Authenticated Key Exchange Protocols for Parallel Network File Systems

[@] Jansi Sophia Mary.C [#] Dhanalakshmi.P [$] Kalaivani.R.E
[@,#,$] Department of Computer Science and Engineering,
Idhaya Engineering College for Women, Chinnasalem, Tamil Nadu, India

Abstract—This paper deals the problem of key establishment for secure many-to-many communications. The problem is inspired by the proliferation of large-scale distributed file systems supporting *parallel access* to multiple storage devices. Our 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 54% 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.

I. INTRODUCTION

In a parallel file system, file data is distributed acrossmultiple 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 accessto multiple storage devices within large compute clusters;while data loss is protected through data mirroring usingfault-tolerant striping algorithms. Some examples of highperformanceparallel file systems that are in production useare the IBM General Parallel File System (GPFS) [48], GoogleFile System (GoogleFS) [21], Lustre [35], Parallel Virtual FileSystem (PVFS) [43], and Panasas File System [53]; whilethere also exist research projects on distributed object storagesystems such as Usra Minor [1], Ceph [52], XtreemFS [25],and Gfarm [50]. These are usually required for advancedscientific or data-intensive applications such as, seismic dataprocessing, digital animation studios, computational fluid dynamics,and semiconductor manufacturing. In these environments,hundreds or thousands of file system clients share dataand generate very high aggregate I/O load on the file systemsupporting petabyte- or terabyte-scale storage capacities.

Independent of the development of cluster and high performance computing, the emergence of clouds [6], [37] and the MapReduce programming model [13] has resultedin 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-spreaduse of distributed and parallel computation on large datasetsin many organizations. Some notable users of the HDFSinclude AOL, Apple, eBay, Facebook, Hewlett-Packard, IBM,LinkedIn, Twitter, and Yahoo! [23].

In this work, we investigate the problem of secure manyto-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 area 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.

Our primary goal in this work is to design efficient and secure authenticated key exchange protocols that meet specificrequirements of pNFS. Particularly, we attempt to meet thefollowing desirable properties, which either have not beensatisfactorily achieved or are not achievable by the currentKerberos-based solution (as described in Section II):

- *Scalability* – the metadata server facilitating access requestsfrom a client to multiple storage devices shouldbear as little workload as possible such that the serverwill not become a performance bottleneck, but is capableof supporting a very large number of clients;
- *Forward secrecy* – the protocol should guarantee thesecurity of past session keys when the long-term secret key of a client or a storage device is compromised [39];
- *Escrow-free* – the metadata server should not learn anyinformation about any session key used by the client andthe storage device, provided there is no collusion amongthem.

The main results of this paper are three new provablysecure authenticated key exchange protocols. Ourprotocols,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 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.

II. 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 filesystem migration and replication, file locking, data caching, delegation (from server to client), and crash recovery.

In recent years, NFS is typically used in environments where performance is a major factor, for example, high-performance Linux clusters. The NFS version 4.1 (NFSv4.1) [46] protocol, the most recent version, provides a feature called *parallel* NFS (pNFS) that allows direct, concurrent client access to multiple storage devices to improve performance and scalability. As described in the NFSv4.1 specification:

When file data for a single NFS server is stored on multiple and/or higher-throughput storage devices (by comparison to the server's throughput capability), the result can be significantly better file access performance.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.

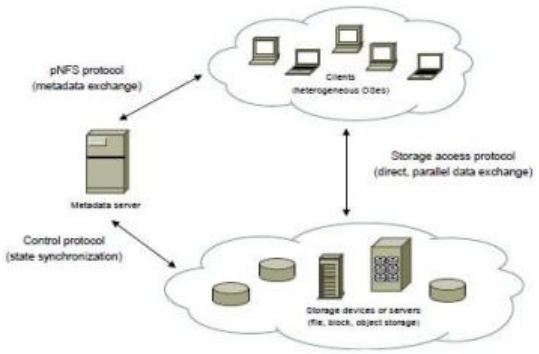

Fig 1The conceptual model of pNFS

More specifically, pNFS comprises a collection of threeprotocols: (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]

A. Security Consideration

Earlier versions of NFS focused on simplicity and efficiency, and were designed to work well on intranets and local networks. Christo Ananth et al. [5] discussed about a method, End-to-end inference to diagnose and repair the data-forwarding failures, our optimization goal to minimize the faults at minimum expected cost of correcting all faulty nodes that cannot properly deliver data. First checking the nodes that has the least checking cost does not minimize the expected costin fault localization. We construct a potential function for identifying the candidate nodes, one of which should be first checked by an optimal strategy. We proposes efficient inferring approach to the node to be checked in large-scale networks**.**Moreover, consideration should be given to the integrity and privacy (confidentiality) of NFS requests and responses [45].

The RPCSEC GSS framework [17], [16] is currently the core security component of NFS that provides asic security services. RPCSEC GSS allows RPC protocols to access the Generic Security Services Application Programming Interface (GSS-API) [33]. The latter is used to facilitate exchange of credentials between a local and a remote communicating parties, for example between a client and a server, in order to establish a security context. The GSS-API achieves these through an interface and a set of generic functions that are independent of the underlying security mechanisms and communication protocols employed by the

communicating parties. Hence, with RPCSEC GSS, various security mechanisms or protocols can be employed to provide services such as, encrypting NFS
traffic and performing integrity check on the entire body of an NFSv4 call.

Similarly, in pNFS, communication between the client and the metadata server are authenticated and protected through RPCSEC GSS. The metadata server grants access permissions (to storage devices) to the client according to pre-defined access control lists (ACLs).3 The client's I/O request to a storage device must include the corresponding valid layout. Otherwise, the I/O request is rejected. In an environment where eavesdropping on the communication between the client and the storage device is of sufficient concern, RPCSEC GSS is used to provide privacy protection [46].

B. 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. User and Server Authentication. Kerberos, a widely deployed network authentication protocol supported by all major
operating systems, allows nodes communicating over a nonsecure 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.

Hence, LIPKEY is used instead to authenticate the client with a password and the metadata server with a public key certificate, and to establish a secure channel between the client and the server. LIPKEY leverages the existing Simple Public- Key Mechanism (SPKM) [2] and is specified as an GSSAPI mechanism layered above SPKM, which in turn, allows both unilateral and mutual authentication to be accomplished without the use of secure time-stamps. Through LIPKEY, analogous to a typical TLS deployment scenario that consists of a client with no public key certificate accessing a server with a public key certificate, the client in NFS [14]:

- obtains the metadata server's certificate;
- verifies that it was signed by a trusted CertificationAuthority (CA);
- generates a random session symmetric key;
- encrypts the session key with the metadata server's public key; and
- sends the encrypted session key to the server.

At this point, the client and the authenticated metadata server have set up a secure channel. The client can then provide a user name and a password to the server for user authentication.

Single Sign-on. In NFS/pNFS that employs Kerberos, each storage device shares a (long-term) symmetric key with the metadata server (which acts as the KDC). Kerberos then allows the client to perform single sign-on, such that the client is authenticated once to the KDC for a fixed period of time but may be allowed access to multiple storage devices governed by the KDC within that period. This can be summarized in three rounds of communication between the client, the metadata server, and the storage devices as follows:

1) the client and the metadata server perform mutual authentication through LIPKEY (as described before), and the server issues a ticket-granting ticket (TGT) to the client upon successful authentication;
2) the client forwards the TGT to a ticket-granting server(TGS), typically the same entity as the KDC, in orderto obtain one or more service tickets (each containinga session key for access to a storage device), and validlayouts (each presenting valid access permissions to astorage device according to the ACLs);
3) the client finally presents the service tickets and layoutsto the corresponding storage devices to get access to thestored data objects or files.

We describe the above Kerberos-based key establishmentprotocol in more detail in Section III-C.

Secure storage access. The session key generated by theticket-granting server (metadata server) for a client and astorage device during single sign-on can then be used in thestorage access protocol. It protects the integrity and privacyof data transmitted between the client and the storage device.Clearly, the session key and the associated layout are validonly within the granted validity period.

C. Current Limitations

The current design of NFS/pNFS focuses on *interoperability*, instead of efficiency and scalability, of various mechanisms to provide basic security. Moreover, key establishment between a client and multiple storage devices in pNFS are based on those for NFS, that is, they are not designed specifically for parallel communications. Hence, the metadata server is not only responsible for processing access requests to storage devices (by granting valid layouts to authenticated and authorized clients), but also required to generate all the corresponding session keys that the client needs to communicate securely with the

storage devices to which it has been granted access. Consequently, the metadata server may become a performance bottleneck for the file system. Moreover, such protocol design leads to key escrow. Hence, in principle, the server can learn all information transmitted between a client and a storage device. This, in turn, makes the server an attractive target for attackers.

Another drawback of the current approach is that past session keys can be exposed if a storage device's long-term keyshared with the metadata server is compromised. We believe
that this is a realistic threat since a large-scale file system mayhave thousands of geographically distributed storage devices.It may not be feasible to provide strong physical security andnetwork protection for all the storage devices.

$$
\begin{array}{lll}
(1) & C \rightarrow M: & ID_C \\
(2) & M \rightarrow C: & E(K_C; K_{CT}),\ E(K_T; ID_C, t, K_{CT}) \\
(3) & C \rightarrow T: & ID_{S_1}, \ldots, ID_{S_n},\ E(K_T; ID_C, t, K_{CT}),\ E(K_{CT}; ID_C, t) \\
(4) & T \rightarrow C: & \sigma_1, \ldots, \sigma_n,\ E(K_{MS_1}; ID_C, t, sk_1), \ldots, E(K_{MS_n}; ID_C, t, sk_n),\ E(K_{CT}; sk_1, \ldots, sk_n) \\
(5) & C \rightarrow S_i: & \sigma_i, E(K_{MS_i}; ID_C, t, sk_i),\ E(sk_i; ID_C, t) \\
(6) & S_i \rightarrow C: & E(sk_i; t+1)
\end{array}
$$

Fig 2 A Simplified version of the KerberOS based pNFS protocol.

III. 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.

A. 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 S_i will not expose the associated past session keys shared between *C* and S_i. 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.

B. Main Ideas

Recall that in Kerberos-based pNFS, the metadata server is required to generate all service tickets $E(K_{MS_i}; ID_C; t; sk_i)$ and session keys sk_i between *C* and S_i 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 S_i (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 amodified version of Kerberos that allows the client togenerate its own session keys. That is, the key materialused to derive a session key is pre-computed by theclient foreach *v* and forwarded to the correspondingstorage device in the form of an authentication tokenat time *t* (within *v*). As with Kerberos, symmetric keyencryption is used to protect the confidentiality of secretinformation used in the protocol. However, the protocoldoes not provide any forward secrecy. Further, the keyescrow issue persists here since the authentication tokenscontaining key materials for computing session keys aregenerated by the server.

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 deviceS_i 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 S_i is compromised, the attacker will no longer have access

to the 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.

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), butnot 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 S_i 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.

IV. Description of Our Protocols

We first introduce some notation required for our protocols. Let $F(k;m)$ denote a secure key derivation function that takes as input a secret key k and some auxiliary information m, and outputs another key. Let *sid*denote a session identifier which can be used to uniquely name the ensuing session. Let also N be the total number of storage devices to which a client is allowed to access. We are now ready to describe the
construction of our protocols.

A. pNFS-AKE-I

Our first pNFS-AKE protocol is illustrated in Figure 3. For each validity period v, C must first pre-compute a set of keymaterialsK_{CS1} ; : : : ;K_{CSN} before it can access any of the N storage device S_i (for 1 _ i _ N). The key materials are transmitted to M. We assume that the communication between C and$M$is authenticated and protected through a secure channel associated with key K_{CM} established using the existing methods as described in Section II-B. M then issues an authentication token of the form $E(K_{MSi}; ID_C; ID_{Si}; v;K_{CSi})$ for each key material if the associated storage device S has not been revoked.[7] This completes Phase I of the protocol. From this point onwards, any request from C to access S_i is considered part of Phase II of the protocol until v expires.

When C submits an access request to M, the request contains all the identities of storage devices S_i for 1 _ i _ n _ Nthat C wishes to access. For each S_i, M issues a layout $\__i$. C then forwards the respective layouts, authentication tokens (from Phase I), and encrypted messages of the form $E(sk^0_i ; ID_C; t)$ to all n storage devices.

Upon receiving an I/O request for a file object from C, each S_i performs the following:

1) check if the layout $__i$is valid;
2) decrypt the authentication token and recover key K_{CSi};
3) compute keys $sk_{zi}= F(K_{CSi}; ID_C; ID_{Si}; v; sid; z)$ for$z = 0; 1$;
4) decrypt the encrypted message, check if ID_C matches the identity of C and if t is within the current validity period v;
5) if all previous checks pass, S_i replies C with a key confirmation message using key $sk_{0\,i}$.

At the end of the protocol, $sk_{1\,i}$is set to be the session key for securing communication between C and S_i. We note that, as suggested in [7], *sid*in our protocol is uniquely generated for each session at the application layer, for example through the GSS-API.

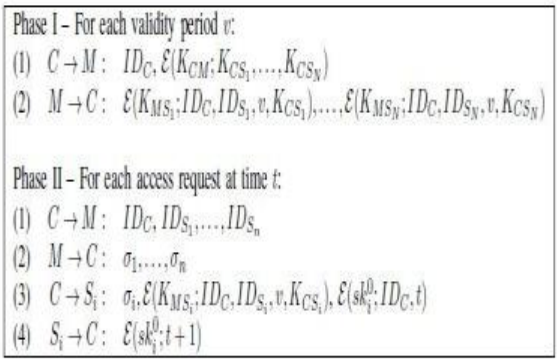

Fig 3 Specification of pNFS- AKE-I

B. pNFS-AKE-II

We now employ a Diffie-Hellman key agreement technique to both provide forward secrecy and prevent key escrow. In this protocol, each S_i is required to pre-distribute some key material to M at Phase I of the protocol. Let g_x2 G denote a Diffie-Hellman component, where G is an appropriate group generated by g, and x is a number randomly chosen by entity X 2 $fC; Sg$. Let _ $(k;m)$ denote a secure MAC scheme that takes as input a secret key k and a target message m, and output a MAC tag. Our partially forward secure protocol is specified in Figure 4. At the beginning of each v, each S_i that is governed by M generates a Diffie-Hellman key component g_{si}. The key
component$g_{si}$is forwarded to and stored by M. Similarly, C generates its Diffie-Hellman key component $g_c$and sends it to M.[8] At the end of Phase I, C receives all the key components corresponding to all N storage devices that it may access within time period v, and a set of authentication tokens of the form _ $(K_{MSi}; ID_C; ID_{Si}; v; g_c; g_{si})$. We note that for ease of exposition, we use the same key $K_{MSi}$for encryption in step (1) and MAC in step (2). In actual implementation, however, we assume that different keys are derived for encryption and MAC, respectively, with K_{MSi}as the master key. For example, the encryption key can be set to be $F(K_{MSi}; —enc‖)$, while the MAC key can be set to be $F(K_{MSi}; —mac‖)$. Steps (1) & (2) of Phase II are identical to those in the previous variants. In step (3), C submits its Diffie-Hellman component $g_c$in addition to other information required in step (3) of pNFS-AKE-I. S_i must verify the authentication token to ensure the integrity of g_c. Here C and S_i compute $sk_{zi}$for $z = 0; 1$ as follow:
$sk_{zi}= F(g_{csi}; ID_C; ID_{Si}; g_c; g_{si}; v; sid; z)$:

At the end of the protocol, C and S_i share a session key sk_{1i} .

Note that since C distributes its chosen Diffie-Hellman value $g_c$during each protocol run (in Phase II), each S_i needs

to store only its own secret value $s_i$and is not required to maintain a list of g^cvalues for different clients. Upon expiry of v, they erase their secret values c and s_i, respectively, from their internal states (or memory). Clearly, M does not learn anything about sk_z
$_i$unless it colludes with the associated C or S_i, and thus achieving escrow-freeness.

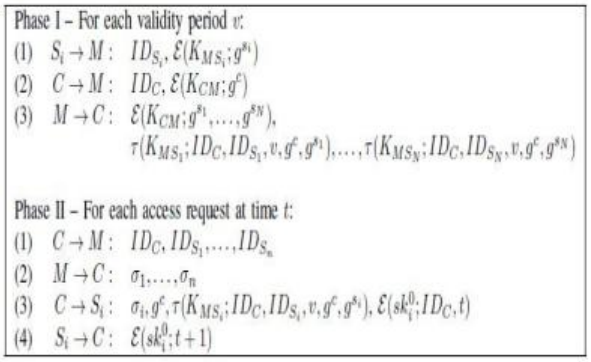

Fig 4 Specification of pNFS- AKE-II (With partial forward secrecy and escrow free)

C. pNFS-AKE-III

As explained before, pNFS-AKE-II achieves only partial forward secrecy (with respect to v). In the third variant of our pNFS-AKE, therefore, we attempt to design a protocol that achieves full forward secrecy and escrow- reeness. A straightforward and well-known technique to do this is through requiring both C and S_i to generate and exchange fresh Diffie-Hellman components for each access request at time t. However, this would drastically increase the computational overhead at the client and the storage devices. Hence, we adopt a different approach here by combining the Diffie-Hellman key exchange technique used in pNFS-AKE-II with a very efficient key update mechanism. The latter allows session keys to be derived using only symmetric key operations based on a agreed Diffie-Hellman key. Our protocol is illustrated in Figure 5.

Phase I of the protocol is similar to that of pNFS-AKEII.
In addition, M also distributes C's chosen Diffie-Hellman component $g^c$to each S_i. Hence, at the end of Phase I, both C and S_i are able to agree on a Diffie-Hellman value g^{cs_i}. Moreover, C and S_i set $F_1(g^{cs_i}; ID_C; ID_{S_i}; v)$ to be their initial shared secret state K_{0CS_i} .9 During each access request at time t in Phase II, steps (1) & (2) of the protocol are identical to those in pNFS-AKE-II. In step (3), however, C can directly establish a secure session with S_i by computing $sk_{j,zi}$as follows:

$$sk_{j,zi} = F_2(K_{j-1CS_i}; ID_C; ID_{S_i}; j; sid; z)$$

where$j$ _ 1 is an increasing counter denoting the j-th session between C and S_i with session key $sk_{j,1\ i}$. Both C and S_i then set

$$K_{jCS_i} = F_1(K_{j-1CS_i}; j)$$

and update their internal states. Note that here we use two different key derivation functions F_1 and F_2 to compute $K_{jCS_i}$and $sk_{j,zi}$, respectively. Our design can enforce independence among different session keys. Even if the adversary has obtained a session key $sk_{j,1i}$, the adversary cannot derive $K_{j-1\ CS_i}$or K_{jCS_i}. Therefore, the adversary cannot obtain $sk_{j+1,zi}$or any of the following session keys. It is worth noting that the shared state K_{jCS_i}should never be used as the session key in real communications, and just like the long-term secret key, it should be kept at a safe place, since otherwise, the adversary can use it to derive all the subsequent session keys within the validity period (i.e., K_{jCS_i}can be regarded as a medium-term secret key material). This is similar to the situation that once the adversary compromises the long-term secret key, it can
learn all the subsequence sessions.
However, we stress that knowing the state information K_{jCS_i}allows the adversary to compute only the subsequence sessionkeys (i.e., $sk_{j+1,zi}$; $sk_{j+2,zi}$; _ _ _) within a validity period, but*not* the previous session keys (i.e., $sk_{1,\ i}$; $sk_{2,zi}$; _ _ _ ; $sk_{j,zi}$)within the same period. Our construction achieves thisby making use of one-way hash chains constructed usingthe pseudo-random function F_1. Since knowing K_{jCS_i}does not help the adversary in obtaining the previous states(K_{j-1CS_i};K_{j-2CS_i}; :::;$K_{0C\ S_i}$), we can prevent the adversary from
obtaining the corresponding session keys. Also, since compromiseof K_{MS_i}or K_{CM} does not reveal the initial state K_{0CS_i}during the Diffie-Hellman key exchange, we can achieve fullforward secrecy.

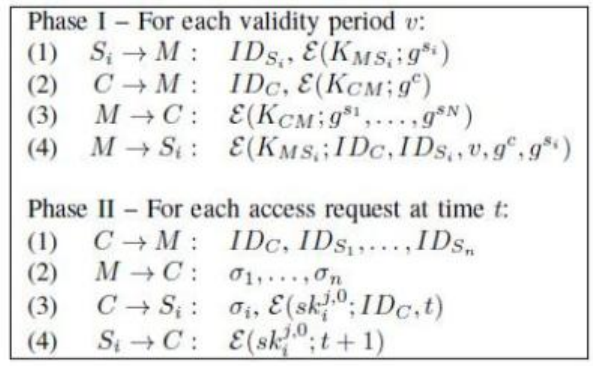

Fig 3 Specification of pNFS- AKE-III (With full forward secrecy and escrow free)

V. Performance Evaluation

A. Computational Overhead

We consider the computational overhead for w accessrequests over time period v for a metadata server M, a client C, and storage devices S_i for i *2* [1*;N*]. We ssume 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. Our estimation is based on a fixed message size of 1024 bytes for all cryptographic operations, and we consider the following case:

- $N = 2n$ and $w = 50$ (total access requests by C within v);
- C interacts with 10^3 storage devices concurrently for each access request, *i.e.* $n = 10^3$;
- M has interacted with 10^5 clients over time period v; and
- each S_i has interacted with 10^4 clients over time period v.

Table II shows that our protocols reduce the workload of in the existing Kerberos-based protocol by up to approximately 54%. This improves the scalability of the metadata server considerably. The total estimated computational cost for M for serving 10^5 clients is 8:02 _ 10^4 s (_ 22.3 hours) in Kerberos-based pNFS, compared with 3:68 _ 10^4 s (_ 0.2 hours) in pNFS-AKE-I and 3:86 _ 10^4 s (_ 10.6 hours) in pNFS-AKE-III. In general, one can see from Table I that the workload of M is always reduced by roughly half for any values of (w; n;N). The scalability of our protocols from the

TABLE I
COMPARISON IN TERMS OF CRYPTOGRAPHIC OPERATIONS FOR w ACCESS REQUESTS FROM C TO S_i VIA M OVER TIME PERIOD v, FOR ALL $1 \leq i \leq n$ AND WHERE $n \leq N$.

Protocol	M	C	all S_i	Total
Kerberos-pNFS				
– Symmetric key encryption / decryption	$w(n+5)$	$w(2n+3)$	$3wn$	$w(6n+8)$
– MAC generation / verification	wn	0	wn	$2wn$
pNFS-AKE-I				
– Symmetric key encryption / decryption	$N+1$	$2wn+1$	$3wn$	$5wn+N+2$
– MAC generation / verification	wn	0	wn	$2wn$
– Key derivation	0	$2wn$	$2wn$	$4wn$
pNFS-AKE-II				
– Symmetric key encryption / decryption	$N+2$	$2wn+2$	$2wn+1$	$4wn+N+5$
– MAC generation / verification	$wn+N$	0	$2wn$	$3wn+N$
– Key derivation	0	$2wn$	$2wn$	$4wn$
– Diffie-Hellman exponentiation	0	$N+1$	$N+wn$	$2N+wn+1$
pNFS-AKE-III				
– Symmetric key encryption / decryption	$2N+2$	$2wn+2$	$2wn+1$	$4wn+2N+5$
– MAC generation / verification	wn	0	wn	$2wn$
– Key derivation	0	$3wn+N$	$3wn+N$	$6wn+2N$
– Diffie-Hellman exponentiation	0	$N+1$	$2N$	$3N+1$

server's perspective in terms of supporting a large number of clients is further illustrated in the left graph of Figure 6 when we consider each client requesting access to an average of $n = 10^3$ storage devices.

Moreover, the additional overhead for C (and all S_i) for achieving full forward secrecy and escrow-freeness using our techniques are minimal. The right graph of Figure 6 shows that our pNFS-AKE-III protocol has roughly similar computational overhead in comparison with Kerberos-pNFS when the number of accessed storage devices is small; and the increased computational overhead for accessing 10^3 storage devices in parallel is only roughly 1/500 of a second compared to that of Kerberos-pNFS—a very reasonable trade-off between efficiency and security. The small increase in overhead is partly due to the fact that some of our cryptographic cost is amortized over a time period v (recall that and for each access request at time t, the client runs only Phase II of the protocol).

On the other hand, we note that the significantly highercomputational overhead incurred by S_i in pNFS-AKE-II is largely due to the cost of Diffie-Hellman exponentiations. This is a space-computation trade-off as explained in Section V-B (see Section VII-C for further discussion on key storage). Nevertheless, 256 s is an average computation time for 10^3 storage devices over time period v, and thus the average computation time for a storage device is still reasonably small, *i.e.* less than 1/3 of a second over time period v. Moreover, we can reduce the computational cost for S_i to roughly similar to that of pNFS-AKE-III if C pre-distributes its g^c value to all relevant S_i so that they can pre-compute the g^{cs_i} value for each time period v.

B. Communication Overhead

Assuming fresh session keys are used to secure communications between the client and multiple storage devices, clearly all our protocols have reduced bandwidth requirements. This is because during each access request, the client does not need to fetch the required authentication token set from M. Hence, the reduction in bandwidth consumption is approximately the size of n authentication tokens.

C. 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.

TABLE II

COMPARISON IN TERMS OF COMPUTATION TIMES IN SECONDS (S) OVER TIME PERIOD v BETWEEN KERBEROS-PNFS AND OUR PROTOCOLS. HERE **FFS** DENOTES FULL FORWARD SECRECY, WHILE **EF** DENOTES ESCROW-FREENESS.

Protocol	FFS	EF	M	C	S_i
Kerberos-pNFS			8.02×10^4	0.90	17.00
pNFS-AKE-I			3.68×10^4	1.50	23.00
pNFS-AKE-II		✓	3.82×10^4	2.40	256.00
pNFS-AKE-III	✓	✓	3.86×10^4	2.71	39.60

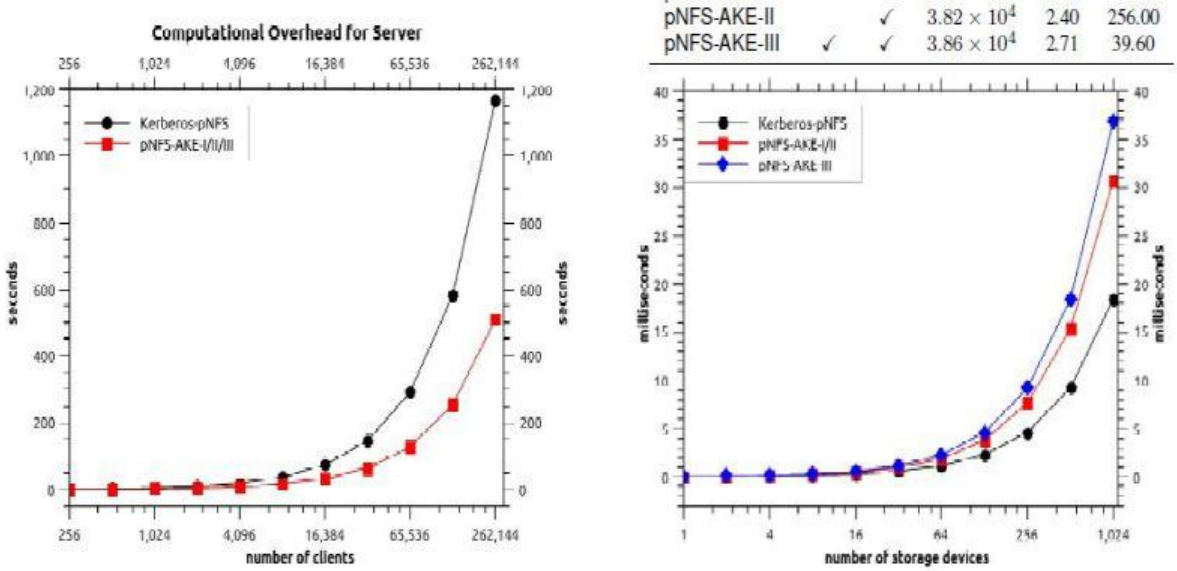

Fig. 6. Comparison in terms of computation times for M (on the left) and for C (on the right) at a specific time t.

VI. CONCLUSIONS

We proposed three authenticated key exchangeprotocols for parallel network file system (pNFS). Our protocols offer threeappealing advantages over the existing Kerberos-based pNFSprotocol. First, the metadata server executing our protocolshas much lower workload than that of the Kerberos-basedapproach. Second, two our protocols provide forward secrecy:one is partially forward secure (with respect to multiplesessions within a time period), while the other is fully forwardsecure (with respect to a session). Third, we have designed aprotocol which not only provides forward secrecy, but is alsoescrow-free.

VII. REFERENCES

[1] M. Abd-El-Malek, W.V. Courtright II, C. Cranor, G.R. Ganger, J. Hen-dricks, A.J. Klosterman, M.P. Mesnier, M. Prasad, B. Salmon, R.R. Sam-basivan, 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 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. Block-level security for network-attached disks. In *Proceedings of the 2nd International Conference on File and Storage Technologies (FAST)*. USENIX Association, Mar 2003.

[5] Christo Ananth, Mary Varsha Peter, Priya.M., Rajalakshmi.R., Muthu Bharathi.R., Pramila.E., "Network Fault Correction in Overlay Network through Optimality", International Journal of Advanced Research Trends in Engineering and Technology (IJARTET), Volume 2, Issue 8, August 2015, pp: 19-22

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