SeMiNAS: A Secure Middleware for Wide-Area Network-Attached Storage

Ming Chen, Erez Zadok
Stony Brook University
mchen, ezk@cs.stonybrook.edu

Arun Olappamanna Vasudevan
Nutanix Inc.
aov@nutanix.com

Kelong Wang
EMC DSSD
kelong@dssd.com

Abstract

Utility computing is being gradually realized as exemplified by cloud computing. Outsourcing computing and storage to global-scale cloud providers benefits from high accessibility, flexibility, scalability, and cost-effectiveness. However, users are uneasy outsourcing the storage of sensitive data due to security concerns. We address this problem by presenting *SeMiNAS*—an efficient middleware system that allows files to be securely outsourced to providers and shared among geo-distributed offices. SeMiNAS achieves end-to-end data integrity and confidentiality with a highly efficient authenticated-encryption scheme. SeMiNAS leverages advanced NFSv4 features, including compound procedures and data-integrity extensions, to minimize extra network round trips caused by security meta-data. SeMiNAS also caches remote files locally to reduce accesses to providers over WANs. We designed, implemented, and evaluated SeMiNAS, which demonstrates a small performance penalty of less than 26% and an occasional performance boost of up to 19% for Filebench workloads.

Categories and Subject Descriptors D.4.3 [*File Systems Management*]: Distributed File Systems

1. Introduction

Cloud storage has demonstrated many benefits of utility computing, including high accessibility (from multiple devices, at multiple locations), availability, flexibility, scalability, and cost-effectiveness [3, 39, 68]. However, storing data in intangible devices of external providers is a worrisome decision intensified by high-profile incidents such as silent data corruption [61], theft of patient records [43], and leakage of intimate photos of celebrities [2]. To ease these security concerns, many cloud users manually encrypt and generate integrity checksums of data. These security mechanisms can cause significant performance penalty and might be too expensive to justify storing encrypted data in the cloud [9]. Thus it is critical for security mechanisms to be efficient.

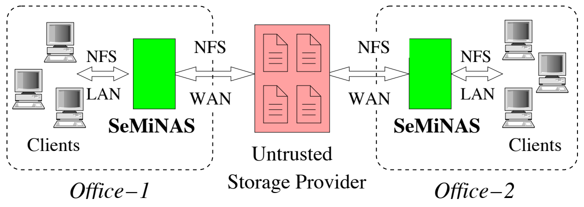


Figure 1. SeMiNAS high-level architecture. Each geographic office has a trusted SeMiNAS proxy to protect clients from the untrusted storage provider. SeMiNAS can securely share files among geo-distributed offices.

To allow files to be securely and efficiently stored in clouds, we present *SeMiNAS*—an on-premises middleware that seamlessly secures accesses to remote storage servers over WANs. As shown in Figure 1, SeMiNAS inherits many advantages from the popular middleware architecture, as exemplified by network firewalls. For instance, SeMiNAS can protect a large number of clients by consolidating a small number of SeMiNAS proxies; SeMiNAS also minimizes migration costs by requiring only minimal configuration changes to existing clients and servers.

SeMiNAS provides end-to-end data integrity and confidentiality using authenticated encryption before sending data to storage providers. Data stays encrypted until SeMiNAS retrieves and decrypts the data on clients' behalf. End-to-end integrity and confidentiality protects data not only from attacks during data transmission over WANs, but also from misbehaving providers. Using a simple and robust key exchange scheme, SeMiNAS can share files securely among geo-distributed offices without relying on any trusted third-party or secret channel for key management.

SeMiNAS ensures its security scheme is efficient with three mechanisms: (1) SeMiNAS stores Message Authentication Codes (MAC) together with file data using NFS data-integrity extension [46], so that no extra WAN round trips are needed for checking and updating MACs. (2) SeMiNAS uses NFSv4's compound procedures to combine operations on file headers (added by SeMiNAS for distributing file keys

SYSTOR'16, June 06 - 08, 2016, Haifa, Israel.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-4381-7/16/06…$15.00.
http://dx.doi.org/10.1145/2928275.2928282

securely among proxies) with small meta-data operations. Therefore, SeMiNAS does not introduce any extra WAN round trips for its security meta-data. (3) SeMiNAS contains a persistent write-back cache that stores recently used data and coalesces writes to the server; this further reduces the communication to remote servers. With these three optimizations, SeMiNAS is able to provide integrity and confidentiality with a small overhead of less than 26%.

Overall, we have three contributions: **(1)** a middleware system that allows clients to securely and efficiently store and share files in remote NAS providers; **(2)** a study of leveraging NFSv4 compound procedures for a highly efficient security scheme; and **(3)** an implementation and evaluation of NFSv4 end-to-end Data Integrity eXtension (DIX) [46].

The rest of this paper is organized as follows. Section 2 presents the background and motivation behind SeMiNAS. Section 3 and Section 4 describe its design and implementation, respectively. Section 5 evaluates SeMiNAS under different networks and workloads. Section 6 discusses related work and Section 7 concludes.

2. Background and Motivation

We present the background and motivation behind SeMiNAS by answering two questions: (1) Why we need yet another cryptographic file system on an untrusted server—an area studied more than a decade ago? (2) Why SeMiNAS uses an NFS back-end instead of a key-value object back-end?

A revisit of cryptographic file systems. Utility computing requires storing data in external providers. Consequently, security concerns arise because of the opaque and multi-tenant nature of external servers, as well as the large exploitation surface area of public wide-area networks. Using cryptographic file systems to securely outsource data has been studied before in SFS [37], Cepheus [29], SiRiUS [24], SUNDR [33], Iris [58], and others [19, 20, 28, 62, 70]. However, a revisit of these studies is needed for three reasons.

First, modern cryptographic file systems should protect data not only from potentially malicious servers, but also from the deep storage stack (with multiple layers of virtualization, software, firmware, hardware, and networking), which is much more complex and error-prone than before [4, 34]. Data Integrity eXtensions (DIX) [13] is a growing trend of making the once hidden Data Integrity Fields (DIF) of storage devices available to applications. This can help keep data safe from both malicious servers and a misbehaving storage stack. By providing an eight-byte out-of-band storage for security checksums per 512-byte sector, DIX can protect the whole data path from applications all the way down to physical storage media.

Second, newer and more powerful networking storage protocols have emerged, particularly NFSv4 [54, 55, 59]. Compared to its previous versions, NFSv4 is superior not only in performance, scalability, and manageability [8, 38], but also in security with RPCSEC_GSS [16] and ACLs [54, 55, 59]. Moreover, with advanced features including compound procedures and delegations, NFSv4 provides great opportunities for making cryptographic file system flexible and efficient when storage servers are remote over WANs.

Third, the performance penalty of modern cryptographic file systems should be small enough to maintain a lower total cost of ownership—a key incentive for outsourcing storage. Some researchers [9] argued that encrypting data in cloud was too expensive, whereas others [1, 62] claimed new hardware acceleration makes encryption viable and cheap for cloud storage. These debates highlight the importance of reducing performance overhead when securing cloud storage. Therefore, SeMiNAS strives for high performance as well as security, whereas many prior systems [19, 24, 33] sacrificed performance by up to 90% for security.

An NFS vs. a key-value object back-end. Currently, most cloud storage vendors provide key-value object stores. However, SeMiNAS uses an NFS back-end instead for four reasons. First, we believe that cloud storage is still in its early age and future cloud storage will offer richer, file-system APIs in addition to key-value object APIs. Key-value object stores are popular now primarily because of simplicity. File-systems APIs in clouds are likely to grow in popularity as cloud applications demand more functions from cloud storage vendors. This is a trend as seen by the recent cloud offering of the NFSv4-based Amazon Elastic File System [26].

Second, the open, pervasive, and standard NFS API has many advantages over vendor-specific object APIs. NFS is compatible with the POSIX standard, and most applications based on direct-attached storage can continue to work on NFS without change. Therefore, migration from on-premises storage to cloud NAS providers requires only minimal effort. By contrast, a full migration from file systems to key-value object stores can be prohibitive because object stores support fewer features (i.e., absence of file attributes, links, locks) and sometimes use weaker consistency models (e.g., eventual consistency [11]) than file systems.

Third, file systems have much richer semantics than key-value object stores, and can significantly simplify application development. As more applications are deployed in clouds, rudimentary object stores have begun to fall short of functionalities to support complex systems [49]. The richer semantics of file systems also provide more optimization opportunities than key-value stores. For example, NFS can be much more efficient with pNFS [54], server-side copying [59], and Application-Data Blocks [59].

Fourth, NFSv4 is optimized for WANs, and its speed over WANs can be considerably improved by caching, as was demonstrated by both academia [35] and industry [52]. The performance boost of an NFS cache, such as SeMiNAS, can be particularly significant with NFSv4 delegations—a client caching mechanism that enables local file operations without communication to remote servers. Delegations do not compromise NFS's strong consistency [54]; they are effective as long as concurrent and conflicting file sharing among clients is rare, which is often true [32].

3. SeMiNAS Design

We present the design of SeMiNAS including its threat model, design goals, architecture, security scheme, NFSv4-based performance optimizations, and caching scheme.

3.1 Threat Model

Our threat model reflects the settings of an organization with offices in multiple locations, and employees in each office store and share files via a SeMiNAS proxy (see Figure 1).

The Storage Provider. We do not trust the storage provider in terms of confidentiality and integrity. Transfering data over WANs or the Internet is vulnerable to network sniffering and man-in-the-middle attacks. Even if the communication is protected by encryption, storing data in plaintext format is still dangerous because the storage device may be shared with other malicious tenants. The same is true for data integrity: attackers inside and outside the provider may covertly tamper with the data. However, we think availability is a smaller concern: all major cloud services had availability higher than four nines (99.99%) [68].

Clients. Clients are trusted. Clients are usually operated by employees of the organization, and are generally trustworthy if proper access control is enforced. SeMiNAS supports NFSv4 and thus can enforce access control using traditional mode bits and advanced ACLs [54].

The Middleware. SeMiNAS is trusted. It provides centralized and consolidated security services. Physically, the middleware is a small cluster of computers and appliances, which can fit in a guarded machine room. Thus, securing the middleware is easier than securing all clients that might scatter over multiple buildings. An organization can also dedicate experienced security personnel to fortify the middleware. SeMiNAS currently does not handle replay attacks, which we are addressing in future work.

3.2 Design Goals

We designed SeMiNAS to achieve the following four goals, ordered by descending importance:

- **High security**: SeMiNAS should ensure high integrity and confidentiality while storing and sharing data among geo-distributed clients.
- **Low overhead**: SeMiNAS should have minimal performance penalty by using a low-overhead security scheme and effectively caching data.
- **Modularity**: SeMiNAS should be modular so that more security features, such as anti-virus and intrusion detection, can be easily added in the future.
- **Simplicity**: SeMiNAS should have a simple architecture that eases development, deployment, and maintenance.

3.3 Architecture

SeMiNAS is a cryptographic file system that serves as a proxy between clients and remote cloud servers. SeMiNAS has a stackable file system architecture so that its security mechanisms can be easily added as layers on top of existing storage gateways and WAN accelerators. Stackable file systems, such as Linux's UnionFS [69] and OverlayFS [7], are flexible for three reasons: (1) they can intercept all file operations including `ioctls`; (2) they can be stacked on top of any other file systems (e.g., `ext4` and NFS); and (3) the stacking can be composed in different orders to achieve a wider range of functionalities. Stackable file systems are also simpler than standalone file systems because they can use existing unmodified file systems as building blocks. Stackable file systems can also achieve high security as shown in previous studies [25, 27, 42, 70].

SeMiNAS consists of multiple proxies in geo-distributed offices that share files securely via a common storage provider. Each office has a SeMiNAS proxy, which acts as an NFS server to clients and as a client to remote NFS servers. SeMiNAS protects files transparently and stores files in ciphertext format in remote cloud servers. A client writes a file by first sending an NFS request to SeMiNAS. Then, SeMiNAS simultaneously encrypts and authenticates the data to generate ciphertext and Message Authentication Codes (MACs). After that, SeMiNAS sends the ciphertext and MACs to the cloud. File reading happens in reverse: SeMiNAS simultaneously verifies and decrypts the data from the ciphertext and MACs when reading from remote servers. Each file has a unique encryption key, which is secretly shared among geo-distributed offices using a PGP-like scheme. SeMiNAS combines the encryption and authentication functionality in one stackable file-system layer.

In addition to the security stackable layer, SeMiNAS contains another stackable file-system layer that caches file content in persistent storage devices in SeMiNAS proxies. Caching is indispensable to avoid the long latency in WANs. SeMiNAS stacks the caching layer on top of the security layer so that file content is cached in cleartext format and reading from the cache does not require decryption. Saving the file's content as cleartext in SeMiNAS is secure because SeMiNAS is fully trusted in our threat model.

3.4 The Security Stackable Layer

In the security stackable file-system layer, SeMiNAS uses authenticated-encryption to simultaneously authenticate and encrypt files [66]. Authenticated-encryption is desirable for strong security because combining a separate encryption layer and an authentication layer is susceptible to security flaws. There are three common ways to combine encryption and authentication: (1) Authenticate then Encrypt (AtE) as used in SSL; (2) Encrypt then Authenticate (EtA) as used in IPSec; and (3) Encrypt and Authenticate (E&A) as used in SSH. Despite being used by popular security protocols

(SSL and SSH), both AtE and E&A turned out to be "not generically secure" [30]. Only one out of the three combinations (i.e., EtA) is considered to be secure [36]. The security ramifications of these combinations are rather complex [6]: even experts have made mistakes [31]. Therefore, SeMiNAS avoids separating encryption and authentication, and instead uses one of the standard authenticated-encryption schemes that perform both operations simultaneously.

Out of the ISO-standardized authenticated-encryption modes, we chose the Galois/Counter Mode (GCM) because of its superior performance [10] over other modes. SeMiNAS strictly follows NIST's guidance of using GCM and meets the "uniqueness requirements on IVs and keys" [15]. GCM accepts three inputs and produces two outputs. The three inputs are the plaintext to be both authenticated and encrypted (PDATA), additional data only to be authenticated (ADATA), and a key; the two outputs are ciphertext and a Message Authentication Code (MAC). Out of the three inputs, either PDATA or ADATA can be absent. This lets SeMiNAS achieve integrity but not encryption by leaving PDATA empty and using the concatenation of data and meta-data as ADATA. This is useful when encryption is not necessary, for example, when storing open-source files.

On write operations, GCM uses the data to be written as PDATA and additional security meta-data (discussed in Section 3.4.2) as ADATA. GCM outputs the ciphertext and MAC, which are then written to the remote server. On read operations, SeMiNAS retrieves the ciphertext and MAC, and then simultaneously verifies the MAC and decrypts the ciphertext. SeMiNAS thus achieves end-to-end data integrity and confidentiality as the protection covers both the transport channel and the storage stack of the server.

3.4.1 Key Management

Key management is critical for strong security. SeMiNAS uses a simple yet robust key management scheme. Each SeMiNAS proxy has a master key pair, which is used for asymmetric encryption (RSA) and consists of a public key (MuK) and a private key (MrK). The public keys are exchanged among geo-distributed proxies manually by security personnel. This is feasible because one geographic office usually has only one SeMiNAS proxy, and key exchange is only needed when opening an office in a new site. This scheme has the advantages of not relying on any third-party for public key distribution. Each file has a symmetric file key (FK) and a 128-bit initialization vector (IV); both FK and IV are 128-bit long and randomly generated. To avoid reusing IVs [15], SeMiNAS adds to the IV the block offset number to generate a unique IV for each block.

Because each SeMiNAS proxy maintains the public keys (MuKs) of all other proxies, the file keys (FKs) can be secretly shared among all SeMiNAS proxies under the protection of MuKs. When creating a file, a SeMiNAS proxy (*creator*) generates an FK. Then for each SeMiNAS proxy with which the creator is sharing the file (*accessor*), the creator encrypts the FK using the accessor's public key (MuK) with the RSA algorithm, and generates a $\langle$SID, EFK$\rangle$ pair where SID is the unique ID of the accessor and EFK is the encrypted FK. All the $\langle$SID, EFK$\rangle$ pairs are then stored in the header of the file. When opening a file, a SeMiNAS proxy, which needs to be an accessor of the file, first finds its $\langle$SID, EFK$\rangle$ pair in the header, and then retrieves the file key FK by decrypting the EFK using its private key (MrK).

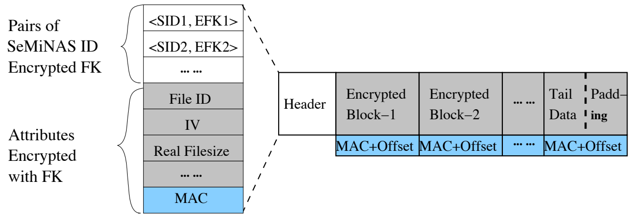


Figure 2. SeMiNAS security meta-data

3.4.2 Security Meta-Data

SeMiNAS maintains per-file meta-data so that files can be encrypted and secretly shared among geo-distributed offices. As shown in Figure 2, the most important per-file meta-data is the encrypted key pairs discussed in Section 3.4.1; other per-file meta-data is authenticated and encrypted file attributes including unique file ID, IV, real file size flags, etc. SeMiNAS saves the per-file meta-data in a 4KB file header. More space can be reserved for the header by punching a hole in the file [59] following the header. Thus, when accommodating more $\langle$SID, EFK$\rangle$ pairs, the header can grow beyond 4KB by filling the hole without shifting any file data.

SeMiNAS divides a file into fix-sized data blocks and applies GCM to each block (with padding if necessary). Therefore, it also maintains per-block meta-data including a 16-byte MAC and an 8-byte block offset (Figure 2). The block offset is combined with the per-file IV to generate the unique per-block IV, and is also used to detect an attack of swapping blocks. SeMiNAS can detect inter-file swapping as well because each file has a unique key. The per-block meta-data is stored using DIX as detailed in Section 3.5.1.

3.5 NFSv4-Based Performance Optimizations

SeMiNAS leverages two advanced NFSv4 features to ensure its security scheme has low overhead: Data-Integrity eXtension (DIX) and compound procedures, discussed next.

3.5.1 NFS Data-Integrity eXtension

DIX gives applications access to the long-existed out-of-band channel of information in storage media. With NFSv4, NFS clients can utilize DIX to store extra information in NFS servers [46]. Figure 3 shows how SeMiNAS leverages DIX and stores the per-block meta-data (a MAC and an offset). This is particularly beneficial in wide-area environments because it saves many extra network round trips for meta-data operations.

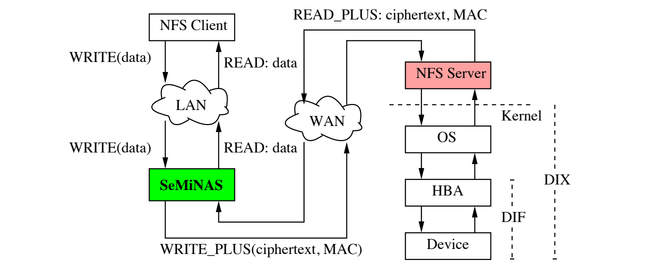


Figure 3. NFS end-to-end data integrity using DIX

Storing MACs and offsets using DIX is better than the other three alternative methods: (1) The first alternative is to write the concatenation of each encrypted block and its MAC as one file in the cloud. This method not only burdens file system meta-data management with many small files [22], but also negates the benefits of using a file-system API such as the file-level close-to-open consistency (which means once clients close a file, all their changes to the file will be available to clients who open the file later). (2) The second alternative method is to use an extra file for all per-block meta-data of each file. However, this is suboptimal, especially considering the high latency of WANs, because writing data blocks incurs an extra write request to the meta-data file. (3) The third alternative is to map one block to a larger block in the cloud-stored file. For example, a file with ten 16KB blocks corresponds to a cloud file with ten slightly larger blocks (i.e., 16KB+N where N is the size of the per-block meta-data). However, this method suffers from extra read-modify-update operations caused by breaking block alignment. Using a larger block size (e.g., 256KB instead of 16KB) alleviates this problem by having fewer extra read-modify-update operations, but it also makes each extra operation more expensive.

Using DIX frees SeMiNAS from all aforementioned problems. To accommodate the 24-byte per-block meta-data, we require a block to be at least 2KB large, because each sector uses at least two DIX bytes by itself for an internal checksum and leaves at most six DIX bytes for applications.

3.5.2 Compound Procedures

To maintain the security meta-data in file headers, SeMiNAS performs many extra file-system operations (e.g., a read of the header when opening a file). Sending separate NFS requests for these extra operations incurs extra WAN round trips and consequently large performance overhead. To avoid this, SeMiNAS leverages *compound procedures*—a new NFSv4 feature that combines multiple operations into one NFS request. Compound procedures can significantly shorten the average latency of operations. This is because, in high-latency networks, a single multi-operation RPC takes almost the same amount of time to process as a single-operation RPC.

Extra operations on file headers are great candidates for compound procedures because all file-header operations immediately follow some other user-initiated file-system operations. By packing extra file-header operations into the same NFS request of the initiating operations, no extra requests are needed. Specifically, SeMiNAS packs extra operations into compound procedures in five scenarios: (1) creating the header after creating a file; (2) reading the header after opening a file; (3) updating the header before closing a dirty file; (4) reading the header when getting file attributes; and (5) getting the attributes (GETATTRS) after writing to a file.

Compound procedures are highly effective for SeMiNAS. We have benchmarked the optimization of compound procedures separately using Filebench's File-Server workload: compound procedures cut the performance overhead of SeMiNAS from 52% down to merely 5%.

3.6 Caching

SeMiNAS has a stackable file-system layer that caches recently used file data blocks, so that hot data can be read in the low-latency on-premises network instead of over WANs. The caching layer is designed to be a write-back cache to minimize writes to the cloud as well as reads. Being write-back, our cache is persistent because some NFS requests—WRITEs with the stable flag, and COMMITs—require dirty data to be flushed to "stable storage" [55] before replying. Because the NFS protocol demands stable writes to survive server crashes, the cache layer also maintains additional meta-data in stable storage to ensures correct crash recovery. The meta-data includes a list of dirty files and a per-block flag to distinguish dirty blocks from clean blocks.

For each cached file, SeMiNAS maintains a sparse file of the same size in the proxy's local file system. Insertion of file blocks are performed by writing to the corresponding blocks of the sparse files. Evictions are done by punching holes at the corresponding locations using Linux's `fallocate` system call. This design delegates file block management to the local file system, and thus significantly simplifies the caching layer. SeMiNAS also stores the crash recovery meta-data of each file in a local file. The caching layer does not explicitly keep hot data blocks in memory, but implicitly does so by relying on the OS's page cache.

When holding a write delegation of a file, SeMiNAS does not have to write cached dirty blocks of the file back to the cloud until the delegation is recalled. Without the write delegation, SeMiNAS has to write dirty data backs upon file close to maintain NFS's close-to-open consistency. To avoid bursty I/O requests and long latency upon delegation recall or file close, SeMiNAS also allows dirty data to be written back periodically at a configurable interval.

4. Implementation

We have implemented a prototype of SeMiNAS in C and C++ on Linux. We have tested our implementation thoroughly using functional unit tests and ensured our prototype passed all `xfstests` [71] cases that are applicable to NFS.

NFS-Ganesha. Our SeMiNAS prototype is based on NFS-Ganesha [12, 21], an open-source user-level NFS server. NFS-Ganesha provides a generic interface to file system implementations with a *File System Abstraction Layer* (FSAL), which is similar to a Virtual File System (VFS) in Linux. With different FSAL implementations, NFS-Ganesha can provide NFS services to clients using different back-ends such as local and distributed file systems. NFS-Ganesha's FSAL implementations include `FSAL_VFS` that uses a local file system as back-end, and `FSAL_PROXY` that uses another NFS server as back-end. We use `FSAL_VFS` for the cloud NFS server, and `FSAL_PROXY` for our SeMiNAS proxy.

Like their stackable counterparts in Linux [73], FSALs can also be stacked to add features in a modular manner. For example, an FSAL for encryption can be stacked on top of `FSAL_PROXY`. NFS-Ganesha originally allowed only one stackable layer; we added the support of multiple stackable layers [47]. NFS-Ganesha configures each exported directory and its backing FSAL separately in a configuration file, allowing SeMiNAS to specify security policies for each exported directory separately.

Authenticated Encryption. We implemented SeMiNAS's authenticated encryption in an FSAL called `FSAL_SECNFS`. We used `cryptopp` as our cryptographic library because it supports a wide range of cryptographic schemes such as AES, GCM, and VMAC [60]. We used AES as the block cipher for GCM. We implemented the NFS DIX in NFS-Ganesha so that ciphertext and the security meta-data can be transmitted together between SeMiNAS proxies and the cloud. First, we implemented the READ_PLUS and WRITE_PLUS operations of NFSv4.2 [59] so that the out-of-band DIX bytes can be transfered together with file block data in one request. Then, at the proxy side, we changed `FSAL_PROXY` to use READ_PLUS and WRITE_PLUS for communications with the cloud NFS server (Figure 3). Lastly, at the cloud side (running `FSAL_VFS`), we changed `FSAL_VFS` to use WRITE_PLUS and READ_PLUS, and to write the ciphertext and security meta-data together to storage devices. Currently, Linux does not have system calls to pass file data and their DIX bytes from user space to kernel; so we used a DIX kernel patchset [51] after we fixed its bugs.

We implemented `FSAL_SECNFS` carefully to avoid any side effects caused by the security meta-data. For example, updating meta-data in a file header has the side effect of changing the file's `ctime` and `mtime`, with an unexpected consequence of invalidating NFS clients' page cache and hurting performance: an NFS client uses `ctime` to check the validity of an NFS file's page cache; an external change of `ctime` implies the file has been modified by another NFS client, and demands the client to invalidate the cache to prevent reading stale data. To avoid this inadvertent cache invalidation, `FSAL_SECNFS` maintains the effective `ctime` and `mtime` in the file header instead of using the real `ctime` and `mtime` attributes of the file.

We also implemented two additional performance optimizations in `FSAL_SECNFS`: (1) We cache the file key (FK) and the $\langle$SID, EFK$\rangle$ pairs in memory to reduce the frequency of expensive RSA decryptions of FKs. This is secure because `FSAL_SECNFS` runs in the trusted SeMiNAS proxies. (2) We use the faster VMAC [10, 60] (3.2$\times$ faster on our testbed) instead of GCM when only integrity (but not encryption) is required.

Caching. The persistent caching file-system layer of SeMiNAS is implemented as another FSAL named `FSAL_PCACHE`. Because `FSAL_PCACHE` needs to write back dirty data using separate threads, we implemented `FSAL_PCACHE` on top of a home-built external caching library to avoid complicating NFS-Ganesha's threading model. The caching library provides caching (lookup, insert, invalidate, etc.) and write-back APIs for `FSAL_PCACHE`. When inserting dirty blocks of a file into the cache using this library, `FSAL_PCACHE` registers a write-back callback function along with the dirty buffer to the library. The callbacks are invoked periodically as long as the file remains dirty. When closing a file, `FSAL_PCACHE` calls the write-back function directly, and deregisters the callback to the library.

Lines of Code. The implementation of our SeMiNAS prototype took about 25 man-months (of serveral graudate students over 3 years), and added around 13,000 lines of C/C++ code. In addition, we have fixed bugs and added the multi-layer stacking feature in NFS-Ganesha; our patches have been merged into the mainstream NFS-Ganesha. We have also fixed bugs in the DIX kernel patchset. We plan to release all code as open source in the near future.

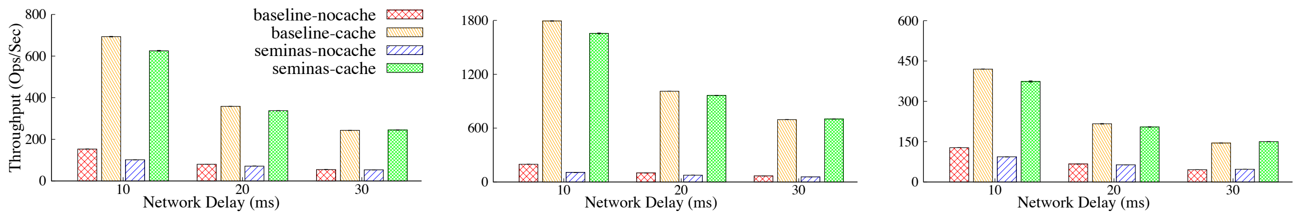
5. Evaluation

We now present the evaluation of SeMiNAS under different configurations, workloads, and network settings.

5.1 Experimental Setup

Our testbed consisted of seven Dell R710 machines running CentOS 7.0 with a 3.14 Linux kernel. Each machine has a six-core Intel Xeon X5650 CPU, a Broadcom 1GbE NIC, and an Intel 10GbE NIC. Five machines run as NFS clients and each of them has 1GB RAM. Both remaining machines have 64GB: one of them runs as a SeMiNAS proxy and the other emulates a cloud NFS server. Clients communicated to the proxy using the 10GbE NIC, whereas the proxy communicated to the server using the 1GbE NIC (to simulate a slower WAN). The average RTT between the clients and the SeMiNAS proxy is 0.2ms. The SeMiNAS proxy uses an Intel DC S3700 200GB SSD for the persistent cache. We emptied the cache before each experiment to observe the system's behavior when an initial empty cache is gradually filled. We used 4KB as the block size of SeMiNAS.

To better emulate the network between the SeMiNAS proxy and the cloud, we injected 10–30ms delays in the



(a) 1:1 Read-Write Ratio (b) 5:1 Read-Write Ratio (c) 1:5 Read-Write Ratio

Figure 4. Aggregate throughput of baseline and SeMiNAS with 4KB I/O size, 5 NFS clients, and one thread per client under different read-write ratios and network delays.

outbound link of the server using `netem`; 10ms and 30ms are the average network latencies we measured from our machines to in-state data centers and the closest Amazon data center, respectively. We patched the server's kernel with the DIX support [51] (with our bug fixes) that allows DIX bytes to be passed from user space to kernel.

Physical storage devices that support DIX are still rare [**?**], so we had to set up a 20GB DIX-capable virtual SCSI block device backed by RAM using `targetcli` [45]. Using RAM, instead of a disk- or flash-backed loop device, allowed us to emulate the large storage bandwidth provided by distributed storage systems in the cloud. Although using RAM fails to account for the server-side storage latency, the effect is minor because the Internet latency (typically 10–100ms) usually dwarfs the storage latency (typically 1–10ms), especially considering the popularity of cloud in-memory caching systems such as RAMcloud [48] and Memcached [18]. If storage latency in the cloud was counted, the extra latency added by SeMiNAS's security mechanisms would actually be smaller relative to the overall latency; hence the results we report here are more conservative. The DIX-capable device was formatted with `ext4`, and exported by NFS-Ganesha using `FSAL_VFS`.

We verified that all SeMiNAS's security features work correctly. To test integrity, we created files on a client, changed different parts of the files on the cloud server, and verified that SeMiNAS detected all the changes. To test encryption, we manually confirmed that file data was an unreadable ciphertext when reading directly from the server, but its plaintext was identical to what was written by clients.

We used the vanilla `FSAL_PROXY` as baseline. `FSAL_PROXY` uses up to 64 concurrent requests each with a 2MB-large RPC buffer. We benchmarked two cases—with and without the persistent cache (`FSAL_PCACHE`) for both the baseline and SeMiNAS. We benchmarked a set of simple synthetic micro-workloads, and Filebench [17] macro-workloads including the NFS Server, Web Proxy, and Mail Server.

5.2 Micro-Workloads

We benchmarked SeMiNAS using three micro-workloads: (1) random file accesses with different read-write ratios, (2) file creation, and (3) file deletion.

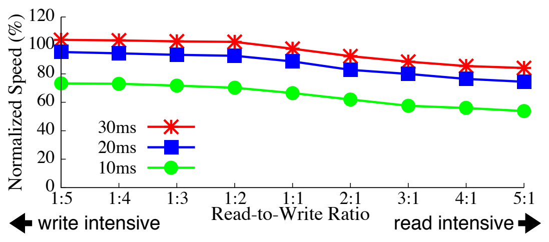
5.2.1 Read-Write Ratio Workload

Read-write ratio is an important workload characteristic that influences the performance impact of SeMiNAS's security mechanisms and the persistent cache (`FSAL_PCACHE`). We studied read-write ratios from write-intensive (1:5) to read-intensive (5:1) to cover common ratios in real workloads [32, 53]. We pre-allocated 100 1MB-large files for each of the five NFS clients, and then repeated the following operations for two minutes: randomly pick one file, open it with `O_SYNC`, perform n 4KB reads and m 4KB writes at random offsets, and close it. We varied n and m to control the read-write ratio. We also ensured $n + m$ is a constant (i.e., 60) so that dirty contents are written back in the same frequency.

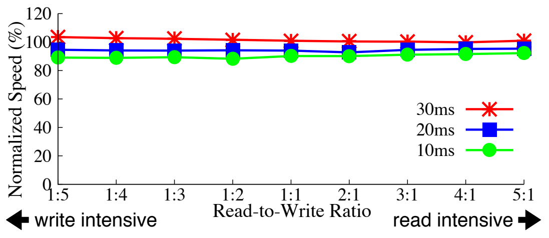
Figure 4 shows the results when the read-write ratios are 1:1, 5:1, and 1:5. Overall, the configurations with caching outperform their "nocache" counterparts. For the 1:1 read-write ratio, caching speeds the workloads up by 4–6×. The degree of speed-up grows to 9–16× as the workload becomes read-intensive (Figure 4(b)), but drops to 3–4× as the workloads become write-intensive (Figure 4(c)). The cache's help to writes is smaller than to reads because SeMiNAS has to write dirty writes back to the cloud server upon file close, so that clients in other offices can observe the latest changes.

To better illustrate the performance impact of SeMiNAS, we show SeMiNAS's relative throughput to the baseline in Figure 5. When it is write-intensive, SeMiNAS can be up to 3% faster than the baseline regardless of the presence of `FSAL_PCACHE`. This is because the baseline uses extra COMMITs following WRITEs to make write operations stable, whereas SeMiNAS does so by simply setting the stable flag of WRITE_PLUS requests. The normalized throughput of SeMiNAS drops as the workload becomes more read-intensive (Figure 5(a)) for two reasons: (1) the effect of saving COMMITs becomes smaller as the number of writes goes down; and (2) SeMiNAS has to authenticate and decrypt (encrypt) data when reading from (writing to) the cloud server.

When cache is on (Figure 5(b)), the normalized throughput decreases much slower and is almost flat. This is because (1) the cache content is in plaintext format and reading from cache needs no more authentication or decryption; and (2) writes are acknowledged once dirty data is inserted into the cache and the real write-back happens asynchronously.



(a) Persistent Cache (`FSAL_PCACHE`) Off



(b) Persistent Cache (`FSAL_PCACHE`) On

Figure 5. Relative throughput of SeMiNAS to the baseline under 10ms, 20ms, and 30ms network delays.

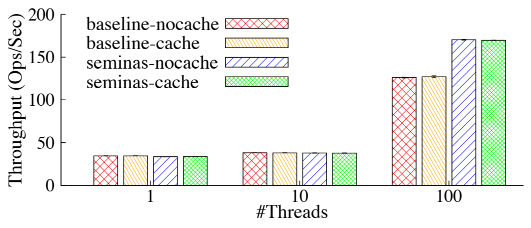


Figure 6. Throughput of creating empty files in a 10ms-delay network with one NFS client.

Note that the normalized throughput of SeMiNAS is better for longer network delay no matter if the cache is on or off. This is because SeMiNAS is optimized for wide-area environments and minimizes the number of round trips between the proxy and the cloud.

5.2.2 File-Creation Workload

Depending on the number of threads, SeMiNAS has different performance impact over the baseline for file creation. As shown in Figure 6, SeMiNAS has only negligible performance impact when there are only one or ten threads. Surprisingly, SeMiNAS makes file creation 35% faster than the baseline when the number of threads grows to 100. This is caused by the TCP connection between the proxy and the server, particularly due to the TCP Nagle algorithm [65]. The algorithm adds extra delay to outbound packets in the hope of coalescing multiple small packets into fewer, larger ones; TCP Nagle trades off latency for bandwidth. This trade-off hurts the baseline performance of this file-creation workload, which is meta-data intensive and generates many small network packets. In contrast, the algorithm favors SeMiNAS because SeMiNAS uses compound procedures to pack file creations and extra secure operations (e.g., creating file headers) together to form larger packets.

The number of threads influences the performance because all threads share one common TCP connection between the proxy and the server. More threads bring more coalescing opportunities; otherwise, the extra waiting of TCP Nagle is useless if the current request is blocked and no other requests are coming. To verify this explanation, we temporarily disabled TCP Nagle by setting the `TCP_NODELAY` socket option, and observed that SeMiNAS's throughput became about the same (99%) as the baseline thereafter.

Figure 6 also shows that, as expected, the persistent cache (`FSAL_PCACHE`) does not make a difference in file creation because `FSAL_PCACHE` caches only data, but not meta-data.

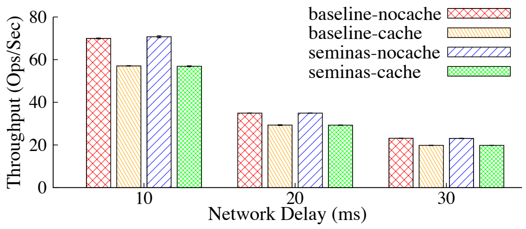


Figure 7. Throughput of deletion of 256KB files, with one NFS client and 100 threads.

5.2.3 File-Deletion Workload

Figure 7 shows the results of deleting files, where SeMiNAS have the same throughput as the baseline with and without the persistent cache. This is because SeMiNAS does not incur any extra operations upon file deletion. However, adding `FSAL_PCACHE` makes file deletion 12–18% slower. This is because `FSAL_PCACHE` needs one extra lookup operation to delete a file. `FSAL_PCACHE` uses file handles as unique keys of cached content, but the file deletion function (i.e., `unlink`) uses the parent directory and file name, rather than the file handle, to specify the file. Those extra lookups could be saved if `FSAL_PCACHE` maintains a copy of the file-system namespace, which we left as future work.

5.3 Macro-Workloads

We evaluated SeMiNAS using three Filebench macro-workloads: (1) NFS Server, (2) Web Proxy, and (3) Mail Server.

5.3.1 Network File-System Server Workload

Filebench's NFS-Server workload emulates the I/O activities experienced by an NFS server. We used the default settings of the workload, which contains 10,000 1KB-to-1700KB-large files totalling 2.5GB. The read sizes of the workload range from 8K to 135K with 85% reads 8KB-large; the write sizes range from 9K to 135K with 50% writes 9KB- to 15KB-large. The workloads perform a variety of operations including open, read, write, append, close, create, and delete.

Figure 8 shows the results of running this workload. Without cache, the baseline proxy's throughput decreases

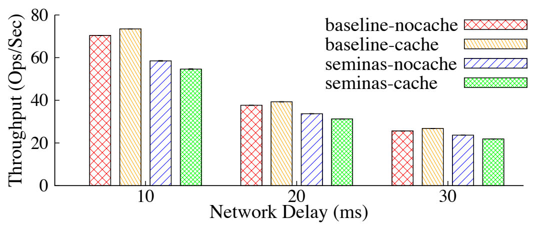


Figure 8. Throughput of Filebench NFS-Server workload, with one benchmarking client and one thread.

from 70 ops/sec to 25 ops/sec as the network latency between the proxy and the server increased from 10ms to 30ms. After adding the persistent data cache, the baseline throughput increases but only slightly. The performance boost of caching is small because the workload contains many meta-data operations that cannot be cached; for example, open and close operations have to talk to the server in order to maintain close-to-open consistency.

In this NFS-Server workload, SeMiNAS is 8–17% and 18–26% slower than the baseline without and with cache, respectively. As the network delay grows, the performance penalty of SeMiNAS becomes smaller regardless of the presence of cache. This is because we optimized SeMiNAS for wide-area environment by minimizing the number of round trips between the proxy and the cloud server.

We noticed that adding cache to SeMiNAS actually makes the performance slightly worse (the last two bins in each group of Figure 8). This is because `FSAL_PCACHE` makes file deletions slower with extra lookups (see Section 5.2.3), and file deletions count for as much as 8% of all WAN round trips in this workload. The extra lookups incurred by file deletions are also one of the reasons why the cache's performance boost to the baseline is small, although a lookup in the baseline is cheaper than in SeMiNAS (because SeMiNAS needs extra bookkeeping during lookups).

5.3.2 Web-Proxy Workload

Filebench's Web-Proxy workload emulates the I/O activities of a simple Web-Proxy server, which fits well with SeMiNAS's proxy architecture. The workload has a mix of file creation, deletion, many open-read-close operations, and a file append operation to emulate logging. The default Web-Proxy workload has 10,000 files with an average size of 16KB in a flat directory, and 100 benchmarking threads. We made three changes to the default settings: (1) we placed the files in a file-system directory tree with a mean directory width of 20 because a flat directory made the baseline so slow (around 20 ops/sec) that SeMiNAS did not show any performance impact at all; (2) we enlarges the average file size to 256KB so that the working set size (2.56GB) is more than twice the size of the NFS client's RAM (1G) but smaller than the size of the persistent cache; and (3) we used a Gamma distribution [64, 67] to control the access pattern

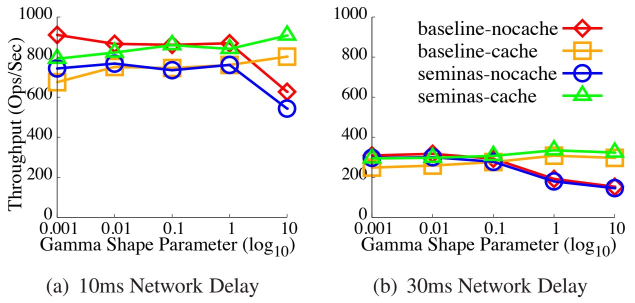


Figure 9. Web-proxy results with different access patterns, one NFS client, and 100 threads. A larger value of the shape parameter means less locality in the access pattern.

of the files, but varied the Gamma's shape parameter (k) to emulate access patterns with different degrees of locality.

Figure 9 shows the Web-Proxy workload results. With 10ms network delay, the throughput of "baseline-nocache" drops from 910 to 630 ops/sec as the degree of workload locality decreases. The "seminas-nocache" curve in Figure 9(a) has a similar shape to its baseline counterpart, but at 11–18% lower throughputs as a result of extra security mechanisms in SeMiNAS. With high locality ($k <= 1$), adding `FSAL_PCACHE` (blue circle curve) actually slows down the baseline (red diamond curve) because (1) `FSAL_PCACHE` is not useful when most reads are served from the client's page cache; and (2) `FSAL_PCACHE` also introduces extra overhead for file deletions. Conversely, as the locality drops ($k = 10$), the client's page cache becomes less effective and the persistent cache, which is larger than the working set size, becomes effective.

Figure 9(a) shows that SeMiNAS actually makes the workload up to 15% faster than the baseline when there is a cache (i.e., the green triangle curve is higher than the orange rectangle curve). This is because SeMiNAS makes file creations faster in this highly-threaded workload thanks to the TCP Nagle algorithm (see Section 5.2.2).

For a slower network of 30ms latency (Figure 9(b)), the throughputs of baseline and SeMiNAS are both slower than in the faster network (10ms). However, the relative order of the four configurations remains the same. Without the cache, SeMiNAS has a small performance penality of 4–6%; with the cache, SeMiNAS sees a performance *boost* of 9–19%.

5.3.3 Mail-Server Workload

Filebench's Mail-Server workload emulates the I/O activity of an `mbox`-style e-mail server that stores each e-mail in a separate file. The workload consists of 16 threads, each performing create-append-sync, read-append-sync, read, and delete operations on a fileset of 10,000 16KB files.

We used this Mail-Server workload to test the scalability of SeMiNAS by gradually increasing the number of NFS clients. As shown in Figure 10, both the baseline and SeMiNAS scales well as the number of clients grows. The relative

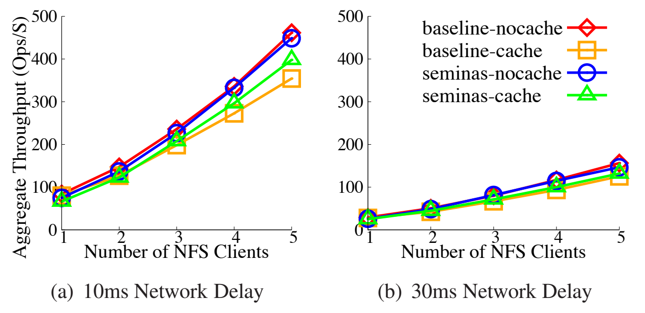


Figure 10. Filebench Mail Server throughput results.

order and trend of the four curves in Figure 10 share similarity with the curves of the Web-Proxy workload results for similar reasons. In terms of relative speed to the baseline, SeMiNAS is 1–11% slower without cache, and is between 17% slower (fewer clients) and 12% faster (more clients) with cache depending on the network delay and effectiveness of TCP Nagle algorithm (see Section 5.2.2).

6. Related Work

SeMiNAS is related to (1) secure distributed storage systems, (2) cloud NAS, and (3) cloud storage gateways.

Secure distributed storage systems. SFS [37], Cepheus [29], SiRiUS [24], and SUNDR [33] are cryptographic file systems that provide end-to-end file integrity and confidentiality with minimal trust on the server; but they all used remote servers as block stores instead of file-system servers, and none of them took advantage of NFSv4, which was not invented at the time. SeMiNAS' sharing of file keys (FK) is similar to SiRiUS [24]. However, because access control is enforced by the trusted middleware, SeMiNAS needs only one key per file instead of two (one for reading and the other for writing) in SiRiUS. NASD [23] and SNAD [41] add strong security to distributed storage systems using secure distributed disks. In both NASD's and SNAD's threat models, disks are trusted; these are fundamentally different from threat models in the cloud where storage hosts are physically inaccessible by clients and thus hard to be trusted.

Cloud NAS. Panache [35] is a parallel file-system cache that enables efficient global file access over WANs but without WAN's fluctuations and latencies. It uses pNFS to read data from remote cloud servers and caches them locally in a cache cluster. Using NFS, Panache enjoys the strong consistency of file system API. However, its main focus is high performance with parallel caching, instead of security. Cloud NAS services are provided by companies such as Amazon [26], SoftNAS [57] and Zadara Storage [72]. These services focus on providing file system services in public clouds. These cloud NAS service providers control and trust the ultimate storage devices, whereas SeMiNAS cannot control or trust the devices. FileWall [56] combines the idea of network firewalls with network file systems, and provides file access control based on both network context (e.g., IP address) and file system context (e.g., file owner). FileWall can protect cloud NAS servers from malicious clients, whereas SeMiNAS is for protecting clients from clouds.

Cloud storage gateways. Using the cloud as back-end, a cloud gateway gives a SAN or NAS interface to local clients, and can provide security and caching features. There are several cloud gateway technologies, in both industry and academia. In academia, Hybris [14], BlueSky [62], and Iris [58] are examples of cloud storage gateway systems that provide integrity. Hybris additionally gives fault tolerance by using multiple cloud providers, whereas BlueSky also provides encryption. BlueSky and Iris have a file system interface on the client side, and Hybris provides a key-value store. However, none of them uses a file system API for cloud communication, and thus they offer only a weaker model—the eventual consistency model that usually uses a RESTful API. In the storage industry, NetApp SteelStore [44] is a cloud integrated storage for backup. Riverbed SteelFusion [52] provides a hyper-converged infrastructure with WAN optimization, data consolidation, and cloud back-ends. The exact security mechanisms of SteelStore and SteelFusion are not publicly known although they claim to support encryption.

7. Conclusions

We presented the design, implementation, and evaluation of SeMiNAS, a secure middleware using cloud NAS as back-end. SeMiNAS provides end-to-end data integrity and confidentiality while allowing files to be securely shared among geo-distributed offices. SeMiNAS uses authenticated encryption to safely and efficiently encrypt data and generate MACs at the same time. SeMiNAS is optimized for WANs and has a persistent cache to hide high WAN latencies. SeMiNAS leverages advanced NFSv4 features, including Data Integrity eXtention (DIX) and compound procedures, to manage its secure meta-data without incurring extra network round trips. Our benchmarking with Filebench workloads showed that SeMiNAS has a performance penalty less than 26%, and occasionally improve performance by up to 19% thanks to its effective use of compound procedures.

7.1 Limitations and Future Work

SeMiNAS does not handle attacks based on side-channel information or file-access patterns. SeMiNAS is vulnerable to replay attacks, which usually requires building a Merkle tree [40] for the entire file system and is thus expensive in WANs. We are developing an efficient scheme to thwart replay attacks. We are also adding file-name encryption and anti-virus scanning of file content.

Acknowledgments

We thank the anonymous SYSTOR reviewers for their useful feedback. This work was made possible in part thanks to NSF awards CNS-1251137 and CNS-1302246.

References

[1] A. Bessani and R. Mendes and T. Oliveira and N. Neves and M. Correia and M. Pasin and P. Verissimo. SCFS: A Shared Cloud-backed File System. In *USENIX ATC 14*, pages 169–180. USENIX, 2014.

[2] CBS SF Bay Area. Nude celebrity photos flood 4chan after apple icloud hacked, 2014. *http://goo.gl/p5a49Y*.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, 2010.

[4] L. N. Bairavasundaram, G. R. Goodson, B. Schroeder, A. C. Arpaci-dusseau, and R. H. Arpaci-dusseau. An Analysis of Data Corruption in the Storage Stack. In *Proceedings of the Sixth USENIX Conference on File and Storage Technologies (FAST '08)*, San Jose, CA, February 2008. USENIX Association.

[5] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology – ASIACRYPT 2000*, pages 531–545. Springer, 2000.

[6] Neil Brown. Overlay filesystem, 2015.

[7] M. Chen, D. Hildebrand, G. Kuenning, S. Shankaranarayana, B. Singh, and E. Zadok. Newer Is Sometimes Better: An Evaluation of NFSv4.1. In *Proceedings of the 2015 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2015)*, Portland, OR, June 2015. ACM.

[8] Yao Chen and Radu Sion. To cloud or not to cloud?: musings on costs and viability. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 29. ACM, 2011.

[9] Wei Dai. Crypto++ 5.6.0 Benchmarks. *http://www.cryptopp.com/benchmarks.html*, March 2009.

[10] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, 2007.

[11] Philippe Deniel, Thomas Leibovici, and Jacques-Charles Lafoucrière. GANESHA, a multi-usage with large cache NFSv4 server. In *Linux Symposium*, page 113, 2007.

[12] I/O Controller Data Integrity Extensions, 2016. *https://oss.oracle.com/~mkp/docs/dix.pdf*.

[13] Dan Dobre, Paolo Viotti, and Marko Vukolić. Hybris: Robust hybrid cloud storage. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–14. ACM, 2014.

[14] M. Dworkin. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. National Institute of Standards and Technology (NIST), November 2007.

[15] M. Eisler, A. Chiu, and L. Ling. RPCSEC_GSS protocol specification. Technical Report RFC 2203, Network Working Group, September 1997.

[16] Filebench, 2016. *http://filebench.sf.net*.

[17] B. Fitzpatrick. Memcached. *http://memcached.org*, January 2010.

[18] K. Fu, M. F. Kaashoek, and D. Mazières. Fast and secure distributed read-only file system. In *Proceedings of the 4th Usenix Symposium on Operating System Design and Implementation (OSDI '00)*, pages 181–196, San Diego, CA, October 2000. USENIX Association.

[19] K. Fu, M. F. Kaashoek, and D. Mazi'eres. Fast and secure distributed read-only file system. *Computer Systems*, 20(1):1–24, 2002.

[20] NFS-GANESHA, 2016. *http://sourceforge.net/apps/trac/nfs-ganesha/*.

[21] G. R. Ganger and M. F. Kaashoek. Embedded inodes and explicit grouping: exploiting disk bandwidth for small files. In *Proceedings of the Annual USENIX Technical Conference*, Anaheim, CA, January 1997. USENIX Association.

[22] G. A. Gibson, D. F. Nagle, W. Courtright II, N. Lanza, P. Mazaitis, M. Unangst, and J. Zelenka. NASD Scalable Storage Systems. In *Proceedings of the 1999 USENIX Extreme Linux Workshop*, Monterey, CA, June 1999.

[23] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh. Sirius: Securing remote untrusted storage. In *Proceedings of the Tenth Network and Distributed System Security (NDSS) Symposium*, pages 131–145, San Diego, CA, February 2003. Internet Society (ISOC).

[24] M. Halcrow. eCryptfs: a stacked cryptographic filesystem. *Linux Journal*, 2007(156):54–58, April 2007.

[25] Red Hat. What is DIF/DIX (also known as PI)? *https://access.redhat.com/solutions/41548*, December 2015.

[26] Amazon Inc. Amazon elastic file system. *https://aws.amazon.com/efs/*, September 2015.

[27] N. Joukov and J. Sipek. GreenFS: Making Enterprise Computers Greener by Protecting Them Better. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008 (EuroSys 2008)*, Glasgow, Scotland, April 2008. ACM.

[28] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable secure file sharing on untrusted storage. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, pages 29–42, San Francisco, CA, March 2003. USENIX Association.

[29] Kevin Fu. Group sharing and random access in cryptographic storage file systems. Master's thesis, MIT, 1999.

[30] H. Krawczyk. The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?). In *Proceedings of CRYPTO'01*. Springer-Verlag, 2001.

[31] H. Krawczyk. Encrypt-then-MAC for TLS and DTLS. *http://www.ietf.org/mail-archive/web/tls/current/msg12766.html*, June 2014.

[32] A. W. Leung, S. Pasupathy, G. Goodson, and E. L. Miller. Measurement and analysis of large-scale network file system workloads. In *Proceedings of the Annual USENIX Technical Conference*, pages 213–226, Boston, MA, June 2008. USENIX Association.

[33] J. Li, M. Krohn, D. Mazières, and D. Shasha. Secure untrusted data repository (SUNDR). In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*

(OSDI), pages 121–136, San Francisco, CA, December 2004. ACM SIGOPS.

[34] L. Lu, A. C. Arpaci-dusseau, R. H. Arpaci-dusseau, and S. Lu. A Study of Linux File System Evolution. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, San Jose, CA, February 2013. USENIX Association.

[35] M. Eshel and R. Haskin and D. Hildebrand and M. Naik and F. Schmuck and R. Tewari. Panache: A Parallel File System Cache for Global File Access. In *FAST*, pages 155–168. USENIX, 2010.

[36] U. Maurer and B. Tackmann. On the soundness of authenticate-then-encrypt: Formalizing the malleability of symmetric encryption. In *Proceedings of CCS'10*. ACM, 2010.

[37] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 124–139, Charleston, SC, December 1999. ACM.

[38] Alex McDonald. The background to NFSv4.1. *;login: The USENIX Magazine*, 37(1):28–35, February 2012.

[39] Peter Mell and Tim Grance. The NIST definition of cloud computing. Technical report, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, 2011.

[40] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO'87, pages 369–378, London, UK, 1988. Springer-Verlag.

[41] E. Miller, W. Freeman, D. Long, and B. Reed. Strong security for network-attached storage. In *Proceedings of the First USENIX Conference on File and Storage Technologies (FAST '02)*, pages 1–13, Monterey, CA, January 2002. USENIX Association.

[42] Y. Miretskiy, A. Das, C. P. Wright, and E. Zadok. Avfs: An On-Access Anti-Virus File System. In *Proceedings of the 13th USENIX Security Symposium (Security 2004)*, pages 73–88, San Diego, CA, August 2004. USENIX Association.

[43] CNN Money. Hospital network hacked, 2014. *http://goo.gl/wSfzAx*.

[44] NetApp. NetApp SteelStore Cloud Integrated Storage Appliance. *http://www.netapp.com/us/products/protection-software/steelstore/*, 2014.

[45] Linux-IO Target, 2015. *https://lwn.net/Articles/592093/*.

[46] End-to-end Data Integrity For NFSv4, 2014. *http://tools.ietf.org/html/draft-cel-nfsv4-end2end-data-protection-01*.

[47] Arun Olappamanna Vasudevan. Support stacking multiple FSALs, 2014. *http://sourceforge.net/p/nfs-ganesha/mailman/message/32999686/*.

[48] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. Rumble, E. Stratmann, and R. Stutsman. The Case for RAMClouds: Scalable High-performance Storage Entirely in DRAM. *SIGOPS Oper. Syst. Rev.*, 43(4):92–105, 2010.

[49] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. big web services: Making the right architectural decision. In *Proceedings of the 17th International conference on World Wide Web*, pages 805–814, New York, NY, April 2008. ACM.

[50] Data integrity user-space interfaces, 2014. *https://lwn.net/Articles/592093/*.

[51] Riverbed. SteelFusion. *http://www.riverbed.com/products/branch-office-data/*, 2012.

[52] D. Roselli, J. R. Lorch, and T. E. Anderson. A comparison of file system workloads. In *Proceedings of the Annual USENIX Technical Conference*, pages 41–54, San Diego, CA, June 2000. USENIX Association.

[53] S. Shepler and M. Eisler and D. Noveck. NFS Version 4 Minor Version 1 Protocol. Technical Report RFC 5661, Network Working Group, January 2010.

[54] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. NFS Version 4 Protocol. Technical Report RFC 3530, Network Working Group, April 2003.

[55] S. Smaldone, A. Bohra, and L. Iftode. FileWall: A Firewall for Network File Systems. In *Dependable, Autonomic and Secure Computing, Third IEEE International Symposium on*, pages 153–162, 2007.

[56] SoftNAS. SoftNAS Cloud. *https://www.softnas.com/wp/*, 2016.

[57] Emil Stefanov, Marten van Dijk, Ari Juels, and Alina Oprea. Iris: A scalable cloud file system with efficient integrity checks. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 229–238. ACM, 2012.

[58] T. Haynes. NFS Version 4 Minor Version 2 Protocol. Technical report, Network Working Group, September 2015. *https://tools.ietf.org/html/draft-ietf-nfsv4-minorversion2-39*.

[59] Ted Krovetz and Wei Dai. VMAC: Message Authentication Code using Universal Hashing. Technical report, CFRG Working Group, April 2007. *http://www.fastcrypto.org/vmac/draft-krovetz-vmac-01.txt*.

[60] vinf.net. Silent data corruption in the cloud and building in data integrity, 2011. *http://goo.gl/IbMyu7*.

[61] Michael Vrable, Stefan Savage, and Geoffrey M Voelker. Bluesky: a cloud-backed file system for the enterprise. In *FAST*, page 19, 2012.

[62] Wikipedia. Gamma Distribution. *http://en.wikipedia.org/wiki/Gamma_distribution*.

[63] Wikipedia. Nagle's algorithm. *http://en.wikipedia.org/wiki/Nagle's_algorithm*.

[64] Wikipedia. Authenticated Encryption, 2015. *https://en.wikipedia.org/wiki/Authenticated_encryption*.

[65] A. W. Wilson. Operation and implementation of random variables in Filebench.

[66] Network World. Which cloud providers had the best uptime last year?, 2014. *http://goo.gl/SZOKUT*.

[67] C. P. Wright, J. Dave, P. Gupta, H. Krishnan, D. P. Quigley, E. Zadok, and M. N. Zubair. Versatility and unix semantics in namespace unification. *ACM Transactions on Storage (TOS)*, 2(1):1–32, February 2006.

[68] C. P. Wright, M. Martino, and E. Zadok. NCryptfs: A secure and convenient cryptographic file system. In *Proceedings of the Annual USENIX Technical Conference*, pages 197–210, San Antonio, TX, June 2003. USENIX Association.

[69] SGI XFS. xfstests, 2016. *http://xfs.org/index.php/Getting_the_latest_source_code*.

[70] Zadara Storage. Virtual Private Storage Array. *https://www.zadarastorage.com/*, 2016.

[71] E. Zadok, I. Bădulescu, and A. Shender. Extending file systems using stackable templates. In *Proceedings of the Annual USENIX Technical Conference*, pages 57–70, Monterey, CA, June 1999. USENIX Association.