International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2015): 6.391

Cloud And Parallel Network File System Using Authenticated Key Exchange Protocols

Anupama T[1], Refeeda K[2]

[1]M Tech in Dept. of Computer Science & Engineering, M.Dasan Institute of Technology, Calicut University, Kerala, India

[2]Assistant Professor in Dept. of Computer Science & Engineering, M.Dasan Institute of Technology, Calicut University, Kerala, India

Abstract: ***Already we studied the issues of key establishment for secure many-to-many communications. The main problem is inspired by the proliferation of large-scale distributed file systems supporting parallel access to multiple storage devices. The system work focuses on the current Internet standard for such file systems, i.e., parallel Network File System (pNFS), which makes use of Kerberos to establish parallel session keys between clients and storage devices. Our review of the existing Kerberos-based protocol shows that it has a number of limitations: (i) a metadata server facilitating key exchange between the clients and the storage devices has heavy workload that restricts the scalability of the protocol; (ii) the protocol does not provide forward secrecy; (iii) the metadata server generates itself all the session keys that are used between the clients and storage devices, and this inherently leads to key escrow. . In this paper, we propose a variety of authenticated key exchange protocols that are designed to address the above issues. We show that our protocols are capable of reducing up to approximately 90% of the workload of the metadata server and concurrently supporting forward secrecy and escrow-freeness. All this requires only a small fraction of increased computation overhead at the client.***

Keywords: Parallel sessions; authenticated key exchange; network file systems; forward secrecy; key escrow.

1. Introduction

In a parallel file system, file data is distributed across multiple storage devices or nodes to allow concurrent access by multiple tasks of a parallel application. This is typically used in large-scale cluster computing that focuses on *high performance* and *reliable* access to large datasets. That is, higher I/O bandwidth is achieved through concurrent access to multiple storage devices within large compute clusters; while data loss is protected through data mirroring using fault-tolerant striping algorithms. Some examples of high performance parallel file systems that are in production use are the IBM General Parallel File System (GPFS) [48], Google File System (GoogleFS) [21], Lustre [35], Parallel Virtual File System (PVFS) [43], and Panasas File System [53]; while there also exist research projects on distributed object storage systems such as Usra Minor [1], Ceph [52], XtreemFS [25], and Gfarm [50]. These are usually required for advanced scientific or data-intensive applications such as, seismic data processing, digital animation studios, computational fluid dynamics, and semiconductor manufacturing. In these environments, hundreds or thousands of file system clients share data and generate very high aggregate I/O load on the file system supporting petabyte- or terabyte-scale storage capacities.

Independent of the development of cluster and high performance computing, the emergence of clouds [5], [37] and the MapReduce programming model [13] has resulted in file systems such as the Hadoop Distributed File System (HDFS) [26], Amazon S3 File System [6], and Cloud-Store [11]. This, in turn, has accelerated the wide-spread use of distributed and parallel computation on large datasets in many organizations. Some notable users of the HDFS include AOL, Apple, eBay, Facebook, Hewlett-Packard, IBM, LinkedIn, Twitter, and Yahoo! [23].

In this work, we investigate the problem of secure many to many communications in large-scale network file systems that support parallel access to multiple storage devices. That is, we consider a communication model where there are a large number of clients (potentially hundreds or thousands) accessing multiple remote and distributed storage devices (which also may scale up to hundreds or thousands) in parallel. Particularly, we focus on how to exchange key materials and establish *parallel secure sessions* between the clients and the storage devices in the parallel Network File System (pNFS) [46]—the current Internet standard—in an efficient and scalable manner. The development of pNFS is driven by Panasas, Netapp, Sun, EMC, IBM, and UMich/CITI, and thus it shares many common features and is compatible with many existing commercial/proprietary network file systems.

- *Scalability* – the metadata server facilitating access requests from a client to multiple storage devices should bear as little workload as possible such that the server will not become a performance bottleneck, but is capable of supporting a very large number of clients;
- *Forward secrecy* – the protocol should guarantee the security of past session keys when the long-term secret key of a client or a storage device is compromised [39]; and
- *Escrow-free* – the metadata server should not learn any information about any session key used by the client and the storage device, provided there is no collusion among them.

The main results of this paper are three new provably secure authenticated key exchange protocols. Our protocols, progressively designed to achieve each of the above properties, demonstrate the trade-offs between efficiency and security. We show that our protocols can reduce the workload of the metadata server by approximately half compared to the current Kerberos-based protocol, while achieving the desired security properties and keeping the

Licensed Under Creative Commons Attribution CC BY

computational overhead at the clients and the storage devices at a reasonably low level. We define an appropriate security model and prove that our protocols are secure in the model.
In the next section, we provide some background on pNFS and describe its existing security mechanisms associated with secure communications between clients and distributed storage devices. Moreover, we identify the limitations of the current Kerberos-based protocol in pNFS for establishing secure channels in parallel. In Section III, we describe the threat model for pNFS and the existing Kerberos-based protocol. In Section IV, we present our protocols that aim to address the current limitations. We then provide formal security analyses of our protocols under an appropriate security model, as well as performance evaluation in Sections VI and VII, respectively. In Section VIII, we describe related work, and finally in Section IX, we conclude and discuss some future work.

2. Internet Standard-NFS

Network File System (NFS) [46] is currently the sole file system standard supported by the Internet Engineering Task Force (IETF). The NFS protocol is a distributed file system protocol originally developed by Sun Microsystems that allows a user on a client computer, which may be diskless, to access files over networks in a manner similar to how local storage is accessed [47]. It is designed to be portable across different machines, operating systems, network architectures, and transport protocols. Such portability is achieved through the use of Remote Procedure Call (RPC) [51] primitives built on top of an eXternal Data Representation (XDR) [15]; with the former providing a procedure-oriented interface to remote services, while the latter providing a common way of representing a set of data types over a network. The NFS protocol has since then evolved into an open standard defined by the IETF Network Working Group [49], [9], [45]. Among the current key features are file system migration and replication, file locking, data caching, delegation (from server to client), and crash recovery.

pNFS separates the file system protocol processing into two parts: metadata processing and data processing. Metadata is information about a file system object, such as its name, location within the namespace, owner, permissions and other attributes. The entity that manages metadata is called a metadata server. On the other hand, regular files' data is striped and stored across storage devices or servers. Data striping occurs in at least two ways: on a file-by-file basis and, within sufficiently large files, on a block-by-block basis. Unlike NFS, a read or write of data managed with pNFS is a direct operation between a client node and the storage system itself. Figure 1 illustrates the conceptual model of pNFS.

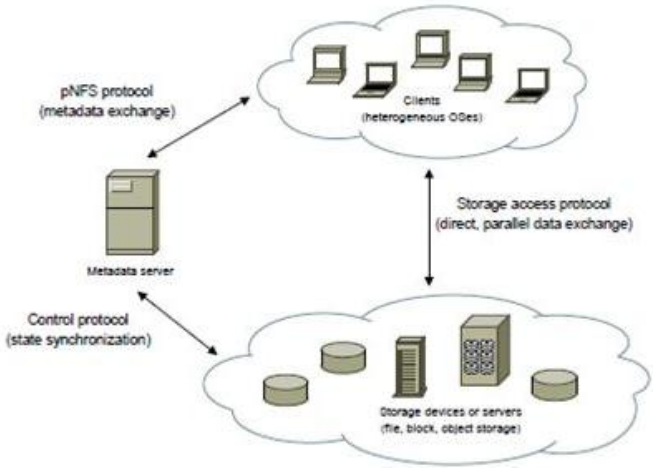


Figure 1: The conceptual model of pNFS.

More specifically, pNFS comprises a collection of three protocols: (i) the *pNFS protocol* that transfers file metadata, also known as a *layout*,1 between the metadata server and a client node; (ii) the *storage access protocol* that specifies how a client accesses data from the associated storage devices according to the corresponding metadata; and (iii) the *control protocol* that synchronizes state between the metadata server and the storage devices.

2.1 Security Consideration

Earlier versions of NFS focused on simplicity and efficiency, and were designed to work well on intranets and local networks. Subsequently, the later versions aim to improve access and performance within the Internet environment. However, security has then become a greater concern. Among many other security issues, user and server authentication within an open, distributed, and cross-domain environment are a complicated matter. Key management can be tedious and expensive, but an important aspect in ensuring security of the system. Moreover, data privacy may be critical in high performance and parallel applications, for example, those associated with biomedical information sharing [28], [44], financial data processing & analysis [20], [34], and drug simulation & discovery [42]. Hence, distributed storage devices pose greater risks to various security threats, such as illegal modification or stealing of data residing on the storage devices, as well as interception of data in transit between different nodes within the system. NFS (since version 4), therefore, has been mandating that implementations support end-to-end authentication, where a user (through a client) mutually authenticates to an NFS server.

2.2 Kerberos & LIPKEY

In NFSv4, the Kerberos version 5 [32], [18] and the Low Infrastructure Public Key (LIPKEY) [14] GSS-API mechanisms are recommended, although other mechanisms may also be specified and used. Kerberos is used particularly for user authentication and single sign-on, while LIPKEY provides an TLS/SSL-like model through the GSS-API, particularly for server authentication in the Internet environment.

Licensed Under Creative Commons Attribution CC BY

User and Server Authentication. Kerberos, a widely deployed network authentication protocol supported by all major operating systems, allows nodes communicating over a no secure network to perform mutual authentication. It works in a client-server model, in which each domain (also known as realm) is governed by a Key Distribution Center (KDC), acting as a server that authenticates and provides ticket-granting services to its users (through their respective clients) within the domain. Each user shares a password with its KDC and a user is authenticated through a password-derived symmetric key known only between the user and the KDC. However, one security weakness of such an authentication method is that it may be susceptible to an off-line password guessing attack, particularly when a weak password is used to derive a key that encrypts a protocol message transmitted between the client and the KDC. Furthermore, Kerberos has strict time requirements, implying that the clocks of the involved hosts must be synchronized with that of the KDC within configured limits.

3. Overview of Our Protocols

We describe our design goals and give some intuition of a variety of pNFS authenticated key exchange6 (pNFS-AKE) protocols that we consider in this work. In these protocols, we focus on parallel session key establishment between a client and *n* different storage devices through a metadata server. Nevertheless, they can be extended straightforwardly to the multi-user setting, *i.e.*, many-to-many communications between clients and storage devices.

3.1 Design Goals

In our solutions, we focus on *efficiency* and *scalability* with respect to the metadata server. That is, our goal is to reduce the workload of the metadata server. On the other hand, the computational and communication overhead for both the client and the storage device should remain reasonably low. More importantly, we would like to meet all these goals while ensuring at least roughly similar security as that of the Kerberos-based protocol shown in Section III-C. In fact, we consider a stronger security model with *forward secrecy* for three of our protocols such that compromise of a long-term secret key of a client *C* or a storage device *Si* will not expose the associated past session keys shared between *C* and *Si*. Further, we would like an *escrow-free* solution, that is, the metadata server does not learn the session key shared between a client and a storage device, unless the server colludes with either one of them.

3.2 Main Idea

Major goals of texture analysis in computer vision area unit to know, model and process texture. To extract the features from image use transforms and textures. In transform mainly used wavelet transform and in texture analysis DCT and GLCM methods are used. In additions to this used area and orientation. Feature extraction involves reducing the number of resources needed to explain an oversized set of knowledge. By using Gray Level Co-occurrence Matrix(GLCM) examining the texture and consider the spatial relationship of pixels and form a matrix [10]. The Discrete Wavelet Transform (DWT) provides necessary information for scrutinity and amalgamation of original signal with a relevant reduction in the computation time.

Recall that in Kerberos-based pNFS, the metadata server is required to generate all service tickets E(KMSi ; IDC; t; ski) and session keys ski between C and Si for all 1< i< n, and thus placing heavy workload on the server. In our solutions, intuitively, C first pre-computes some key materials and forward them to M, which in return, issues the corresponding "authentication tokens" (or service tickets). C can then, when accessing Si (for all i), derive session keys from the precomputed key materials and present the corresponding authentication tokens. Note here, C is not required to compute the key materials before each access request to a storage device, but instead this is done at the beginning of a pre defined validity period v, which may be, for example, a day or week or month. For each request to access one or more storage devices at a specific time t, C then computes a session key from the pre-computed material. This way, the workload of generating session keys is amortized over v for all the clients within the file system. Our three variants of pNFS-AKE protocols can be summarized as follows:

pNFS-AKE-I: Our first protocol can be regarded as a modified version of Kerberos that allows the client to generate its own session keys. That is, the key material used to derive a session key is pre-computed by the client for each *v* and forwarded to the corresponding storage device in the form of an authentication token at time *t* (within *v*). symmetric key encryption is used to protect the confidentiality of secret information used in the protocol. However, the protocol does not provide any forward secrecy. Further, the key escrow issue persists here since the authentication tokens containing key materials for computing session keys are generated by the server.

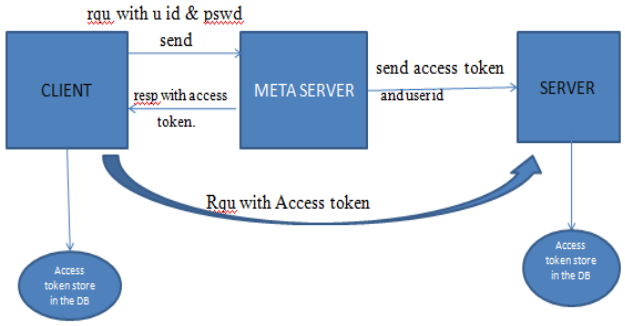
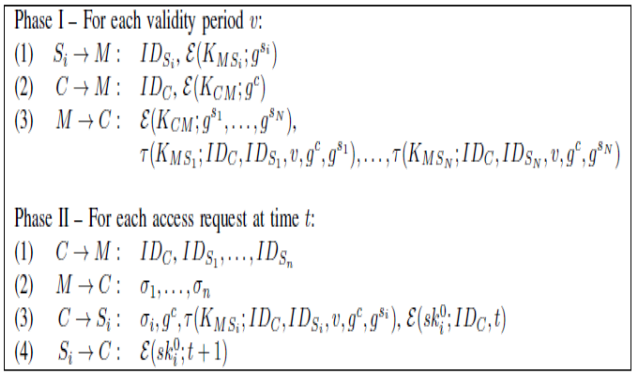


Figure 2: Protocol 1, pNFS-AKE I

pNFS-AKE-II: To address key escrow while achieving forward secrecy simultaneously, we incorporate a Diffie-Hellman key agreement technique into Kerberos-like pNFS-AKE-I. Particularly, the client *C* and the storage device *Si* each now chooses a secret value (that is known only to itself) and pre-computes a Diffie-Hellman key component. A session key is then generated from both the Diffie-Hellman components. Upon expiry of a time period *v*, the secret values and Diffie-Hellman key components are permanently erased, such that in the event when either *C* or *Si* is compromised, the attacker will no longer have access to the

Licensed Under Creative Commons Attribution CC BY

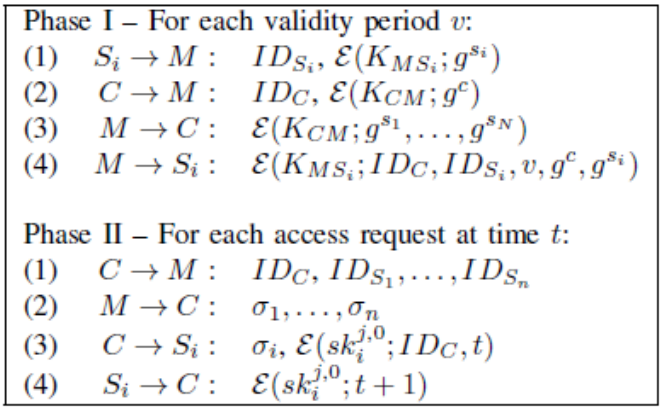
key values required to compute past session keys. However, note that we achieve only *partial* forward secrecy (with respect to *v*), by trading efficiency over security. This implies that compromise of a long-term key can expose session keys generated within the current *v*. However, past session keys in previous (expired) time periods *v′* (for *v′* < *v*) will not be affected.


Phase I – For each validity period v:
(1) $S_i \to M: \ ID_{S_i}, \mathcal{E}(K_{MS_i}; g^{s_i})$
(2) $C \to M: \ ID_C, \mathcal{E}(K_{CM}; g^c)$
(3) $M \to C: \ \mathcal{E}(K_{CM}; g^{s_1}, \ldots, g^{s_N}),$
$\tau(K_{MS_1}; ID_C, ID_{S_1}, v, g^c, g^{s_1}), \ldots, \tau(K_{MS_N}; ID_C, ID_{S_N}, v, g^c, g^{s_N})$

Phase II – For each access request at time t:
(1) $C \to M: \ ID_C, ID_{S_1}, \ldots, ID_{S_n}$
(2) $M \to C: \ \sigma_1, \ldots, \sigma_n$
(3) $C \to S_i: \ \sigma_i, g^c, \tau(K_{MS_i}; ID_C, ID_{S_i}, v, g^c, g^{s_i}), \mathcal{E}(sk_i^0; ID_C, t)$
(4) $S_i \to C: \ \mathcal{E}(sk_i^0; t+1)$

Figure 3: Specification of pNFS-AKE-II (with partial forward secrecy and escrow-free).

pNFS-AKE-III: Our third protocol aims to achieve *full* forward secrecy, that is, exposure of a long-term key affects only a current session key (with respect to *t*), but not all the other past session keys. We would also like to prevent key escrow. In a nutshell, we enhance pNFSAKE- II with a key update technique based on any efficient one-way function, such as a keyed hash function. In Phase I, we require *C* and each *Si* to share some initial key material in the form of a Diffie-Hellman key. In Phase II, the initial shared key is then used to derive session keys in the form of a keyed hash chain. Since a hash value in the chain does not reveal information about its pre-image, the associated session key is forward secure.


Phase I – For each validity period v:
(1) $S_i \to M: \ ID_{S_i}, \mathcal{E}(K_{MS_i}; g^{s_i})$
(2) $C \to M: \ ID_C, \mathcal{E}(K_{CM}; g^c)$
(3) $M \to C: \ \mathcal{E}(K_{CM}; g^{s_1}, \ldots, g^{s_N})$
(4) $M \to S_i: \ \mathcal{E}(K_{MS_i}; ID_C, ID_{S_i}, v, g^c, g^{s_i})$

Phase II – For each access request at time t:
(1) $C \to M: \ ID_C, ID_{S_1}, \ldots, ID_{S_n}$
(2) $M \to C: \ \sigma_1, \ldots, \sigma_n$
(3) $C \to S_i: \ \sigma_i, \mathcal{E}(sk_i^{j,0}; ID_C, t)$
(4) $S_i \to C: \ \mathcal{E}(sk_i^{j,0}; t+1)$

Figure 4: Specification of pNFS-AKE-III (with full forward secrecy and escrowfree).

4. Performance Evaluation

4.1 Computational Overhead

We consider the computational overhead for *w* access requests over time period *v* for a metadata server *M*, a client *C*, and storage devices *Si* for *i 2* [1*;N*]. We assume that a layout is of the form of a MAC, and the computational cost for authenticated symmetric encryption *E* is similar to that for the non-authenticated version *E*.10 Table I gives a comparison between Kerberos-based pNFS and our protocols in terms of the number of cryptographic operations required for executing the protocols over time period *v*. To give a more concrete view, Table II provides some estimation of the total computation times in seconds (s) for each protocol by using the Crypto++ benchmarks obtained on an Intel Core 2 1.83 GHz processor under Windows Vista in 32-bit mode [12]. We choose AES/CBC (128-bit key) for encryption, AES/GCM (128-bit, 64K tables) for authenticated encryption, HMAC(SHA-1) for MAC, and SHA-1 for key derivation. Also, Diffie-Hellman exponentiations are based on DH 1024 bit key pair generation.

4.2 Key Storage

We note that the key storage requirements for KerberospNFS and all our described protocols are roughly similar from the client's perspective. For each access request, the client needs to store *N* or *N* + 1 key materials (either in the form of symmetric keys or Diffie-Hellman components) in their internal states.

However, the key storage requirements for each storage device is higher in pNFS-AKE-III since the storage device has to store some key material for each client in their internal state. This is in contrast to Kerberos-pNFS, pNFS-AKE-I and pNFS-AKE-II that are not required to maintain any client key information.

5. Conclusion

We proposed three advanced authenticated key exchange protocols for cloud and parallel network file system (pNFS). Our protocols offer three appealing advantages over the existing Kerberos-based pNFS protocol. First, the metadata server executing our protocols has much lower workload than that of the Kerberos-based approach. Second, two our protocols provide forward secrecy: one is partially forward secure (with respect to multiple sessions within a time period), while the other is fully forward secure (with respect to a session). Third, we have designed a protocol which not only provides forward secrecy, but is also escrow-free.

References

[1] M. Abd-El-Malek, W.V. Courtright II, C. Cranor, G.R. Ganger, J. Hendricks, A.J. Klosterman, M.P. Mesnier, M. Prasad, B. Salmon, R.R. Sambasivan, S. Sinnamohideen, J.D. Strunk, E. Thereska, M. Wachs, and J.J. Wylie. Ursa Minor: Versatile cluster-based storage. In Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST), pages 59–72. USENIX Association, Dec 2005.
[2] C. Adams. The simple public-key GSS-API mechanism (SPKM). The Internet Engineering Task Force (IETF), RFC 2025, Oct 1996.
[3] A. Adya, W.J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J.R. Douceur, J. Howell, J.R. Lorch, M. Theimer, and R. Wattenhofer. FARSITE: Federated, available, and reliable storage for an incompletely\

Licensed Under Creative Commons Attribution CC BY

trusted environment. In Proceedings of the 5th Symposium on Operating System Design and Implementation (OSDI). USENIX Association, Dec 2002.

[4] M.K. Aguilera, M. Ji, M. Lillibridge, J. MacCormick, E. Oertli, D.G. Andersen, M. Burrows, T. Mann, and C.A. Thekkath. Blocklevel security for network-attached disks. In Proceedings of the 2nd International Conference on File and Storage Technologies (FAST). USENIX Association, Mar 2003.

[5] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. Communications of the ACM, 53(4):50–58. ACM Press, Apr 2010.

[6] Amazon simple storage service (Amazon S3). http://aws.amazon.com/ s3/.

[7] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In Advances in Cryptology – Proceedings of EUROCRYPT, pages 139–155. Springer LNCS 1807, May 2000.

[8] D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short cipher texts and private keys. In Advances in Cryptology – Proceedings of CRYPTO, pages 258–275. Springer LNCS 3621, Aug 2005.

[9] B. Callaghan, B. Pawlowski, and P. Staubach. NFS version 3 protocol specification. The Internet Engineering Task Force (IETF), RFC 1813, Jun 1995.

Author Profile

Anupama T is pursuing her M.Tech degree in Computer Science and Engineering from M.Dasan Institute of Technology, Calicut University, Kerala. She obtained her B.Tech Degree in Computer Science and Engineering from Paavai College of Engineering, Anna University Chennai, in 2014.

Licensed Under Creative Commons Attribution CC BY