File Sharing Based on Authenticated Key Exchange Protocol in Parallel Network

H Shyam Sundar
Assistant Professor
Department of CSE, VITS (N6), Karimnagar, JNTUH Hyderbad, TS, India
shyamhanmandla@gmail.com

N Sai Chander Reddy
B.Tech IV CSE Student, VITS (N6), Karimnagar, JNTUH Hyderbad, TS, India
saichanderreddy96@gmail.com

G Kavya
B.Tech IV CSE Student, VITS (N6), Karimnagar, JNTUH Hyderbad, TS, India
kavyagadila20@gmail.com

K Shruthi
B.Tech IV CSE Student, VITS (N6), Karimnagar, JNTUH Hyderbad, TS, India
kamreddyshruthi@gmail.com

Abstract—We study the problem of key establishment for se-cure 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, net-work file systems, forward secrecy, key escrow.

I. 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

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 Sys-tem (HDFS) [26], Amazon S3 File System [6], and Cloud-Store [11]. This, in turn, has accelerated the wide-spread use of distributed and parallel computation on large datasets in many organizations. Some notable users of the HDFS include AOL, Apple, eBay, Facebook, Hewlett-Packard, IBM, LinkedIn, Twitter, and Yahoo! [23].

In this work, we investigate the problem of secure many-to-many communications in large-scale network file systems that support parallel access to multiple storage devices. That is, we consider a communication model where there are a large number of clients (potentially hundreds or thousands) accessing multiple remote and distributed storage devices (which also may scale up to hundreds or thousands) in parallel. Particularly, we focus on how to exchange key materials and establish *parallel secure sessions* between the clients and the storage devices in the parallel Network File System (pNFS) [46]—the current Internet standard—in an efficient and scalable manner. The development 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 dy-namics, and semiconductor manufacturing. In these environ-ments, 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.

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 II):

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

The main results of this paper are three new provably secure authenticated key exchange protocols. Our protocols, progressively designed to achieve each of the above properties, demonstrate the trade-offs between efficiency and security. We show that our protocols can reduce the workload of the metadata server by approximately half compared to the current Kerberos-based protocol, while achieving the desired security properties and keeping the 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 cur-rent Kerberos-based protocol in pNFS for establishing secure channels in parallel. In Section III, we describe the threat model for pNFS and the existing Kerberos-based protocol. In Section IV, we present our protocols that aim to address the current limitations. We then provide formal security analyses of our protocols under an appropriate security model, as well as performance evaluation in Sections VI and VII, respectively. In Section VIII, we describe related work, and finally in Section IX, we conclude and discuss some future work.

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 trans-port 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:

pNFS separates the file system protocol processing into two parts: metadata processing and data processing. Metadata is in-formation 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.

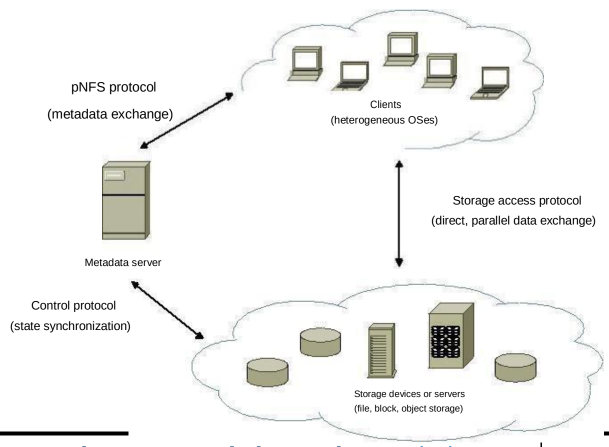

Fig. 1. The conceptual model of pNF

C. Current Limitations

The current design of NFS/pNFS focuses on *interoperabil-ity*, instead of efficiency and scalability, of various mechanisms to provide basic security. Moreover, key establishment be-tween 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 de-vices (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.

III. Preliminaries

A. Notation

We let M denote a metadata server, C denote a client, and S denote a storage device. Let entity $X; Y\ 2\ fM; C;$ Sg, 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 $E(K; m)$ 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 to denote a layout. We may introduce other notation as required.

B. Threat Assumptions

Existing proposals [19], [40], [29], [30], [31] 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 some-times 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.

C. Kerberos-based pNFS Protocol

For the sake of completeness, we describe the key establish-ment protocol[4] recommended for pNFS in RFC 5661 between a client C and n storage devices S_i, for 1 i n, through a metadata server M in Figure 2. We will compare the efficiency of the pNFS protocol against ours in Section VII.

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 Figure 2, we assume that C and M have already setup a secure channel through LIPKEY (as described in Section II-B).

Once C has been authenticated by M and granted ac-cess to $S_1; : : : ; S_n$, it receives a set of service tickets $E(K_{MS_i}\ ; ID_C\ ; t;$ $sk_i)$, session keys $sk\ i$, and layouts[5] i (for all $i\ 2$ [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 $E(K_{CT}\ ; sk_1; : : : ; sk_n)$. 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 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.

IV. 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] 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.

[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].

(1) $C ! M :\ ID_C$

(2) $M ! C :\ E(K_C ; K_{CT}),\ E(K_T ; ID_C ; t; K_{CT})$

(3) $C ! T :\ ID_{S_1} ; : : : ; ID_{S_n} ,\ E(K_T ; ID_C ; t; K_{CT}),\ E(K_{CT} ; ID_C ; t)$

(4) $T ! C :\ 1; : : : ; n,\ E(K_{MS_1} ; ID_C ; t; sk_1); : : : ; E(K_{MS_n} ; ID_C ; t; sk_n),\ E(K_{CT} ; sk_1; : : : ; sk_n)$

(5) $C ! S_i :\ i; E(K_{MS_i} ; ID_C ; t; sk_i),\ E(sk_i; ID_C ; t)$

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

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

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 solu-tions, 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 authen-tication 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 *Si* each now chooses a secret value (that is known only to itself) and pre-computes a Diffie-Hellman key component. A session key is then generated from both the Diffie-Hellman components. Upon expiry of a time period *v*, the secret values and Diffie-Hellman key components are permanently

  erased, such that in the event when either *C* or *Si* is compromised, the attacker will no longer have access to the 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*), 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 *Si* to share some initial key material in the form of a Diffie-Hellman key. In Phase II, the initial shared key is then used to derive session keys in the form of a keyed hash chain. Since a hash value in the chain does not reveal information about its pre-image, the associated session key is forward secure.

V. 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 key

Phase I – For each validity period *v*:

(1) *C ! M* : *IDC* , *E*(*KCM* ; *KCS*1 ; : : : ; *KCSN*)

(2) *M ! C* : *E*(*KMS*1 ; *IDC* ; *IDS*1 ; *v; KCS*1); : : : ; *E*(*KMSN* ; *IDC* ; *IDSN* ; *v; KCSN*)

Phase II – For each access request at time *t*:

(1) *C ! M* : *IDC* , *IDS*1 ; : : : ; *IDSn*

(2) *M ! C* : 1; : : : ; *n*

(3) *C ! Si* : *i*; *E*(*K MSi* ; *IDC* ; *IDSi* ; *v; KCSi*), *E*(*ski* 0 ; *IDC* ; *t*)

(4) *Si ! C* : *E*(*ski* 0 ; *t* + 1)

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

materials *KCS*1 ; : : : ; *KCSN* before it can access any of the *N* storage device *Si* (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 chan-nel associated with key *K* CM established using the existing methods as described in Section II-B. *M* then issues an au-thentication token of the form *E*(*KMSi* ; *IDC* ; *IDSi* ; *v; KCSi*) for each key material if the associated storage device *Si* has not been revoked. 7 This completes Phase I of the protocol. From

this point onwards, any request from *C* to access *Si* is considered part of Phase II of the protocol until *v* expires.

When *C* submits an access request to *M*, the request con-tains all the identities of storage devices *Si* for 1 *i n N* that *C* wishes to access. For each *Si*, *M* issues a layout *i*. *C* then forwards the respective layouts, authentication tokens (from Phase I), and encrypted messages of the form *E*(*ski* 0 ; *ID* ; *t*) to all *n* storage devices.

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

1) check if the layout *i* is valid;
2) decrypt the authentication token and recover key *KCSi* ;
3) compute keys *ski* z = *F* (*KCSi* ; *IDC* ; *IDSi* ; *v; sid; z*) for *z* = 0; 1;
4) decrypt the encrypted message, check if *IDC* matches the identity of *C* and if *t* is within the current validity period *v*;
5) if all previous checks pass, *Si* replies *C* with a key confirmation message using key *ski* 0 .

At the end of the protocol, *ski* 1 is set to be the session key for securing communication between *C* and *Si*. 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.

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 *Si* is required to pre - distribute some key material to *M* at Phase I of the protocol.

Let *g* x 2 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 *Si* 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 *KMSi* 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 *KMSi* as the master key. For example, the encryption key can be set to be *F* (*KMSi* ; "enc"), while the MAC key can be set to be *F* (*KMSi* ; "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. *Si* must verify the authentication token to ensure the integrity of *g* c . Here *C* and *Si* compute *ski* z for *z* = 0; 1 as follow:

$$ski^{z} = F(g^{cs_i}; IDC; IDS_i; g^{c}; g^{s_i}; v; sid; z):$$

At the end of the protocol, *C* and *Si* share a session key *ski* 1 .

Note that since *C* distributes its chosen Diffie-Hellman value *g* c during each protocol run (in Phase II), each *Si* needs to store only its own secret value *si* 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 *si*, respectively, from

Clearly, *M* does not learn anything about *skiz* unless it

7 Here *KMSi* is regarded as a long-term symmetric secret key shared be-tween *M* and *Si*. Also, we use authenticated encryption instead of encryption only encryption for security reasons. This will be clear in our security analysis.

colludes with the associated *C* or *Si*, and thus achieving escrow-freeness.

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

[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.

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 filesystem 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].

IX. Conclusions

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

Acknowledgement

We are thankful to Liqun Chen and Kenny Paterson for their helpful feedback on an earlier version of this paper.

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

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

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

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

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

[10] R. Canetti and H. Krawczyk. Analysis of key-exchange protocolsd their use for building secure channels. In *Advances in Cryptology – Proceedings of EUROCRYPT*, pages 453–474. Springer LNCS 2045, May 2001.

[11] CloudStore. http://gcloud.civilservice.gov.uk/cloudstore/.

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

[13] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI)*, pages 137–150. USENIX Association, Dec 2004.

[14] M. Eisler. LIPKEY - A Low Infrastructure Public Key mechanism using SPKM. *The Internet Engineering Task Force (IETF)*, RFC 2847, Jun 2000.

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

[16] M. Eisler. RPCSEC GSS version 2. *The Internet Engineering Task Force (IETF)*, RFC 5403, Feb 2009.

[17] M. Eisler, A. Chiu, and L. Ling. RPCSEC GSS protocol specification. *The Internet Engineering 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. *The Internet Engineering Task Force (IETF)*, RFC 6542, Mar 2012.

[19] M. Factor, D. Nagle, D. Naor, E. Riedel, and J. Satran. The OSD security protocol. In *Proceedings of the 3rd IEEE International Security in Storage Workshop (SISW)*, pages 29–39. IEEE Computer Society, Dec 2005.

[20] Financial Services Grid Initiative. http://www.fsgrid.com/.

[21] S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 29–43. ACM Press, Oct 2003.

[22] G.A. Gibson, D.F. Nagle, K. Amiri, J. Butler, F.W. Chang, H. Go-bioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A cost-effective, high-bandwidth storage architecture. *ACM SIGPLAN Notices*, 33(11):92–103. ACM Press, Nov 1998.

[23] Hadoop Wiki. http://wiki.apache.org/hadoop/PoweredBy.

[24] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satya-narayanan, R.N. Sidebotham, and M.J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems (TOCS)*, 6(1):51–81. ACM Press, 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 and Computation: Practice and Experience (CCPE)*, 20(17):2049–2060. Wiley, Dec 2008.

[26] Hadoop distributed file system. http://hadoop.apache.org/hdfs/.

[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 *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 190–201. ACM Press, Nov 2000.

[28] S. Langella, S. Hastings, S. Oster, T. Pan, A. Sharma, J. Permar, D. Ervin, B.B. Cambazoglu, T.M. Kurc¸, and J.H. Saltz. Model formu-lation: Sharing data and analytical resources securely in a biomedical research grid environment. *Journal of the American Medical Informatics Association (JAMIA)*, 15(3):363–373. BMJ, May 2008.

[29] A.W. Leung and E.L. Miller. Scalable security for large, high perfor-mance storage systems. In *Proceedings of the ACM Workshop on Storage Security and Survivability (StorageSS)*, pages 29–40. ACM Press, Oct 2006.

[30] A.W. Leung, E.L. Miller, and S. Jones. Scalable security for petascale parallel file systems. In *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing (SC)*, page 16. ACM Press, Nov 2007.

[31] H.W. Lim. Key management for large-scale distributed storage systems. In *Proceedings of the 6th European Public Key Infrastructure Workshop (EuroPKI)*, pages 99–113. Springer LNCS 6391, Sep 2010.

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

[33] J. Linn. Generic security service application program interface version 2, update 1. *The Internet Engineering Task Force (IETF)*, RFC 2743, Jan 2000.

[34] Libris Financial. http://www.librisfinancial.com/stratolibris.html.

[35] Lustre. http://www.lustre.org.