Authenticated Key Exchange Protocols for Parallel Network File Systems

Hoon Wei Lim and Guomin Yang

Abstract—We study 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 percent 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.

Index Terms—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 CloudStore [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 (NFSs) 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.

Our primary goal in this work is to design efficient and secure authenticated key exchange protocols that meet specific requirements of pNFS. Particularly, we attempt to meet the following desirable properties, which either have not been satisfactorily achieved or are not achievable by the current Kerberos-based solution (as described in Section 2):

- *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

- *H.W. Lim is with the School of Computing, National University of Singapore, Singapore. E-mail: hoonwei@nus.edu.sg.*
- *G. Yang is with the School of Computer Science and Software Engineering, University of Wollongong, Australia. E-mail: gyang@uow.edu.au.*

Manuscript received 23 July 2014; revised 7 Dec. 2014; accepted 29 Dec. 2014. Date of publication 6 Jan. 2015; date of current version 16 Dec. 2015. Recommended for acceptance by G. Wang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2015.2388447

- *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 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 3, we describe the threat model for pNFS and the existing Kerberos-based protocol. In Section 4, 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 6 and 7, respectively. In Section 8, we describe related work, and finally in Section 9, we conclude and discuss some future work.

2 Internet Standard—NFS

Network File System [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 [9], [45], [49]. 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.

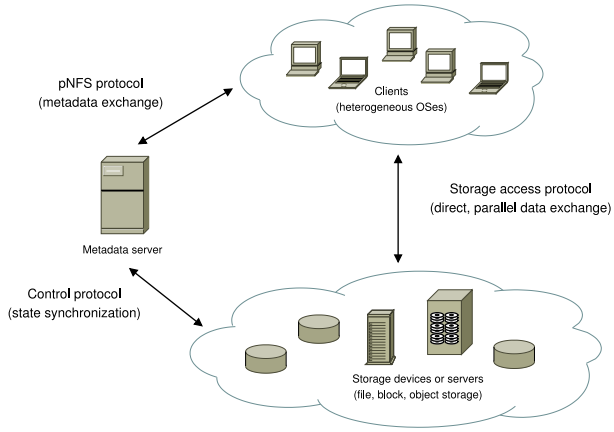

Fig. 1. The conceptual model of pNFS.

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. Fig. 1 illustrates 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]

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

1. A layout can be seen as a map, describing how a file is distributed across the data storage system. When a client holds a layout, it is granted the ability to directly access the byte-range at the storage location specified in the layout.

2. Note that the control protocol is not specified in NFSv4.1. It can take many forms, allowing vendors the flexibility to compete on performance, cost, and features.

[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. Moreover, consideration should be given to the integrity and privacy (confidentiality) of NFS requests and responses [45].

The RPCSEC_GSS framework [16], [17] is currently the core security component of NFS that provides basic 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 predefined 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].

2.2 Kerberos and LIPKEY

In NFSv4, the Kerberos version 5 [18], [32] 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 non-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.

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 GSS-API 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 certification authority (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 order to obtain one or more service tickets (each containing a session key for access to a storage device), and valid layouts (each presenting valid access permissions to a storage device according to the ACLs);
3) the client finally presents the service tickets and layouts to the corresponding storage devices to get access to the stored data objects or files.

We describe the above Kerberos-based key establishment protocol in more detail in Section 3.3.

Secure storage access. The session key generated by the ticket-granting server (metadata server) for a client and a storage device during single sign-on can then be used in the storage access protocol. It protects the integrity and privacy of data transmitted between the client and the storage

3. Typically, operating system principles are matched to a set of user and group access control lists.

(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)	$S_i \rightarrow C:$	$E(sk_i; t+1)$

Fig. 2. A simplified version of the Kerberos-based pNFS protocol.

device. Clearly, the session key and the associated layout are valid only within the granted validity period.

2.3 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 key shared with the metadata server is compromised. We believe that this is a realistic threat since a large-scale file system may have thousands of geographically distributed storage devices. It may not be feasible to provide strong physical security and network protection for all the storage devices.

3 Preliminaries

3.1 Notation

We let M denote a metadata server, C denote a client, and S denote a storage device. Let entity $X, Y \in \{M, C, S\}$, we then use ID_X to denote a unique identity of X, and K_X to denote X's secret (symmetric) key; while K_{XY} denotes a secret key shared between X and Y, and sk denotes a session key.

Moreover, we let $E(K; m)$ be a standard (encryption only) symmetric key encryption function and let $\mathcal{E}(K; m)$ be an authenticated symmetric key encryption function, where both functions take as input a key K and a message m. Finally, we use t to represent a current time and σ to denote a layout. We may introduce other notation as required.

3.2 Threat Assumptions

Existing proposals [19], [29], [30], [31], [40] on secure large-scale distributed file systems typically assume that both the metadata server and the storage device are trusted entities. On the other hand, no implicit trust is placed on the clients.

The metadata server is trusted to act as a reference monitor, issue valid layouts containing access permissions, and sometimes even generate session keys (for example, in the case of Kerberos-based pNFS) for secure communication between the client and the storage devices. The storage devices are trusted to store data and only perform I/O operations upon authorized requests. However, we assume that the storage devices are at a much higher risk of being compromised compared to the metadata server, which is typically easier to monitor and protect in a centralized location. Furthermore, we assume that the storage devices may occasionally encounter hardware or software failure, causing the data stored on them no longer accessible.

We note that this work focuses on communication security. Hence, we assume that data transmitted between the client and the metadata server, or between the client and the storage device can be easily eavesdropped, modified or deleted by an adversary. However, we do not address storage related security issues in this work. Security protection mechanisms for data at rest are orthogonal to our protocols.

3.3 Kerberos-Based pNFS Protocol

For the sake of completeness, we describe the key establishment protocol[4] recommended for pNFS in RFC 5661 between a client C and n storage devices S_i, for $1 \leq i \leq n$, through a metadata server M in Fig. 2. We will compare the efficiency of the pNFS protocol against ours in Section 7.

During the setup phase, we assume that M establishes a shared secret key K_{MS_i} with each S_i. Here, K_C is a key derived from C's password, that is also known by M; while T plays the role of a ticket-granting server (we simply assume that it is part of M). Also, prior to executing the protocol in Fig. 2, we assume that C and M have already setup a secure channel through LIPKEY (as described in Section 2.2).

Once C has been authenticated by M and granted access to $S_1, \ldots, S_n$, it receives a set of service tickets $E(K_{MS_i}; ID_C, t, sk_i)$, session keys sk_i, and layouts[5] σ_i (for all $i \in [1, n]$) from T, as illustrated in step (4) of the protocol. Clearly, we assume that C is able to uniquely extract each session key sk_i from $E(K_{CT}; sk_1, \ldots, sk_n)$. Since the session keys are generated by M and transported to S_i through C, no interaction is required between C and S_i (in terms of key exchange) in order to agree on a session key. This keeps the communication overhead between the client and each storage device to a minimum in comparison with the case where key exchange is required. Moreover, the computational overhead

4. For ease of exposition, we do not provide complete details of the protocol parameters.

5. We assume that a layout (containing the client's identity, file object mapping information, and access permissions) is typically integrity protected and it can be in the form of a signature or MAC.

for the client and each storage device is very low since the protocol is mainly based on symmetric key encryption.

The message in step (6) serves as key confirmation, that is to convince C that S_i is in possession of the same session key that C uses.

4 Overview of Our Protocols

We describe our design goals and give some intuition of a variety of pNFS authenticated key exchange[6] (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.

4.1 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 3.3. 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.

4.2 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 $1 \leq i \leq n$, 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 pre-computed 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). As with Kerberos, 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.
- 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 S_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 (PFS) (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 (FFS), 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 pNFS-AKE-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.

5 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.

5.1 pNFS-AKE-I

Our first pNFS-AKE protocol is illustrated in Fig. 3. For each validity period v, C must first pre-compute a set of key

6. Without loss of generality, we use the term "key exchange" here, although key establishment between two parties can be based on either key transport or key agreement [39].

Phase I – For each validity period v:
(1) $C \rightarrow M$: ID_C, $\mathcal{E}(K_{CM}; K_{CS_1}, \ldots, K_{CS_N})$
(2) $M \rightarrow C$: $\mathcal{E}(K_{MS_1}; ID_C, ID_{S_1}, v, K_{CS_1}), \ldots, \mathcal{E}(K_{MS_N}; ID_C, ID_{S_N}, v, K_{CS_N})$

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

Fig. 3. Specification of pNFS-AKE-I.

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

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

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

materials $K_{CS_1}, \ldots, K_{CS_N}$ before it can access any of the N storage device S_i (for $1 \leq i \leq N$). 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 2.2. M then issues an authentication token of the form $\mathcal{E}(K_{MS_i}; ID_C, ID_{S_i}, v, K_{CS_i})$ for each key material if the associated storage device S_i 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 $1 \leq i \leq n \leq N$ that C wishes to access. For each S_i, M issues a layout σ_i. C then forwards the respective layouts, authentication tokens (from Phase I), and encrypted messages of the form $\mathcal{E}(sk_i^0; ID_C, t)$ 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 σ_i is valid;
2) decrypt the authentication token and recover key K_{CS_i};
3) compute keys $sk_i^z = F(K_{CS_i}; ID_C, ID_{S_i}, v, sid, z)$ 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_i^0.

At the end of the protocol, sk_i^1 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.

5.2 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 $g^x \in \mathbb{G}$ denote a Diffie-Hellman component, where $\mathbb{G}$ is an appropriate group generated by g, and x is a number randomly chosen by entity $X \in \{C, S\}$. Let $\tau(k, m)$ 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 Fig. 4.

At the beginning of each v, each S_i that is governed by M generates a Diffie-Hellman key component g^{s_i}. The key component g^{s_i} 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 $\tau(K_{MS_i}; ID_C, ID_{S_i}, v, g^c, g^{s_i})$. We note that for ease of exposition, we use the same key K_{MS_i} 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_{MS_i} as the

7. Here K_{MS_i} is regarded as a long-term symmetric secret key shared between M and S_i. Also, we use authenticated encryption instead of encryption only encryption for security reasons. This will be clear in our security analysis.

8. For consistency with the existing design of the Kerberos protocol, we assume that the Diffie-Hellman components are "conveniently" transmitted through the already established secure channel between them, although the Diffie-Hellman components may not necessarily be encrypted from a security view point.

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

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

Fig. 5. Specification of pNFS-AKE-III (with full forward secrecy and escrow-free).

master key. For example, the encryption key can be set to be $F(K_{MS_i}; \text{“enc”})$, while the MAC key can be set to be $F(K_{MS_i}; \text{“mac”})$.

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_i^z for $z = 0, 1$ as follow:

$$sk_i^z = F(g^{cs_i}; ID_C, ID_{S_i}, g^c, g^{s_i}, v, sid, z).$$

At the end of the protocol, C and S_i share a session key sk_i^1.

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^c values 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_i^z unless it colludes with the associated C or S_i, and thus achieving escrow-freeness.

5.3 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-freeness. 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 Fig. 5.

Phase I of the protocol is similar to that of pNFS-AKE-II. 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 $K_{CS_i}^0$.[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 $sk_i^{j,z}$ as follows:

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

where $j \geq 1$ is an increasing counter denoting the jth session between C and S_i with session key $sk_i^{j,1}$. Both C and S_i then set

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

and update their internal states. Note that here we use two different key derivation functions F_1 and F_2 to compute $K_{CS_i}^j$ and $sk_i^{j,z}$, respectively. Our design can enforce independence among different session keys. Even if the adversary has obtained a session key $sk_i^{<!----->j,1}$, the adversary cannot derive $K_{CS_i}^{j-1}$ or $K_{CS_i}^j$. Therefore, the adversary cannot obtain $sk_i^{j+1,z}$ or any of the following session keys. It is worth noting that the shared state $K_{CS_i}^j$ 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., $K_{CS_i}^j$ 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 $K_{CS_i}^j$ allows the adversary to compute only the subsequence session keys (i.e., $sk_i^{j+1,z}, sk_i^{j+2,z}, \ldots$) within a validity period, but *not* the previous session keys (i.e., $sk_i^{1,z}, sk_i^{2,z}, \ldots, sk_i^{j,z}$) within the same period. Our construction achieves this by making use of one-way hash chains constructed using the pseudo-random function F_1. Since knowing $K_{CS_i}^j$ does not help the adversary in obtaining the previous states $(K_{CS_i}^{j-1}, K_{CS_i}^{j-2}, \ldots, K_{CS_i}^0)$, we can prevent the adversary from obtaining the corresponding session keys. Also, since compromise of K_{MS_i} or K_{CM} does not reveal the initial state $K_{CS_i}^0$ during the Diffie-Hellman key exchange, we can achieve full forward secrecy.

6 Security Analysis

We work in a security model that allows us to show that an adversary attacking our protocols will not able to learn any information about a session key. Our model also implies implicit authentication, that is, only the right protocol participant is able to learn or derive a session key.

9. Unlike in pNFS-AKE-II where g^c is distributed in Phase II, we need to pre-distribute C's chosen Diffie-Hellman component in Phase I because the secret state $K_{CS_i}^0$ that C and S_i store will be updated after each request. This is essential to ensure forward secrecy.

6.1 Security Model

We now define a security model for pNFS-AKE. Let M denote the metadata server, $\mathsf{SS} = \{S_1, S_2, \ldots, S_N\}$ the set of storage devices, and $\mathsf{CS} = \{C_1, C_2, \ldots, C_\ell\}$ the set of clients. A party $P \in \{M\} \cup \mathsf{SS} \cup \mathsf{CS}$ may run many instances concurrently, and we denote instance i of party P by Π_P^i.

Our adversarial model is defined via a game between an adversary $\mathcal{A}$ and a game simulator $\mathcal{SIM}$. $\mathcal{SIM}$ tosses a random coin b at the beginning of the game and b will be used later in the game. $\mathcal{SIM}$ then generates for each $S_i \in \mathsf{SS}$ ($C_j \in \mathsf{CS}$, respectively) a secret key K_{MS_i} (K_{MC_j}, respectively) shared with M. $\mathcal{A}$ is allowed to make the following queries to the simulator:

- $\textsc{Send}(P, i, m)$. This query allows the adversary to send a message m to an instance Π_P^i. If the message m is sent by another instance $\Pi_{P'}^j$ with the intended receiver P, then this query models a passive attack. Otherwise, it models an active attack by the adversary. The simulator then simulates the reaction of Π_P^i upon receiving the message m, and returns to $\mathcal{A}$ the response (if there is any) that Π_P^i would generate.
- $\textsc{Corrupt}(P)$. This query allows the adversary to corrupt a party $P \in \mathsf{SS} \cup \mathsf{CS}$. By making this query, the adversary learns all the information held by P at the time of the corruption, including all the long-term and ephemeral secret keys. However, the adversary cannot corrupt M (but see Remark 1).
- $\textsc{Reveal}(P, i)$. This query allows the adversary to learn the session key that has been generated by the instance Π_P^i ($P \in \mathsf{SS} \cup \mathsf{CS}$). If the instance Π_P^i does not hold any session key, then a special symbol $\perp$ is returned to the adversary.
- $\textsc{Test}(P^*, i^*)$. This query can only be made to a *fresh* instance $\Pi_{P^*}^{i^*}$ (as defined below) where $P^* \in \mathsf{SS} \cup \mathsf{CS}$. If the instance $\Pi_{P^*}^{i^*}$ holds a session key $\mathsf{SK}_{P^*}^{i^*}$, then $\mathcal{SIM}$ does the following
  - if the coin $b = 1$, $\mathcal{SIM}$ returns $\mathsf{SK}_{P^*}^{i^*}$ to the adversary;
  - otherwise, a random session key is drawn from the session key space and returned to the adversary.

  Otherwise, a special symbol $\perp$ is returned to the adversary.

We define the partner id pid_P^i of an instance Π_P^i as the identity of the peer party recognized by Π_P^i, and sid_P^i as the unique session id belonging to Π_P^i. We say a client instance Π_C^i and a storage device instance Π_S^j are *partners* if $\mathsf{pid}_C^i = S$ and $\mathsf{pid}_S^j = C$ and $\mathsf{sid}_C^i = \mathsf{sid}_S^j$.

We say an instance Π_P^i is *fresh* if

- $\mathcal{A}$ has never made a $\textsc{Corrupt}$ query to P or pid_P^i; and
- $\mathcal{A}$ has never made a $\textsc{Reveal}$ query to Π_P^i or its partner.

At the end of the game, the adversary outputs a bit b' as her guess for b. The adversary's advantage in winning the game is defined as

$$\mathbf{Adv}_{\mathcal{A}}^{\mathrm{pNFS}}(k) = |2\Pr[b' = b] - 1|.$$

Definition 1. *We say a pNFS-AKE protocol is secure if the following conditions hold.*

1) *If an honest client and an honest storage device complete matching sessions, they compute the same session key.*
2) *For any PPT adversary A, $\mathbf{Adv}_{\mathcal{A}}^{\mathrm{pNFS}}(k)$ is a negligible function of k.*

Forward secrecy. The above security model for pNFS-AKE does not consider forward secrecy (i.e., the corruption of a party will not endanger his/her previous communication sessions). Below we first define a weak form of forward secrecy we call *partial forward secrecy*. We follow the approach of Canetti and Krawczyk [10] by introducing a new type of query:

- $\textsc{Expire}(P, v)$. After receiving this query, no instance of P for time period v could be activated. In addition, the simulator erases all the state information and session keys held by the instances of party P which are activated during time period v.

Then, we redefine the *freshness* of an instance Π_P^i as follows:

- $\mathcal{A}$ makes a $\textsc{Corrupt}(P)$ query only after an $\textsc{Expire}(P, v)$ query where the instance Π_P^i is activated during time period v;
- $\mathcal{A}$ has never made a $\textsc{Reveal}(P, i)$ query; and
- If Π_P^i has a partner instance Π_Q^j, then $\mathcal{A}$ also obeys the above two rules with respect to Π_Q^j; otherwise, $\mathcal{A}$ has never made a $\textsc{Corrupt}(\mathsf{pid}_P^i)$ query.

The rest of the security game is the same. We define the advantage of the adversary as

$$\mathbf{Adv}_{\mathcal{A}}^{\mathrm{pNFS-PFS}}(k) = |2\Pr[b' = b] - 1|.$$

We can easily extend the above definition to define *full forward secrecy* by modifying the $\textsc{Expire}$ query as follows:

- $\textsc{Expire}(P, i)$. Upon receiving this query, the simulator erases all the state information and the session key held by the instance Π_P^i.

The rest of the security model is the same as in the PFS game.

Remark 1. In our security model, we do not allow the adversary to corrupt the metadata server M which holds all the long-term secret keys. However, in our Forward Secrecy model, we actually do not really enforce such a requirement. It is easy to see that if the adversary corrupts all the parties in $\mathsf{SS} \cup \mathsf{CS}$, then the adversary has implicitly corrupted M. But we should also notice that there is no way to prevent active attacks once M is corrupted. Therefore, the adversary can corrupt all the parties (or M) only after the Test session has expired.

Remark 2. Our above Forward Secrecy model has also escrow-freeness. One way to define escrow-freeness is to define a new model which allows the adversary to corrupt the metadata server and learn all the long-term secret keys. However, as outlined in Remark 1, our Forward Secrecy model allows the adversary to obtain all the long-term secret keys under some necessary conditions. Hence, our Forward Secrecy model has implicitly captured escrow-freeness.

6.2 Security Proofs

Theorem 1. *The* pNFS-AKE-I *protocol is secure without PFS if the authenticated encryption scheme $\mathcal{E}$ is secure under chosen-ciphertext attacks (CCA) and F is a family of pesudo-random functions.*

Proof. We define a sequence of games $G_i (i \geq 0)$ where G_0 is the original game defined in our security model without PFS. We also define $\mathbf{Adv}_i^{\text{pNFS}}$ as the advantage of the adversary in game G_i. Then we have $\mathbf{Adv}_0^{\text{pNFS}} = \mathbf{Adv}_{\mathcal{A}}^{\text{pNFS}}(k)$.

In game G_1 we change the original game as follows: the simulator randomly chooses an instance $\Pi_{P^*}^{i^*}$ among all the instances created in the game, if the TEST query is not performed on $\Pi_{P^*}^{i^*}$, the simulator aborts and outputs a random bit. Then we have

$$\mathbf{Adv}_1^{\text{pNFS}} = \frac{1}{n_I}\mathbf{Adv}_0^{\text{pNFS}},$$

where n_I denotes the number of instances created in the game. In the following games, we use C^* and S^* to denote the client and the storage device involved in the test session, respectively, and v^* to denote the time period when the test session is activated.

In game G_2, we change game G_1 as follows: let FORGE denote the event that $\mathcal{A}$ successfully forges a valid ciphertext $\mathcal{E}(K_{MS^*}; ID_{C^*}, ID_{S^*}, v^*, K_{C^*S^*})$. If the event FORGE happens, then the simulator aborts the game and outputs a random bit. Since $\mathcal{E}$ is a secure authenticated encryption scheme, we have

$$\begin{aligned}&\Pr[b' = b \text{ in game } G_1 | \overline{\mathsf{FORGE}}]\\ &\quad = \Pr[b' = b \text{ in game } G_2 | \overline{\mathsf{FORGE}}]\end{aligned}$$

and

$$\begin{aligned}&\Pr[b' = b \text{ in game } G_1] - \Pr[b' = b \text{ in game } G_2]\\ &\quad \leq \Pr[\mathsf{FORGE}] \leq \mathbf{Adv}_{\mathcal{E}}^{\text{UF-CMA}}(\mathsf{k}).\end{aligned}$$

Therefore, we have

$$\mathbf{Adv}_1^{\text{pNFS}} \leq \mathbf{Adv}_2^{\text{pNFS}} + 2\mathbf{Adv}_{\mathcal{E}}^{\text{UF-CMA}}(k).$$

In game G_3 we use a random key K' instead of the decryption of $\mathcal{E}(K_{MS^*}; ID_{C^*}, ID_{S^*}, v^*, K_{C^*S^*})$ to simulate the game. In the following, we show that $|\mathbf{Adv}_2^{\text{pNFS}} - \mathbf{Adv}_1^{\text{pNFS}}|$ is negligible if the authenticated encryption scheme $\mathcal{E}$ is secure under adaptive chosen-ciphertext attacks.

We construct an adversary $\mathcal{B}$ in the CCA game for the authenticated encryption scheme $\mathcal{E}$. $\mathcal{B}$ simulates the game G_2 for the adversary $\mathcal{A}$ as follows. $\mathcal{B}$ generates all the long-term keys in the system except K_{MS^*}. $\mathcal{B}$ then randomly selects two keys K_0 and K_1 and obtains a challenge ciphertext $CH^* = \mathcal{E}(K_{MS^*}; ID_{C^*}, ID_{S^*}, v^*, K^*)$ from its challenger where K^* is either K_0 or K_1. $\mathcal{B}$ then uses CH^* as the authentication token used between C^* and S^* during the time period v^*, and uses K_1 as the decryption of CH^* to perform any related computation. For other authentication tokens related to K_{MS^*}, $\mathcal{B}$ generates them by querying its encryption oracle. Also, for any authentication token intended for S^* but not equal to CH^*, $\mathcal{B}$ performs the decryption by querying its decryption oracle. Finally, if the adversary $\mathcal{A}$ wins in the game (denote this event by WIN), $\mathcal{B}$ outputs 1 (i.e., $\mathcal{B}$ guesses $K^* = K_1$), otherwise, $\mathcal{B}$ outputs 0 (i.e., $\mathcal{B}$ guesses $K^* = K_0$).

We can see that if $K^* = K_1$, then the game simulated by $\mathcal{B}$ is the same as game G_2; otherwise, if $K^* = K_0$, then the game simulated by $\mathcal{B}$ is the same as game G_3. So we have

$$\begin{aligned}\mathbf{Adv}_{\mathcal{B}}^{\text{CCA}}(k) &= |2(\Pr[\mathsf{WIN}|\mathsf{K}^* = \mathsf{K_1}]\Pr[\mathsf{K}^* = \mathsf{K_1}]\\ &\quad + \Pr[\overline{\mathsf{WIN}}|K^* = K_0]\Pr[K^* = K_0]) - 1|\\ &= \Pr[\mathsf{WIN}|\mathsf{K}^* = \mathsf{K_1}] - \Pr[\mathsf{WIN}|\mathsf{K}^* = \mathsf{K_0}]\\ &= \frac{1}{2}\left(\mathbf{Adv}_2^{\text{pNFS}} - \mathbf{Adv}_3^{\text{pNFS}}\right)\end{aligned}$$

and

$$\mathbf{Adv}_2^{\text{pNFS}} \leq \mathbf{Adv}_3^{\text{pNFS}} + 2\mathbf{Adv}_{\mathcal{E}}^{\text{CCA}}(k).$$

In game G_4 we then replace the function $F(K', \cdot)$ with a random function $RF(\cdot)$. Since F is a family of pseudo-random functions, if the adversary's advantage changes significantly in game G_4 , we can construct a distinguisher $\mathcal{D}$ against F. $\mathcal{D}$ simulates game G_3 for $\mathcal{A}$ honestly except that whenever $\mathcal{D}$ needs to compute $F(K', x)$, $\mathcal{D}$ queries its own oracle $\mathcal{O}$ which is either $F(K', \cdot)$ or $RF(\cdot)$. At the end of the game, if $\mathcal{A}$ wins the game, $\mathcal{D}$ outputs 1, otherwise, $\mathcal{D}$ outputs 0.

We can see that if $\mathcal{O} = F(K', \cdot)$, $\mathcal{A}$ is in game G_3, otherwise, if $\mathcal{O} = RF(\cdot)$, then $\mathcal{A}$ is in game G_4. Therefore, we have

$$\begin{aligned}\mathbf{Adv}_{\mathcal{D}}^{\text{prf}}(k) &= \Pr[\mathcal{D} \text{ outputs } 1 | \mathcal{O} = F(K', \cdot)]\\ &\quad - \Pr[\mathcal{D} \text{ outputs } 1 | \mathcal{O} = RF(\cdot)]\\ &= \Pr[\mathsf{WIN}|\mathcal{O} = F(K', \cdot)]\\ &\quad - \Pr[\mathsf{WIN}|\mathcal{O} = RF(\cdot)]\\ &= \frac{1}{2}\left(\mathbf{Adv}_3^{\text{pNFS}} - \mathbf{Adv}_4^{\text{pNFS}}\right)\end{aligned}$$

and

$$\mathbf{Adv}_3^{\text{pNFS}} \leq \mathbf{Adv}_4^{\text{pNFS}} + 2\mathbf{Adv}_F^{\text{prf}}(k).$$

In game G_4, we have

$$sk_i^0 = RF(ID_{C^*}, ID_{S^*}, v^*, sid^*, 0),$$

$$sk_i^1 = RF(ID_{C^*}, ID_{S^*}, v^*, sid^*, 1),$$

where sid^* is the unique session id for the test session. Now since RF is a random function, sk_i^1 is just a random key independent of the game. Therefore, the adversary has no advantage in winning the game, i.e.,

$$\mathbf{Adv}_4^{\text{pNFS}} = 0.$$

Combining all together, we have

$$\begin{aligned}&\mathbf{Adv}_{\mathcal{A}}^{\text{pNFS}}(k)\\ &\quad \leq 2n_I(\mathbf{Adv}_{\mathcal{E}}^{\text{UF-CMA}}(k) + \mathbf{Adv}_{\mathcal{E}}^{\text{CCA}}(k) + \mathbf{Adv}_F^{\text{prf}}(k)).\end{aligned}$$

□

Theorem 2. *The* pNFS-AKE-II *protocol achieves partial forward secrecy if τ is a secure MAC scheme, the DDH assumption holds in the underlying group $\mathbb{G}$, and F is a family of pesudo-random functions.*

Proof. The proof is similar to that for Theorem 1. Below we only elaborate on the differences between the two proofs. We also define a sequence of games G_i where G_0 is the original PFS security game.

In game G_1, we change game G_0 in the same way as in the previous proof, then we also have

$$\mathbf{Adv}_1^{\mathrm{pNFS-PFS}} = \frac{1}{n_I}\mathbf{Adv}_0^{\mathrm{pNFS-PFS}},$$

where n_I denotes the number of instances created in the game. Let C^* and S^* denote the client and the storage device involved in the test session, respectively, and v^* the time period when the test session is activated.

In game G_2, we further change game G_1 as follows: let FORGE denote the event that $\mathcal{A}$ successfully forges a valid MAC tag $\tau(K_{MS_i}; ID_C, ID_{S_i}, v, g^c, g^{s_i})$ before corrupting S_i, if the event FORGE happens, then the simulator aborts the game and outputs a random bit. Then we have

$$\begin{aligned}&\Pr[b' = b \text{ in game } G_1|\overline{\mathsf{FORGE}}]\\ &\quad = \Pr[b' = b \text{ in game } G_2|\overline{\mathsf{FORGE}}]\end{aligned}$$

and

$$\begin{aligned}&\Pr[b' = b \text{ in game } G_1] - \Pr[b' = b \text{ in game } G_2]\\ &\quad \le \Pr[\mathsf{FORGE}] \le \mathbf{Adv}_\tau^{\mathrm{UF-CMA}}(\mathsf{k}).\end{aligned}$$

Therefore, we have

$$\mathbf{Adv}_1^{\mathrm{pNFS-PFS}} \le \mathbf{Adv}_2^{\mathrm{pNFS-PFS}} + 2\mathbf{Adv}_\tau^{\mathrm{UF-CMA}}(k).$$

In game G_3, we change game G_2 by replacing the Diffie-Hellman key $g^{c^*s^*}$ in the test session with a random element $K' \in \mathbb{G}$. Below we show that if the adversary's advantage changes significantly in game G_3 , we can construct a distinguisher $\mathcal{B}$ to break the Decisional Diffie-Hellman (DDH) assumption.

$\mathcal{B}$ is given a challenge (g^a, g^b, Z), in which with equal probability, Z is either g^{ab} or a random element of $\mathbb{G}$. $\mathcal{B}$ simulates game G_2 honestly by generating all the long-term secret keys for all the clients and storage devices. Then for the time period v^*, $\mathcal{B}$ sets $g^{c^*} = g^a$ and $g^{s^*} = g^b$. When the value of $g^{c^*s^*}$ is needed, $\mathcal{B}$ uses the value of Z to perform the corresponding computation. Finally, if $\mathcal{A}$ wins the game, $\mathcal{B}$ outputs 1, otherwise, $\mathcal{B}$ outputs 0.

Since the adversary cannot corrupt C^* or S^* before the time period v^* has expired, if a FORGE event did not happen, then the values of the Diffie-Hellman components in the test session must be g^a and g^b. If $Z = g^{ab}$, then $\mathcal{A}$ is in game G_2; otherwise, if Z is a random element of $\mathbb{G}$, then $\mathcal{A}$ is in game G_3. Therefore we have

$$\begin{aligned}\mathbf{Adv}_{\mathcal{B}}^{\mathrm{DDH}}(k) &= \Pr[\mathcal{B} \text{ outputs } 1|Z = g^{ab}]\\ &\quad - \Pr[\mathcal{B} \text{ outputs } 1|Z = g^r]\\ &= \Pr[\mathsf{WIN}|Z = g^{ab}] - \Pr[\mathsf{WIN}|Z = g^r]\\ &= \frac{1}{2}\left(\mathbf{Adv}_2^{\mathrm{pNFS-PFS}} - \mathbf{Adv}_3^{\mathrm{pNFS-PFS}}\right)\end{aligned}$$

and

$$\mathbf{Adv}_2^{\mathrm{pNFS-PFS}} \le \mathbf{Adv}_3^{\mathrm{pNFS-PFS}} + 2\mathbf{Adv}^{\mathrm{DDH}}(k).$$

In game G_4, we replace the pseudo-random function $F(K', \cdot)$ with a random function $RF(\cdot)$. By following the same analysis as in the previous proof, we have

$$\mathbf{Adv}_3^{\mathrm{pNFS-PFS}} \le \mathbf{Adv}_4^{\mathrm{pNFS-PFS}} + 2\mathbf{Adv}_F^{\mathrm{prf}}(k)$$

and

$$\mathbf{Adv}_4^{\mathrm{pNFS-PFS}} = 0.$$

Therefore, combining all together, we have

$$\begin{aligned}&\mathbf{Adv}_{\mathcal{A}}^{\mathrm{pNFS-PFS}}(k)\\ &\le\ 2n_I\left(\mathbf{Adv}_\tau^{\mathrm{UF-CMA}}(k) + \mathbf{Adv}^{\mathrm{DDH}}(k) + \mathbf{Adv}_F^{\mathrm{prf}}(k)\right).\end{aligned}$$

□

Theorem 3. *The* pNFS-AKE-III *protocol achieves full forward secrecy if $\mathcal{E}$ is a secure authenticated encryption scheme, the DDH assumption holds in the underlying group $\mathbb{G}$, and F is a family of pesudo-random functions.*

Proof (Sketch). The proof is very similar to that for Theorem 2. Below we provide a sketch of the proof.

Let C^* and S^* denote the client and the storage device involved in the test session, respectively, and v^* the time period when the test session is activated. Without loss of generality, suppose the test session is the jth session between C^* and S^* within the period v^*. Since the adversary is not allowed to corrupt C^* or S^* before the test session session has expired, due to the unforgeability of $\mathcal{E}$, and the DDH assumption, the simulator can replace $g^{c^*s^*}$ in the time period v^* with a random element $K \in \mathbb{G}$. Then in the next augmented game, the simulator replaces $K^0_{C^*S^*}$ by a random key. Since F_1 is a secure pseudo-random function, such a replacement is indistinguishable from the adversary's view point. The simulator then replaces $sk_i^{i,z}$ (for $z = 0, 1$) and $K^i_{C^*S_*}$ with independent random keys for all $1 \le i \le j$. Once again, since F_1 and F_2 are secure pseudo-random functions, the augmented games are indistinguishable by the adversary. Finally, in the last augmented game, we can claim that the adversary has no advantage in winning the game since a random key is returned to the adversary no matter $b = 0$ or $b = 1$. This completes the sketch of the proof. □

7 Performance Evaluation

7.1 Computational Overhead

We consider the computational overhead for w access requests over time period v for a metadata server M, a client

TABLE 1
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	$N+1$	$2N$	$3N+1$

C, and storage devices S_i for $i \in [1, N]$. We assume that a layout σ is of the form of a MAC, and the computational cost for authenticated symmetric encryption $\mathcal{E}$ is similar to that for the non-authenticated version E.[10] Table 1 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 2 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 1,024-bit key pair generation. Our estimation is based on a fixed message size of 1,024 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 2 shows that our protocols reduce the workload of M in the existing Kerberos-based protocol by up to approximately 54 percent. 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 (≈10.2 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 1 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 server's perspective in terms of supporting a large number of clients is further illustrated in the left graph of Fig. 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 Fig. 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 higher computational 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 5.2 (see Section 7.3 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

TABLE 2
Comparison in Terms of Computation Times in Seconds (s) over Time Period v between Kerberos-pNFS and Our Protocols

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

*Here **FFS** denotes full forward secrecy, while **EF** denotes escrow-freeness.*

10. For example, according to the Crypto++ 5.6.0 Benchmarks, AES/GCM (128-bit, 64K tables) has similar speed as AES/CBC (128-bit key) [12].

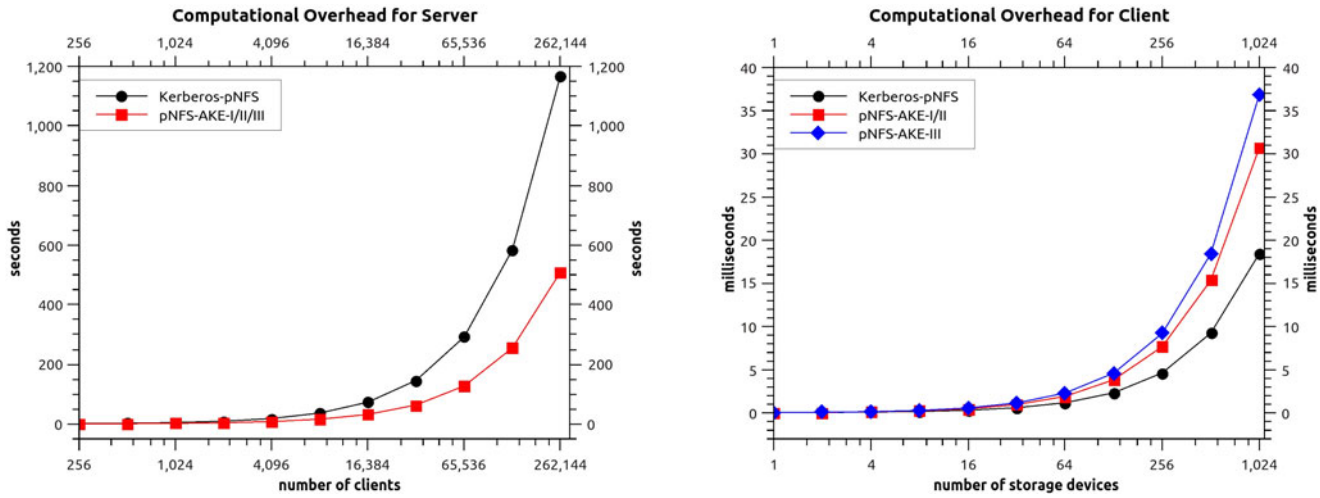

Fig. 6. Comparison in terms of computation times for M (on the left) and for C (on the right) at a specific time t.

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.

7.2 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.

7.3 Key Storage

We note that the key storage requirements for Kerberos-pNFS 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.

8 Other Related Work

Some of the earliest work in securing large-scale distributed file systems, for example [22], [24], have already employed Kerberos for performing authentication and enforcing access control. Kerberos, being based on mostly symmetric key techniques in its early deployment, was generally believed to be more suitable for rather closed, well-connected distributed environments.

On the other hand, data grids and file systems such as, OceanStore [27], LegionFS [54] and FARSITE [3], make use of public key cryptographic techniques and public key infrastructure (PKI) to perform cross-domain user authentication. Independently, SFS [36], also based on public key cryptographic techniques, was designed to enable interoperability of different key management schemes. Each user of these systems is assumed to possess a certified public/private key pair. However, these systems were not designed specifically with scalability and parallel access in mind.

With the increasing deployment of highly distributed and network-attached storage systems, subsequent work, such as [4], [19], [55], focussed on scalable security. Nevertheless, these proposals assumed that a metadata server shares a group secret key with each distributed storage device. The group key is used to produce capabilities in the form of message authentication codes. However, compromise of the metadata server or any storage device allows the adversary to impersonate the server to any other entities in the file system. This issue can be alleviated by requiring that each storage device shares a different secret key with the metadata server. Nevertheless, such an approach restricts a capability to authorising I/O on only a single device, rather than larger groups of blocks or objects which may reside on multiple storage devices.

More recent proposals, which adopted a hybrid symmetric key and asymmetric key method, allow a capability to span any number of storage devices, while maintaining a reasonable efficiency-security ratio [29], [30], [31], [40]. For example, Maat [30] encompasses a set of protocols that facilitate (i) authenticated key establishment between clients and storage devices, (ii) capability issuance and renewal, and (iii) delegation between two clients. The authenticated key establishment protocol allows a client to establish and reuse a shared (session) key with a storage device. However, Maat and other recent proposals do not come with rigorous security analysis.

As with NFS, authentication in Hadoop Distributed File System is also based on Kerberos via GSS-API. Each HDFS client obtains a TGT that lasts for 10 hours and renewable for seven days by default; and access control is based on the Unix-style ACLs. However, HDFS makes use of the simple authentication and security layer (SASL) [38], a framework for providing a structured

interface between connection-oriented protocols and replaceable mechanisms.[11] In order to improve the performance of the KDC, the developers of HDFS chose to use a number of tokens for communication secured with an RPC digest scheme. The Hadoop security design makes use of Delegation Tokens, Job Tokens, and Block Access Tokens. Each of these tokens is similar in structure and based on HMAC-SHA1. Delegation Tokens are used for clients to communicate with the Name Node in order to gain access to HDFS data; while Block Access Tokens are used to secure communication between the Name Node and Data Nodes and to enforce HDFS file-system permissions. On the other hand, the Job Token is used to secure communication between the MapReduce engine Task Tracker and individual tasks. Note that the RPC digest scheme uses symmetric encryption and depending upon the token type, the shared key may be distributed to hundreds or even thousands of hosts [41].

9 Conclusions

We proposed three authenticated key exchange protocols for parallel network file system. 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. Hoon Wei Lim is the corresponding author.

Acknowledgments

The authors are thankful to Liqun Chen and Kenny Paterson for their helpful feedback on an earlier version of this paper. Hoon Wei Lim is the corresponding author.

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 *Proc. 4th USENIX Conf. File Storage Technol.*, Dec. 2005, pp. 59–72.

[2] C. Adams, "The simple public-key GSS-API mechanism (SPKM)," *Internet Eng. 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 *Proc. 5th Symp. Oper. Syst. Des. Implementation*, Dec. 2002, pp. 1–14.

[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 *Proc. 2nd Int. Conf. File Storage Technol.*, Mar. 2003, pp. 159–174.

[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," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

11. SASL's design is intended to allow new protocols to reuse existing mechanisms without requiring redesign of the mechanisms and allows existing protocols to make use of new mechanisms without redesign of protocols [38].

[6] Amazon simple storage service (Amazon S3) [Online]. Available: http://aws.amazon.com/s3/, 2014.

[7] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *Proc. 19th Int. Conf. Theory Appl. Cryptographic Techn.*, May 2000, pp. 139–155.

[8] D. Boneh, C. Gentry, and B. Waters, "Collusion resistant broadcast encryption with short ciphertexts and private keys," in *Proc. 25th Annu. Int. Conf. Adv. Cryptol.*, Aug. 2005, pp. 258–275.

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

[10] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.: Adv. Cryptology*, May 2001, pp. 453–474.

[11] CloudStore [Online]. Available: http://gcloud.civilservice.gov.uk/cloudstore/, 2014.

[12] Crypto++ 5.6.0 Benchmarks [Online]. Available: http://www.cryptopp.com/benchmarks.html, 2014.

[13] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Symp. Oper. Syst. Des. Implementation*, Dec. 2004, pp. 137–150.

[14] M. Eisler, "LIPKEY—A low infrastructure public key mechanism using SPKM," *Internet Eng. Task Force (IETF)*, RFC 2847, Jun. 2000.

[15] M. Eisler, "XDR: External data representation standard," *Internet Eng. Task Force (IETF)*, STD 67, RFC 4506, May 2006.

[16] M. Eisler, "RPCSEC_GSS version 2," *Internet Eng. Task Force (IETF)*, RFC 5403, Feb. 2009.

[17] M. Eisler, A. Chiu, and L. Ling, "RPCSEC_GSS protocol specification," *Internet Eng. Task Force (IETF)*, RFC 2203, Sep. 1997.

[18] S. Emery, "Kerberos version 5 generic security service application program interface (GSS-API) channel binding hash agility," *Internet Eng. Task Force (IETF)*, RFC 6542, Mar. 2012.

[19] M. Factor, D. Nagle, D. Naor, E. Riedel, and J. Satran, "The OSD security protocol," in *Proc. 3rd IEEE Int. Security Storage Workshop)*, Dec. 2005, pp. 29–39.

[20] Financial Services Grid Initiative [Online]. Available: http://www.fsgrid.com/, 2014.

[21] S. Ghemawat, H. Gobioff, and S. Leung, ," The Google file system," in *Proc. 19th ACM Symp. Oper. Syst. Principles*, Oct. 2003, pp. 29–43.

[22] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka, "A cost-effective, high-bandwidth storage architecture," *ACM SIGPLAN Notices*, vol. 33, no. 11, pp. 92–103, Nov. 1998.

[23] Hadoop Wiki [Online]. Available: http://wiki.apache.org/hadoop/PoweredBy, 2014.

[24] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West, "Scale and performance in a distributed file system," *ACM Trans. Comput. Syst.*, vol. 6, no. 1, pp. 51–81, Feb. 1988.

[25] F. Hupfeld, T. Cortes, B. Kolbeck, J. Stender, E. Focht, M. Hess, J. Malo, J. Marti, and E. Cesario, "The XtreemFS architecture—A case for object-based file systems in grids," *Concurrency Comput.: Prac. Exp.*, vol. 20, no. 17, pp. 2049–2060, Dec. 2008.

[26] Hadoop distributed file system [Online]. Available: http://hadoop.apache.org/hdfs/, 2014.

[27] J. Kubiatowicz, D. Bindel, Y. Chen, S. E. Czerwinski, P. R. Eaton, D. Geels, R. Gummadi, S. C. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Y. Zhao, "OceanStore: An architecture for global-scale persistent storage," in *Proc. 9th Int. Conf. Arch. Support Program. Language Oper. Syst.*, Nov. 2000, pp. 190–201.

[28] S. Langella, S. Hastings, S. Oster, T. Pan, A. Sharma, J. Permar, D. Ervin, B. B. Cambazoglu, T. M. Kurç, and J. H. Saltz, "Model formulation: Sharing data and analytical resources securely in a biomedical research grid environment," *J. Am. Med. Informatics Assoc.*, vol. 15, no. 3, pp. 363–373, May 2008.

[29] A. W. Leung and E. L. Miller, "Scalable security for large, high performance storage systems," in *Proc. ACM Workshop Storage Security Survivability*, Oct. 2006, pp. 29–40.

[30] A. W. Leung, E. L. Miller, and S. Jones, "Scalable security for petascale parallel file systems," in *Proc. ACM/IEEE Conf. High Perform. Netw. Comput.*, Nov. 2007, p. 16.

[31] H. W. Lim, "Key management for large-scale distributed storage systems," in *Proc. 6th Eur. Public Key Infrastructure Workshop*, Sep. 2010, pp. 99–113.

[32] J. Linn, "The Kerberos version 5 GSS-API mechanism," *Internet Eng. Task Force (IETF)*, RFC 1964, Jun. 1996.

[33] J. Linn, "Generic security service application program interface version 2, update 1," *Internet Eng. Task Force (IETF)*, RFC 2743, Jan. 2000.
[34] Libris Financial [Online]. Available: http://www.librisfinancial.com/stratolibris.html, 2014.
[35] Lustre [Online]. Available: http://www.lustre.org, 2014.
[36] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel, "Separating key management from file system security," in *Proc. 17th ACM Symp. Oper. Syst. Principles*, Dec. 1999, pp. 124–139.
[37] P. Mell and T. Grance, "The NIST definition of cloud computing," *Nat. Inst. Standards and Technology (NIST)*, Special Publication 800-145, Aug. 2011,
[38] A. Melnikov and K. Zeilenga, "Simple authentication and security layer (SASL)," *Internet Eng. Task Force (IETF)*, RFC 4422, Jun. 2006.
[39] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 1996.
[40] C. Olson and E. L. Miller, "Secure capabilities for a petabyte-scale object-based distributed file system," in *Proc. ACM Workshop Storage Secur. Survivability*, Nov. 2005, pp. 64–73.
[41] O. O'Malley, K. Zhang, S. Radia, R. Marti, and C. Harrell. (2009, Oct.). Hadoop security design, *Yahoo!* [Online]. Available: https://issues.apache.org/jira/secure/attachment/12428537/security-design.pdf
[42] S. Parker. (2012, Jun.) De-risking drug discovery with the use of cloud computing, *iSGTW* [Online]. Available: http://www.isgtw.org
[43] Parallel virtual file systems (PVFS) version 2 [Online]. Available: http://www.pvfs.org, 2014.
[44] A. Rosenthal, P. Mork, M. H. Li, J. Stanford, D. Koester, and P. Reynolds, "Cloud computing: A new business paradigm for biomedical information sharing," *J. Biomed. Informat.*, vol. 43, no. 2, pp. 342–353, Apr. 2010.
[45] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck, "Network file system (NFS) version 4 protocol," *Internet Eng. Task Force (IETF)*, RFC 3530, Apr. 2003.
[46] S. Shepler, M. Eisler, and D. Noveck, "Network file system (NFS) version 4 minor version 1 protocol," *Internet Eng. Task Force (IETF)*, RFC 5661, Jan. 2010.
[47] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and implementation of the Sun network filesystem," in *Proc. Summer USENIX Conf.*, Jun. 1985, pp. 119–130.
[48] F. B. Schmuck and R. L. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *Proc. 1st USENIX Conf. File Storage Technol.*, Jan. 2002, pp. 231–244.
[49] Sun Microsystems, Inc., "NFS: Network file system protocol specification," *Internet Eng. Task Force (IETF)*, RFC 1094, Mar. 1989.
[50] O. Tatebe, K. Hiraga, and N. Soda, "Gfarm grid file system," *New Generation Comput.*, vol. 28, no. 3, pp. 257–275, Jul. 2010.
[51] R. Thurlow, "RPC: Remote procedure call protocol specification version 2," *Internet Eng. Task Force (IETF)*, RFC 5531, May 2009.
[52] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. 7th Symp. Oper. Syst. Design Implementation*, Nov. 2006, pp. 307–320.
[53] B. Welch, M. Unangst, Z. Abbasi, G. A. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, "Scalable performance of the Panasas parallel file system," in *Proc. 6th USENIX Conf. File Storage Technol.*, Feb. 2008, pp. 17–33.
[54] B. S. White, M. Walker, M. Humphrey, and A. S. Grimshaw, "LegionFS: A secure and scalable file system supporting cross-domain high-performance applications," in *Proc. ACM/IEEE Conf. Supercomput.*, Nov. 2001, p. 59.
[55] Y. Zhu and Y. Hu, "SNARE: A strong security scheme for network-attached storage," in *Proc. 22nd Symp. Reliable Distrib. Syst.*, Oct. 2003, pp. 250–259.

Dr. Hoon Wei Lim is a senior research fellow at the National University of Singapore (NUS). His recent research interests have been centered around data security and privacy. He is particularly interested in data protection, privacy, analytic and compliance related problems for cloud and mobile platforms. In the past, Lim was a research fellow at Nanyang Technological University (NTU), Singapore. He had also worked as a researcher for SAP Research, France. Lim received a Ph.D in Information Security from Royal Holloway, University of London, UK.

Dr. Guomin Yang received his Ph.D from the Department of Computer Science, City University of Hong Kong in 2009. From 2009 to 2012, he worked as a Research Scientist at the Temasek Laboratories, National University of Singapore. He is currently a Senior Lecture and DECRA Fellow at the School of Computer Science and Software Engineering, University of Wollongong, Australia.