ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

*(An ISO 3297: 2007 Certified Organization)*

*Website: www.ijircce.com*

Vol. 5, Special Issue 1, March 2017

# Authenticated Communication between Client and Storage Devices

[1]S.Dhanalakshmi, [2]A.Marisbala, [3]M.Subalakshmi, [4]R.Kalaivani

[1] [2] [3]UG Student, P S R Rengasamy College of Engineering for Women, Sivakasi, India

[4] Assistant Professor, P S R Rengasamy College of Engineering for Women, Sivakasi, India

**ABSTRACT**: In a parallel network file system, direct communication is established between client and the server as well as the storage devices by using parallel session keys. Key establishment and key agreement are the problems in secure communication. There are various authentication protocols are established to overcome these problems. One of the protocols to overcome these issues is PAKE which incorporates online and offline password guessing, concurrent sessions, forward secrecy, server compromise, and loss of session keys. Our review of the existing three authenticated key exchange protocols has a number of limitations: (I) Due to the increase in size of a network the workload of a server the performance of the server will degrade, (ii) forward secrecy is not achieved because of the size of the network and (iii) it leads to key escrow. So we propose authenticated key exchange protocol to overcome the above issues.

**KEYWORDS**: Authenticated key exchange, forward secrecy, parallel session, key escrow, network file system.

## I. INTRODUCTION

Network Security is a branch of computer science that consists of policies and practices that involves in securing a computer network and to prevent, data theft, unauthorized access, network misuse and data modification. Network Security is in preventing DOS (Denial Of Service) attacks and assuring continuous service for licit network users. Denial of service is polished by flooding the targeted machine or resource with surplus requests to overload systems and prevent some or all legal requests from being fulfilled. Network Security involves mechanisms and protocol to protect data and network devices from internal and external threats. Parallel network file system is the extended version of network file system. It is used to store application data persistently. Usually large datasets that cannot fit into the memory. It provides global shared namespace (files, directories). It is designed for concurrent access by multiple clients. It is used to increase the performance and also operate over high speed networks such as IB, Myrinet, Portals. It increases the I/O bandwidth by allowing concurrent access. Some of the examples of high performance parallel network file system that are in production use are the Google File System (GoogleFS), IBM General parallel file system (GPFS), Parallel Virtual File System (PVFS), Lustre, Panasas Due to the emergence of clouds, Map Reduce programming model, clusters and high performance computing Hadoop Distributed File System were developed. Some important users of HDFS are Apple, AOL, eBay, Facebook, IBM, Hewlett-Packard, Twitter, LinkedIn and Yahoo.

In this work, we investigate the problems in parallel network file system using Kerberos based protocol to establish the parallel session keys between clients and storage devices. We consider a communication model that consists of large number clients accessing multiple remote and distributed storage devices in parallel. We focus on how to exchange key and to establish parallel sessions between the clients and the storage devices in the parallel Network File System (PNFS). The development of PNFS is driven by Panasas, Netapp, IBM, Sun, EMC and Umich/CITI, and thus it shares many common features of network file system.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 1, March 2017

Our primary goal in this work is to design secure authenticated key exchange protocols and to meet specific requirements of PNFS. Especially, we attempt to meet the following properties of PNFS:

*** Scalability**- Scalability is the capability of a system, process or network to handle a growing amount of work, or its potential to be enlarged in order to accommodate that growth. To achieve scalability bottleneck problem should be avoided.

*** Forward Secrecy**- it generates one random secret key per session to complete a key agreement. forward secrecy only protects keys, not the ciphers themselves.

***Escrow free**- Key escrow systems provide a backup for cryptographic keys. Escrow systems are somewhat risky because a third party is involved.

Our Protocols are designed to achieve the above properties and keeping the computational overhead at the clients and the storage devices at a reasonably low level. Our protocols can reduce the workload of the metadata server by facilitating access request from client to multiple storage devices. In the next section, we provide some basics of Parallel Network File System and describe its existing security mechanisms associated with secure communication between clients and distributed storage devices. we identify the limitations of the current Kerberos-based protocol in PNFS for organizing secure sessions in parallel. In Section III, we describe the drawbacks of already existing Kerberos-based protocol in PNFS. In Section IV, we describe our protocols that aim to address the current limitations of existing system. We then provide security analyses of our protocols under an suitable 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 – NETWORK FILE SYSTEM

An Internet Standard is a specification of a technology or methodology relevant to the Internet. Internet Standards are created and published by the Internet Engineering Task Force (IETF). The Network File System is a client-server application that grant a computer user view and deliberately store and update files on a remote computer. The NFS protocol is one of several distributed file system standards for Network-Attached Storage (NAS). Some of the most notable benefits of NFS are:

- Local workstations use limited disk space because commonly used data can be stored on a single machine and still persistent access to others over the network.
- There is no need for users to have independent home directories on every network machine. Home directories could be set up on the NFS server and made possible throughout the network.
- Storage devices such as CDROM drives, floppy disks and USB Thumb drives can be used by other machines on the network. This may reduce the number of portable media drives throughout the network.

### *A. SECURITY CONSIDERATION:*

Earlier versions of Network File System focused on simplicity and efficiency, and designed to work well on local net-works and intranets. The later versions aim to improve access and performance within the Internet environment. Among many other security issues, server and user 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 financial data processing & analysis and drug simulation & discovery. 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 to transmit between nodes within the system.

The RPCSEC GSS framework 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

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

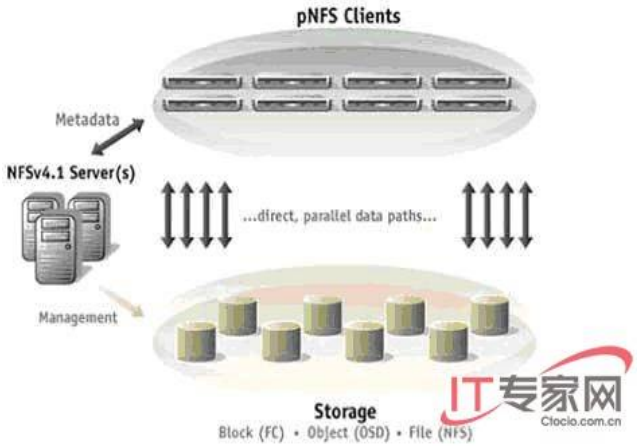
Website: www.ijircce.com

Vol. 5, Special Issue 1, March 2017

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.

### *B. KERBEROS & LIPKEY*

In NFSv4, the Kerberos version and the Low Infrastructure Public Key (LIPKEY) GSS-API mecha-nisms 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.




### *C. CURRENT LIMITATIONS:*

The current design of NFS/pNFS focuses on *interoperability*, instead of efficiency and scalability, of various mechanismsto 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.

## III. OVERVIEW OF OUR PROTOCOL

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*, wethen 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.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 1, March 2017

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.

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.

***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. For each validity period $v$, $C$ must first pre-compute a set of key materials $K_{CS1}; : : : ; K_{CSN}$ before it can access any of the $N$ storage device $S_i$(for1 $i$ $N$). The key materials aretransmitted 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 $E(K_{MSi}$ ; $ID_C; ID_{Si}; v; K_{CSi})$ 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 con-tains all the identities of storage devices $S_i$ for 1 $i$ $n$ $N$ that $C$ wishes to access. For each $S_i$, $M$ issues a layout $_i$. $C$ then forwards the respective layouts, authentication tokens (from Phase I), and encrypted messages of the form $E(sk_i^0; ID_C ; t)$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 $_i$ is valid;
2) decrypt the authentication token and recover key $K_{CSi}$ ;
3) compute keys $sk_i^z = F(K_{CSi}$ ; $ID_C; ID_{Si}; v; sid; z)$ for $z = 0; 1$;
4) decrypt the encrypted message, check if $ID_C$ matches the identity of $C$ and if $t$ is within the current validity period $v$;
5) if all previous checks pass, $S_i$ replies $C$ with a key confirmation message using key $sk_i^0$.

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 *sid* in our protocol is uniquely generated for each session at the application layer, for example through the GSS-API.

***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. 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) timeperiods $v'$ (for $v' < v$) will not be affected.

We now employ a Diffie-Hellman key agreement technique to both provide forward secrecy and prevent key escrow. In this protocol, each $S_i$ is required to pre-distribute some key material to $M$ at Phase I of the protocol.
Let $g^x2$ G denote a Diffie-Hellman component, where G is an appropriate group generated by $g$, and $x$ is a number randomly chosen by entity $X2\ fC; Sg$. Let $(k; m)$ denote a secure MAC scheme that takes as input a secret key $k$ and a target message $m$, and output a MAC tag. Our partially forward secure protocol is specified in Figure 4.
At the beginning of each $v$, each $S_i$ that is governed by 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 .[8] At the end of Phase I, $C$ receives all the key components corresponding to all $N$ storage devices that it may access within time period $v$, and a set of authentication tokens of the form $(K_{MSi}$ ; $ID_C; ID_{Si}; v; g^c; g^{si})$. We note that for ease of exposition, we use the same key $K_{MSi}$ for encryption in step

(1) and MAC in step (2). In actual implementation, however, we assume that different keys are derived for encryption and MAC, respectively, with $K_{MSi}$ as the master key. For example, the encryption key can be set to be $F(K_{MSi}$ ; "enc"), while the MAC key can be set to be $F(K_{MSi}$ ; "mac").

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 1, March 2017

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 $z = 0$; 1as follow:

$skiz = F\ (gcsi\ ;\ IDC\ ;\ IDSi\ ;\ gc;\ gsi\ ;\ v;\ sid;\ z)$:

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.

***pNFS-AKE III:*** 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 computationaloverhead 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. 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 $g^{csi}$.Moreover, $C$ and $S_i$ set $F_1(g^{csi}\ ;\ ID_C;\ ID_{Si};\ v)$ 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:

$$sk_i^{j,z} = F(K_{CS_i}^{j-1};\ ID_2\ ;\ ID_C\ ;\ ID_{S_i}\ ;\ j;\ sid;\ z)$$

where $j$ 1 is an increasing counter denoting the $j$-th session between $C$ and $S_i$ with session key $sk_i^{j,1}$. 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 $ski^{j,1}$, the adversary cannot derive $KCS^{j-1}$ 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.

## IV. PERFORMANCE EVALUATION

### A. COMPUTATIONAL OVERHEAD:

We consider the computational overhead for $w$ access requests over time period $v$ for a metadata server $M$, a client $C$, and storage devices $S_i$for $i$ 2 [1; $N$]. We assume that alayout is of the form of a MAC, and the computational cost for authenticated symmetric encryption $E$ is similar to that for the non-authenticated version $E$. Table I gives a comparison between Kerberos-based pNFS and our protocols in terms of the number of cryptographic operations required for executing the protocols over time period $v$.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 1, March 2017

TABLE I

| 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 | $N + 1$ | $2N$ | $3N + 1$ |

To give a more concrete view, Table II provides some estimation of the total computation times in seconds (s) for each protocol by using the Crypto++ benchmarks obtained on an Intel Core 2 1.83 GHz processor under Windows Vista in 32-bit mode. We choose AES/CBC (128-bit key) for encryption, AES/GCM (128-bit, 64K tables) for authenticated encryption, HMAC(SHA-1) for MAC, and SHA-1 for key derivation. Also, Diffie-Hellman exponentiations are based on DH 1024-bit key pair generation. Our estimation is based on a fixed message size of 1024 bytes for all cryptographic operations, and we consider the following case:

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

Table II shows that our protocols reduce the workload of $M$ in the existing Kerberos-based protocol by up to approximately 54%. This improves the scalability of the metadata server considerably. The total estimated computational cost for $M$ for serving $10^5$ clients is 8:02 $10^4$ s ( 22.3 hours) in Kerberos-based pNFS, compared with 3:68 $10^4$ s ( 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 I that the workload of $M$ is always reduced by roughly half for any values of ($w$; $n$; $N$). The scalability of our protocols from the server's perspective in terms of supporting a large number of clients is further illustrated in the left graph of Figure 6 when we consider each client requesting access to an average of $n = 10^3$storage devices.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 1, March 2017

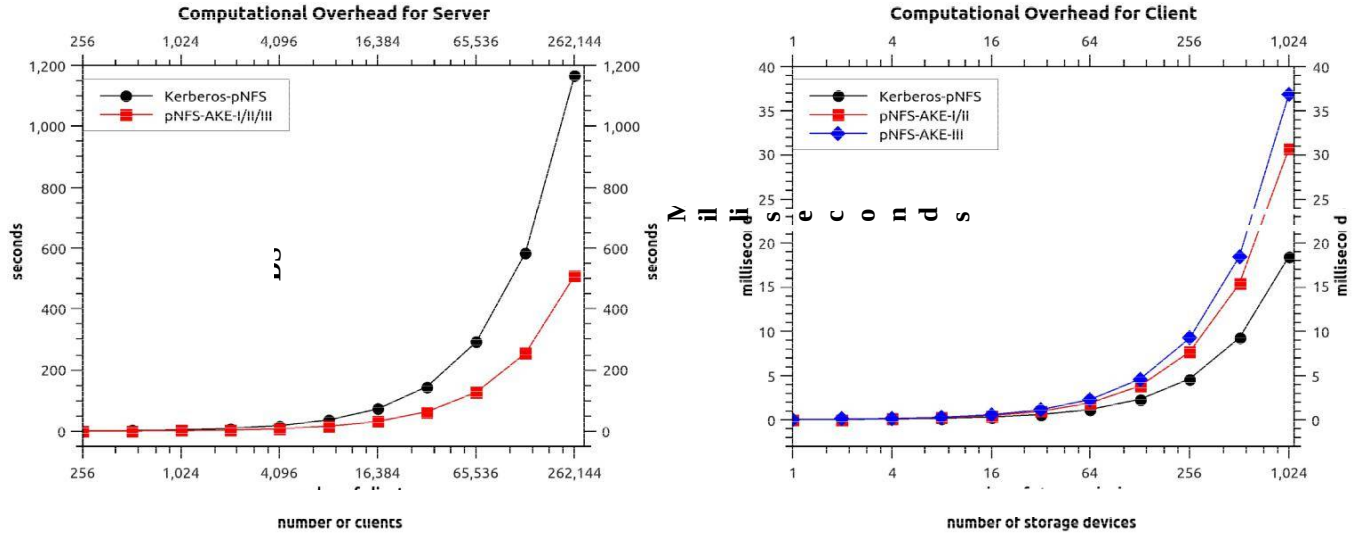
TABLE II

| Protocol | FFS | EF | *M* | *C* | $^{S}i$ |
|---|---|---|---|---|---|
| Kerberos-pNFS | | | ***8:02*** ***4*** ***10*** | 0.90 | 17.00 |
| pNFS-AKE-I | | | ***3:68*** ***10*** | 1.50 | 23.00 |
| pNFS-AKE-II | | ✓ | ***3:82*** ***10*** | 2.40 | 256.00 |
| pNFS-AKE-III | ✓ | ✓ | ***3:86*** ***10*** | 2.71 | 39.60 |

Moreover, the additional overhead for *C* (and all $S_i$) for achieving full forward secrecy and escrow-freeness using our techniques are minimal. The right graph of Figure 6 shows that our pNFS-AKE-III protocol has roughly similar computational overhead in comparison with Kerberos-pNFS when the num-ber 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*.
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 V-B (see Section VII-C for further discussion on key storage). Nevertheless, 256 s is an average computation time for $10^3$ storage devices over time period *v*, and thus the average computation time for a storage device is still reasonably small, *i.e.* less than 1/3 of a second over time period *v*. Moreover, wecan 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 $g^{csi}$ value for each time period *v*.

### *B. COMMUNICATION OVERHEAD:*

Assuming fresh session keys are used to secure communica-tions 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.



ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 1, March 2017

## V. 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.

## 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 *Proceedingsof 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). *TheInternet 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 OperatingSystem Design and Implementation (OSDI)*. USENIX Association, Dec2002.

[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 2ndInternational 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 inCryptology – Proceedings of CRYPTO*, pages 258–275. Springer LNCS3621, 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 protocols and 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 OperatingSystem Design and Implementation (OSDI)*, pages 137–150. USENIXAssociation, 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 InternetEngineering Task Force (IETF)*, STD 67, RFC 4506, May 2006.

[16] M. Eisler. RPCSEC GSS version 2. *The Internet Engineering TaskForce (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 InternetEngineering 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 Securityin Storage Workshop (SISW)*, pages 29–39. IEEE Computer Society, Dec2005.

[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 SystemsPrinciples (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 andExperience (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, C. Geels, R. Gummadi, S.C. Rhea, H. Weatherspoon, W. Weimer,Wells, and B.Y. Zhao. OceanStore: An architecture for global-scale persistent storage. In *Proceedings of the 9th International Conferenceon 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 InformaticsAssociation (JAMIA)*, 15(3):363–373. BMJ, May 2008.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

*(An ISO 3297: 2007 Certified Organization)*

*Website: www.ijircce.com*

Vol. 5, Special Issue 1, March 2017

[29] A.W. Leung and E.L. Miller. Scalable security for large, high perfor-mance storage systems. In *Proceedings of the ACM Workshop on StorageSecurity and Survivability (StorageSS)*, pages 29–40. ACM Press, Oct2006.
[30] A.W. Leung, E.L. Miller, and S. Jones. Scalable security for petascale parallel file systems. In *Proceedings of the ACM/IEEE Conference onHigh Performance Networking and Computing (SC)*, page 16. ACMPress, 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 InternetEngineering 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.
[36] D. Mazieres,` M. Kaminsky, M.F. Kaashoek, and E. Witchel. Separating key management from file system security.