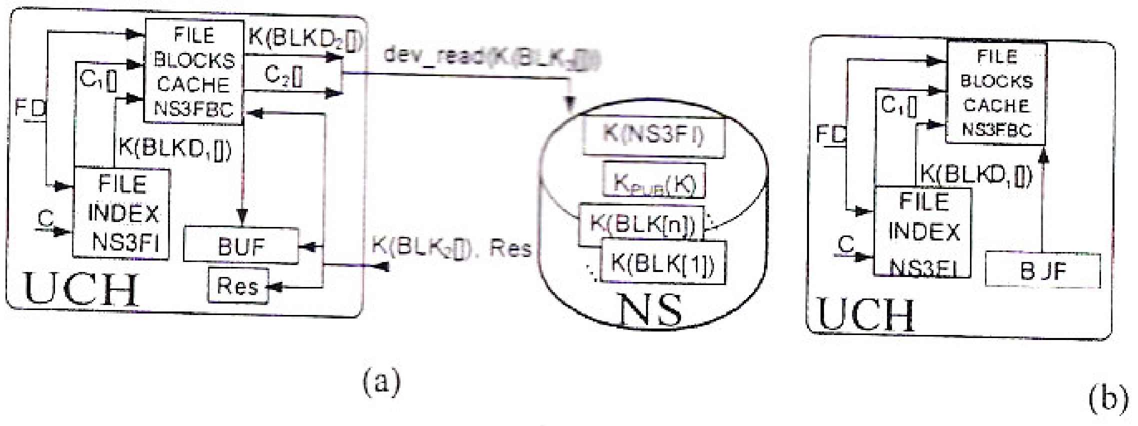
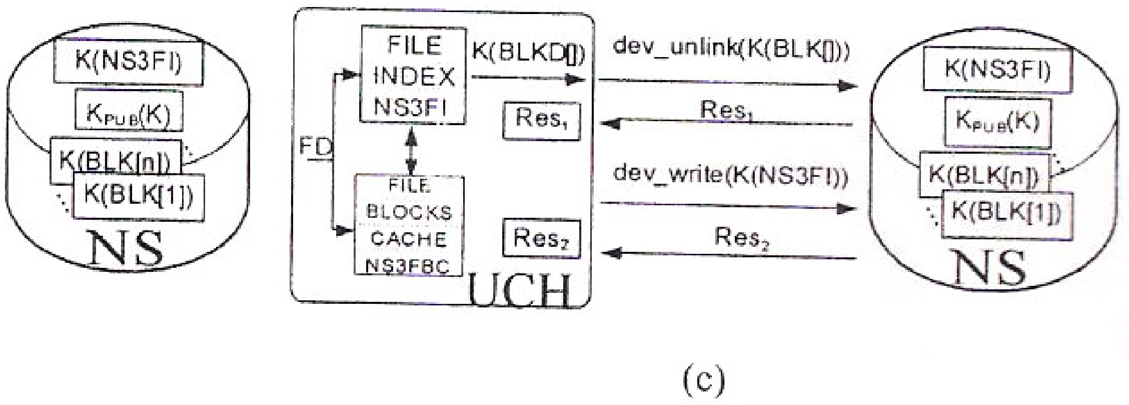


Fig . 7 NS ^{3} Data I / O primitives : NS^3 _ read (a), NS ^{3}_write (b) and NS ^{3}_unlink (c)



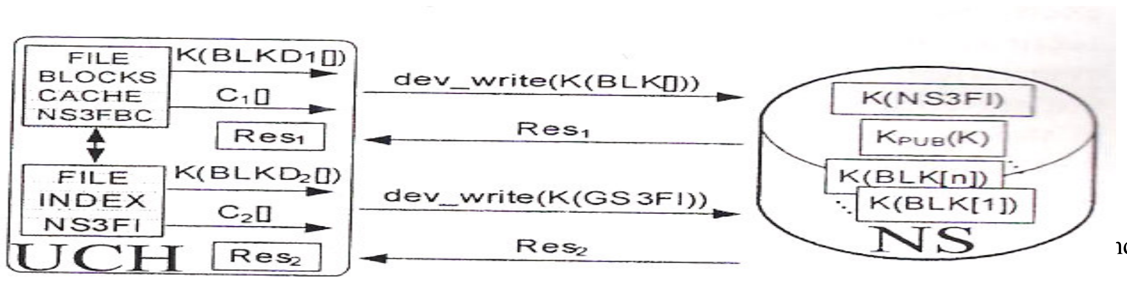
More specifically, the ns3__read (int fd, void *buf, int c) primitive reads c bytes of data of the file referred by the fd file descriptor placing that in the local UCH buffer buf. As pictorially described in Fig. 7(a), by using the file index and the input parameters, the corresponding NS blocks descriptor set (BLKD1) is obtained. The blocks not present in the cache, identified by the set $BLKD_2$ subset $BLKD_1$, are loaded from the NS by a dev_read call. Such data, with the data loaded from cache, are placed in the output buffer, and the file blocks cache NS3FBC is updated with the data just loaded from the NS. The sets BLKD1 and BLKD2 correspond to the vectors BLKD, [ ] and $BLKD_2$ [] of Fig. 7(a).

The ns3_write(int fd, const void *buf, int c) is an operation entirely performed locally to the UCH, as shown in Fig. 7(b). The data blocks to modify in the NS are temporarily saved into the file blocks cache. When the file is closed, renamed, moved, deleted, the flush of the cache is forced, or the NS^3 session is terminated, the data in cache are synchronized with the corresponding one in the NS.

ns3_unlink (int fd) implements a delete operation of the file referred by the file descriptor fd. Since an ns3_unlink (int fd) modifies the file system structure, it is necessary to update the NS^3 file index in the NS each time such operation is invoked.

3) Termination: The main goal of the termination operation is the synchronization of data between the UCH cache and theNS. This is implemented by the ns3_f inalize () function, a simplified version of which is detailed in Fig. 8. It describes two separated dev_write operations into the NS: the former writes all data of the UCH file blocks cache

(NS3FBC), the latter writes the UCH file index NS3FI.



ıderlying

Figure 8. ns3_finalize primitive implementation.

This sequence implements a NS ^{3} _ flush function, called each time a file is closed or deleted. In case the underlying NS devise does not implement the rewriting capability. i.e. a file can be written only when created (as in the GFAL gLite case discussed in next section), the model of Fig. 8 represents only a simplified version of ns3_flush. In such cases, in order to implement this capability in NS^3, it is necessary to bypass the problem of rewriting by deleting and creating a new file each time the file is modified. This mechanism is a little bit complex and hard to pictorially depict, so we only discuss the simplified version reported in Fig. 8. However, the rewriting algorithm has been entirely implemented in the NS^3 library, also taking into consideration this more complex case.

V. CONCLUSIONS

Data security is an open problem in network-distributed computing environments. Protecting data from malicious accesses is the fundamental goal for: companies and enterprises, that try to reduce the loss of information for avoiding industrial espionage; research centers, that have the necessity of hiding their results to competitors; hospitals and medical centers, that must ensure the confidentiality of medical data; etc.

The use of a network-distributed storage allows to remarkably reduce costs from exponentially to linearly with regard to (he growing of the memory ability. Moreover it allows to re-use obsolete resources also with limited computing power.

In this work we described the network secure storage system, a secure (encrypted) storage system for network computing platforms. NS^3 faces the problem of data security in network-distributed environment. In the paper we detail both the NS^3 security algorithm and its implementation.

The security algorithm is based on the idea of combining symmetric and asymmetric cryptography. The symmetric cryptography is directly applied to data, generating encrypted stored data. The symmetric key decrypting such encrypted data is in its turn encrypted by the user public key (asymmetric cryptography), and both the keys are allocated into unswappable locations of the central memory in the user local node/interface. In this way the data can be accessed exclusively by the user creating them. In this context, the security of data is guaranteed to the user wherever and when-ever: when data are in memory, when they are transferred by the network and when they are stored and accessed/managed in the storage system since they are always encrypted.

A strength point of NS^3 is the definition of a specific secure file system on top of the existing network storage library. This choice allows to protect both data/files and also their structure, the whole file system. In particular we implemented the NS^3 technique into the Grid-gLite middleware, by following the implementation guidelines specified in the paper.

A deeper investigation on the NS^3 performance, also considering the impact of cache is one of the imminent future development. Other interesting points to study in depth are: security improvement, cache coherence, fault tolerance and optimization.

REFERENCES

[1] The Authoritative Dictionary of IEEE Standards Terms, 1'h cd., Institute of Electrical and Electronics Engineers, Los Alamitos, CA, USA, 2000.

[2] 1SO/IEC 27002:2005 Standard: Information technology - Security techniques - Code of practice for information security management. International Organization for Standardization (ISO) and International Electroteclinical Commission (IEC), Geneve, Switzerland. 2005.

[3] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, Handbook of Applied Cryptography. Boca Raton, FL, USA: CRC Press, Inc., 1996.

[4] "Federal information, processing strandard pubblicaloin 197." 2001.

[5] R. L. Rivest, A. Shamir, and L. ML Adelman, "A method for obtaning digital signatures and public-key cryptosystems," Commun. ACM, vol. 21, no. 2. pp. 120-126. 1978.

[6] S. Garfinkel, PGP: Pretty Good Privacy. O'Reilly Media, November 1994.

[7] GPG-GNU Privacy Guard - Documentation Sources - GnuPG.org. [Online]. Available: http://www.gnupg.org/ documentation/

[8] L. Junrang, W. Zhaohui, Y. Jianhua, and X. Mingwang, "A secure model for network-attached storage on the grid," in SCC '04: Proceedings of the 2004 IEEE International Conference on Services Computing. Washington, DC, USA: IEEE Computer Society, 2004, pp. 604-608.