Design for a Decentralized Security System for Network Attached Storage

William Freeman[1,2] and Ethan Miller[1]

[1]Dept. of Computer Science and Electrical Engineering, University of Maryland Baltimore County, Baltimore MD
[2]Laboratory for Telecommunication Sciences, Adelphi MD
wef@lts.ncsc.mil, elm@csee.umbc.edu

ABSTRACT

This paper describes an architecture for a secure file system based on network-attached storage that guarantees end-to-end encryption for all user data. We describe the design of this system, focusing on the features that allow it to ensure that data is written and read only by authorized users, even in the face of attacks such as network snooping and physically capturing the storage media.

Our work shows that such a system is feasible given the speeds of today's microprocessors, and we discuss benchmark results using several popular encryption and authentication algorithms that could be used on storage devices in such a system. Based on these calculations, we present the overall performance of the system, showing that it is nearly as fast as the non-encrypted file systems in wide use today.

1. Introduction

While computers have provided a great service in the office automation arena, they have led to billions of dollars in lost revenue due to attacks by both hackers and insiders. Most offices and universities rely heavily on their distributed computer environment, which for the purposes of this study, consists of workstations and a shared file system. This file system is typically stored on a centralized file server that is managed by a system administrator with super-user privileges. The need to back up the file system requires that the super-user has the ability to read the entire file system. When the end users want to read a file, the file is sent across the network without any protection against rogue users simply reading the data as it travels. A more sophisticated hacker could also modify or prevent the modification of data.

In our first paper published on this topic [5], the technical feasibility of placing cryptographic controls in a performance-critical system was established. This paper addresses the security and performance concerns of today's distributed file systems. The performance problems are caused by having few (possibly one) network connections, and shared hardware resources (namely the backplane) in the file server. Each disk drive in a file server can be coupled with a low-cost board computer to make an intelligent disk. These disks can be distributed across the network which in today's switched LANs, has far greater network bandwidth than a single server. The physical separation of the drives also increases reliability, since one drive having a catastrophic failure such as catching on fire will not damage the other drives.

The security of today's distributed file systems such as NFS and AFS merely provide a weak scheme for access control, but neglect data integrity and confidentiality. This is especially true when considering that the system administrator with super-user privileges can read and write any user data in the system. Our proposed system requires that all user data be encrypted and signed at the user's workstation. This mechanism ensures that any files on the file server are protected from reading or from undetected modification by anyone, even someone with unrestricted physical access to the drives. The drives and the tape backups of the drives will thus contain no sensitive data in non-encrypted form.

2. Motivation

Many systems have been designed to try to solve the performance problems or security problems of modern distributed file systems. A few have even attempted to solve both problems. These systems are discussed below with respect to their shortcomings.

2.1 Security Issues

The primary security capability of current distributed file systems is access control. This is comprised of preventing unauthorized release of information, unauthorized modification of information, or unauthorized denial of resource usage. This is usually provided by something as simple as a user ID passed in the clear or a security token, which often can be thwarted using elementary attacks. Many system designers choose to ignore the fact that MAC addresses (unique per-interface identifiers), IP addresses, TCP sequence numbers, user names, user passwords, and host names are sent in the clear across unsecure networks. NFS, for example, uses file handles to identify files that are being accessed, with two parts being public and one being secret. It is well-known that the "secret" number can often be easily guessed or calculated. Amazingly, the main security function in NFS was not designed for security use, but for preventing two clients from simultaneously accessing different versions of the file [9].

To further complicate the security problem, the standard protocols for Internet communications are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) over the Internet protocol (IP). Many systems rely on the "security" provided by this protocol. An example of this is authenticating the beginning of a session and then just trusting that further communications on that TCP session are legitimate. This is a poor method to provide security because there are many well known attacks that can "steal" an open connection such as IP address spoofing [8].

There are many methods being developed for protecting traffic while traversing a network. For example, IPsec can encrypt the entire datagram in transit between two firewalls, as well as encrypt only the data field for host-to-host communications. Hosts can also set up a Secure Sockets Layer (SSL) which is a transport layer encryption scheme.

There are also many methods in use and in development to provide strong authentication of the user for access control. Instead of merely sending the user's password in the clear, one-time-use data could be sent. For example, S/Key uses the one-way property of certain hashing algorithms to prevent a password replay attack [7]. Kerberos can be implemented in a fairly secure manner but does not scale well with large systems and does not provide

for protection of data in transit. Kerberos is a critical component for many systems recently developed, including work securing network attached disks [3].

Although many of these schemes are great ideas and will protect the data as it traverses the network, most do nothing to protect the data as it sits on the disk or backup tapes. The super-user can still read any user's data, and the disks and backup tapes still contain sensitive data, thus requiring physical security. Work has been done to provide a secure local file system, and perhaps even secure a particular user's NFS files, but the idea of providing a secure network attached storage system with decentralized security has not been investigated.

Frangipani, developed at DEC, was one attempt to make a secure distributed file system. It was built over the Petal distributed virtual disk system. Frangipani used a client/server configuration, but unlike other distributed file systems, any Frangipani machine can read or write any block of the Petal virtual disk. This imposed the requirement that any Frangipani computer runs a trusted operating system [11].

Some system designers have tried to fix the single point of failure problem with a centralized security scheme by having some small number of computers at the heart of the security mechanism. While this makes the system more reliable, there is still a privileged user that can compromise the entire system. Decentralized security means that there is no central computer(s) responsible for providing data security services such as access control. There are different degrees of decentralization of information security, determined by how much of the security services are performed at a common, central computer.

There are significant benefits from a decentralized security scheme. From a security standpoint, there is no longer the concept of a "super-user" that has the capability to read and write any of the system's data files. Only each end-user has the capability to decrypt files they have access to, and only the end-user can sign data blocks for writing. The creator / modifier of stored data creates the encrypted keys for users to access the data, because this function can not be performed by anyone who does not have access to the data.

2.2 Performance Problems with Existing and Previously Proposed Systems

The standard office automation system in use today is comprised of many computers located on various users' desks, attached to file server(s) through a high-speed local area network. 10 Megabits per second was the standard local area network speed until recently, when 100 Megabit Ethernet hit the mainstream marketplace. A second major change in local area networking is the use of switching hubs. These allow multiple pairs of users to communicate at full bandwidth, which was not possible with the shared-medium of previous LANs.

Computer hard drives have been increasing in speed each and every year, and more advanced techniques are being developed to maximize the performance of existing and new hard drives. A commodity SCSI disk can sustain transfer rates of about 200 Megabits per second. Traditional file servers held several SCSI disks striped in a RAID configuration to increase speed and reliability, but the backplane connecting the SCSI controllers is now a bottleneck. If we have eight SCSI drives communicating at 200 Megabits per second, the total transfer rate is 1600 Megabits per second. The standard PCI bus in Intel

based computers can only support 1056 Megabits per second (33 MHz * 32 bit,), so the drives are limited by the bus. This problem is exacerbated because the buffers on the drives can transfer data at over 3 times faster than their media transfer rate. To compensate for this deficiency, most file server manufacturers have resorted to expensive proprietary high-speed backplanes. Even this specialized hardware will not be able to keep up with more than 8 hard drives in the near future. There is also a problem with memory bandwidth in the server which is a bottleneck in some faster systems.

There have been attempts to alleviate the bottleneck at the file server by partitioning the directory tree across multiple file servers, but this does not work as well as one might expect. Since the number of file servers is still small, hot spots will still cause throughput problems at particular file servers at various moments in time. With a distributed network file system, the aggregate system bandwidth increases almost linearly with respect to the number of network devices comprising the file system [1]. By striping the data along with parity, the data can still be accessed in spite of a single drive failure. Since each drive in the proposed system is on a separate computer, the probability of a second drive failing before the first failure can be replaced is very small. A drive failure in a modern file server can cause a second drive to fail. For example, one drive could catch fire and damage the other drives. The physical separation of the drives in a network attached file system leads to a very reliable system. The TickerTAIP parallel RAID scheme developed at Hewlett-Packard laboratories has many of the benefits of a network attached storage system, but lacks the reliability of a physically distributed system [4].

The increasing speed of hard disks and networks along with bandwidth intensive applications necessitate high bandwidth network file systems. If the storage is distributed across multiple network segments, the file system could provide aggregate transfer rates well over the rate of a single network connection (1 Gigabit/second soon.) For example, if ten users request data uniformly striped across ten drives, the aggregate transfer rate of the file system could theoretically hit 2 Gigabits per second (10 * 200 Megabits per second for each drive.) Many network file system have used aggressive file caching to improve performance, but with high-speed network storage, caching files on a local disk may become a thing of the past. Caching in local RAM will, of course, still be faster than retrieving data off the network. Network attached disks should provide excellent performance during times of burst traffic because the data will be striped across a large number of disks. By choosing the hashing algorithm correctly to evenly spread out the loads, slow downs due to burst hot-spots can be avoided.

3. Data Structures

The system we propose is called Network Attached Storage with Decentralized Security NAS/DS, building on the Network Attached Secure Disk work performed at CMU [6]. There are four basic new data structures that are used with this system. Secure data objects contain a block of encrypted user data, along with sufficient information to validate the sender of the data, and the data integrity. File objects consist of the file ID, associated key file ID, and one or more secure data objects. Key objects are associated with a file or group of files and contain sufficient information (less the user's private key) to decrypt the file

blocks. Finally, certificate objects are stored on each network drive and are used to determine if a particular user is permitted to write or delete data from a file object.

3.1 Secure Data Objects

The secure data object shown in Figure 1 is the basic unit of data that is written to and read from the network drives. Each object contains sufficient information to verify the cryptographic controls on the data. The HMAC (Hash-based Message Authentication Code) proves the integrity of the data and the sender of the data [2]. The timestamp prevents a replay attack - sending an old block instead of a new one. The IV (Initialization Vector) is needed to decrypt the block assuming a CBC (cipher-block chaining) mode encryption is used. This is further described in Section 4. The client computer is responsible for making sure that any secure data object created has a timestamp (counter) that is greater than the block it is replacing, as the drive will not write the data otherwise. This prevents a hacker from writing an old stored block over a new one. Each file, if sufficiently large, is divided into a sequence of secure data objects. The size of the secure data object in this study is 64000 bytes. The larger the secure data object, the less overhead that is associated (per byte) for encrypting and for placing a header on the data. The problem with large secure data objects is that the entire object must be re-written even if a single byte is modified, unless a code-book encryption is used (and this is not recommended).

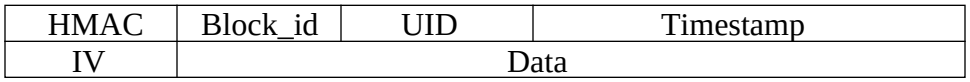

Figure 1. Secure Data Object

3.2 File Objects

A file object contains some meta data for maintaining the file. It consists of the file identifier (fid), key file identifier (key_fid), and a list of block numbers that form the file. When a user wants to read an entire file or a portion of a file, she first reads the file object. This tells what file the encrypted cryptographic keys are stored in, as well as the list of blocks that make up the file. The file object is stored on one network disk, however, the secure data blocks identified by the block_id may be stored across multiple disks. It is the job of a higher level file system built on NAS/DS to manage this hierarchy. Upon reading the file object and key file object, the user can read and write an arbitrary number of secure data objects (assuming permission was granted.)

3.3 Key Objects

Each key object shown in Figure 3 contains two types of information. At the beginning of the key object are the file identification field (fid) and the user identification field (uid.) These are used to determine if a request to modify the actual key object will be allowed. The other information contained in the key object are sets of three-tuples containing a uid, key', and permission bits. The key' is the symmetric key used to encrypt the file — encrypted with the user's public key. The permission bits are similar to the Unix file sys-

fid	key_fid
Block_id 1	
Block_id 2	
...	
Block_id N	

Figure 2. File Object

tem's permission bits. Unix has separate permission fields for the files owner, group, and others; there would be three entries in the key object for similar granularity.

fid		uid
uid	key'	permission bits
uid	key'	permission bits
...		
gid	key'	permission bits

Figure 3. Key Object

3.4 Certificate Objects

Each network drive contains a certificate object, shown in Figure 4, that the drive will use to decide whether or not to grant a write operation. Information for each individual user (UID) and group (GID) is kept as a row in this file. The Key_{MAC} is the shared-secret used for the HMAC generation and verification between each user and the drive. The timestamp field is updated each time a file block is written and is used by the drive to prevent a reply attack. The optional quota field is used by the drive to prevent a particular user from writing more than their allowed amount of data to the drive.

fid		uid		
UID	Key_{PUB}	Key_{MAC}	Timestamp	Quota
UID	Key_{PUB}	Key_{MAC}	Timestamp	Quota
UID	Key_{PUB}	Key_{MAC}	Timestamp	Quota
GID	Key_{PUB}	Key_{MAC}	Timestamp	Quota

Figure 4. Certificate Object

There is the obvious problem of who gets to write the certificate object. One possibility is that with physical access to the drive, a public key that will be used to validate write requests to the certificate object can be loaded manually. Another possibility is that a priv-

ileged user could log into the drive via the network using some secure protocol to write the file. Even if this file is written by a hacker, the confidentiality of the data is still maintained, but a denial of service will likely occur. This assumes that the users do not rely on the certificates stored on the network drives to get public keys when encrypting the symmetric keys, since there is no reason to do this. An error such as using this certificate object for encrypting keys is exactly why the security of this system must be carefully thought out. Subtle errors are usually the key to defeating a security system such as this.

4. System Design and Operation

In this section, the general design of the network attached storage with decentralized security will be discussed. The operation of the system will be described, including such operations as data reads and writes.

4.1 Data Security

The basis of data security in this system lies in the secure data objects. Provided that the user obtains the symmetric encryption key (RC5 key) from the key object, the secure data object contains sufficient information to protect the confidentiality and integrity of the data it contains. This simply means that even if an adversary is able to obtain all of the data that is stored on the network disks, and snoop all of the data that traverses the network, data confidentiality and integrity are still maintained.

This system uses a keyed-hash approach to authenticate the writer of a data block. In particular, the MD4 hash algorithm is used in a manner similar to MD5-HMAC [2]. Note that although MD4 has known weaknesses, it still provides weak-collision protection which is sufficient for this application. Using HMAC for writer authentication has the disadvantage that the network disk contains sufficient information to forge a data block. Keyed-hash functions have the property that the verifier of the keyed-hash can also create the keyed-hash since it is symmetric - same operation for generating and verifying. If a network disk is compromised with this scheme, it is possible that the adversary could write information to the drive. If the system is built properly, this would require that they were able to obtain the writer-authentication keys from the drive or gained physical access to the hardware. These keys could be stored encrypted for greater security. The compromised system still prevents an adversary from accessing any encrypted data stored on the drive or any data in transit. The alternative to HMAC for user authentication is using a digital-signature, but this is too processor intensive for current computers. Perhaps in four to five years processors will be fast enough for this approach, and end-to-end authentication will be feasible.

This scheme works by including an HMAC as part of each secure data object. The drive is able to determine that the writer corresponding with the provided uid or gid that created the block has write access to the drive by using the writer-authentication key stored in the certificate object. Only someone with access to this key would be able to create HMAC on the block.

Performing an HMAC is substantially faster than a signature generation. The main weakness is the loss of end-to-end integrity assurances. There is no guarantee that the drive did not corrupt the data, since the ability to verify a keyed-hash implies the ability to generate

a new one. The corrupted data could almost certainly be detected since it is encrypted. A hash of the plaintext could be appended to each block before it is encrypted to further ensure data integrity.

This system provides the following cryptographic services on secure data objects:

- **Confidentiality** - Each file is written in 64000 byte blocks; the last block may be smaller. Each of these blocks is encrypted with a fast encryption algorithm. This analysis used the RC5 encryption algorithm in CBC mode with a 128-bit key [10]. This ensures the data can not be read without the accompanying RC5 key. This key is used to encrypt each block in a file or group of files. The key is generated by the user upon the creation of a file or file group. This key must be provided to the other users that need access to this file (if any); this is done by encrypting this RC5 key with the public key of each user (UID) or group of users (GID.) These encrypted keys are stored in a key file associated with the data file. Both the data and key files are stored on the network disks.
- **Data Integrity** - Provided by the HMAC of the data. The key used for HMAC can be sent to the drives using RSA when each new user is added to the system.
- **Authentication** - Just as in the other schemes, no user authentication is necessary for the read operation on this system since all of the data is encrypted. Authentication for the write operation is provided by the HMAC. The number of cryptographic functions performed by the host computer and disk computer is shown in Figure 5. The network disks only need to perform one HMAC (Hash) function over the block for each read or write operation, where the host computer needs to perform a hash and encryption. Since the encryption takes longer than the hash (see Table 1), the network disk has less than 1/2 the work to perform as the host computer. This will help the network disks scale to multiple hosts. The bottleneck at the network disk is the network interface as the HMAC function takes substantially less time than the time to send the packet over the 100 Mbps link.

Operation	Host	NAS
En/Decrypt:	1	0
Hash:	1	1
Signature:	0	0
Verification:	0	0

Block Write

Operation	Host	NAS
En/Decrypt:	1	0
Hash:	1	1
Signature:	0	0
Verification:	0	0

Block Read

*note: A key-exchange operation is needed upon file creation for each user that needs access. Since this is only done once for each file (or group of files) regardless of the number of blocks, it is omitted.

Figure 5. Number of Cryptographic Operations

4.2 Basic Operation

The network file system will support the basic distributed file system calls. Both user data files as well as directory information are stored as data objects on the network drives. Higher level operations are accomplished via the simpler operations. For example, revok-

ing permission for a particular uid on a file is accomplished by rewriting the key object for the file. This will prevent the user from writing any new data written to the file. If it is necessary to prevent a user from reading data from files that they were previously able to read, the data must be re-encrypted. Of course, this accomplishes little since the user could have simply cached the file on their local system. The data objects on the network disks are subject to reads, writes, and deletes as described below.

4.2.1 Block Write

The write operation starts with encrypting a block of data. This provides data confidentiality as the data traverses the network as well as while it is stored on the disks. The block is then given a timestamp and an HMAC is appended. This forms the secure data object, and it is sent to the network disk as shown in Figure 6. If a block is simply modified, the file

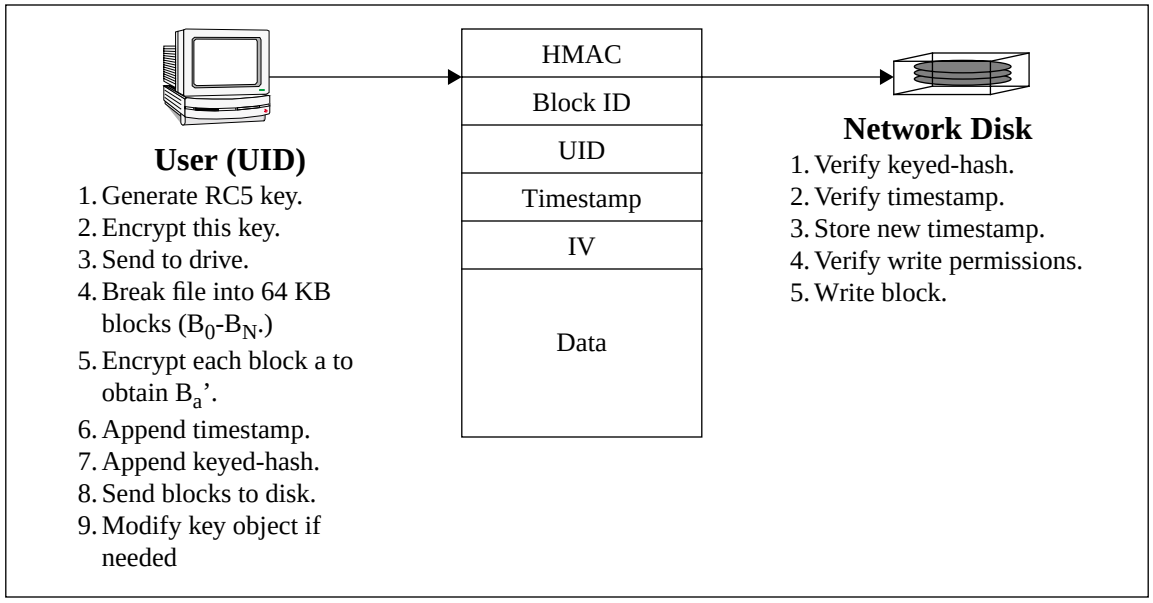

Figure 6. Writing a File

object does not need to be changed. However, if a block is added or deleted, the block identifier needs to be added or deleted from the file object. Note that writes to the file object are protected by an HMAC just as writes of any other objects on the network drives.

The actual HMAC on the secure data object does not necessarily need to be written to the disks. The old HMAC is not sent when a block is read since only the original writer of the block and the network disk could use it.

4.2.2 Block Read

For the read operation the disk needs to append a keyed-hash for the user requesting the block as well as a timestamp newer than the one last received from that user. For group access, the keyed-hash calculated by the writer could be used. For individual access, the new keyed-hash must be calculated because the reader does not have access to the writer's writer-authentication key. This exchange is shown in Figure 7.

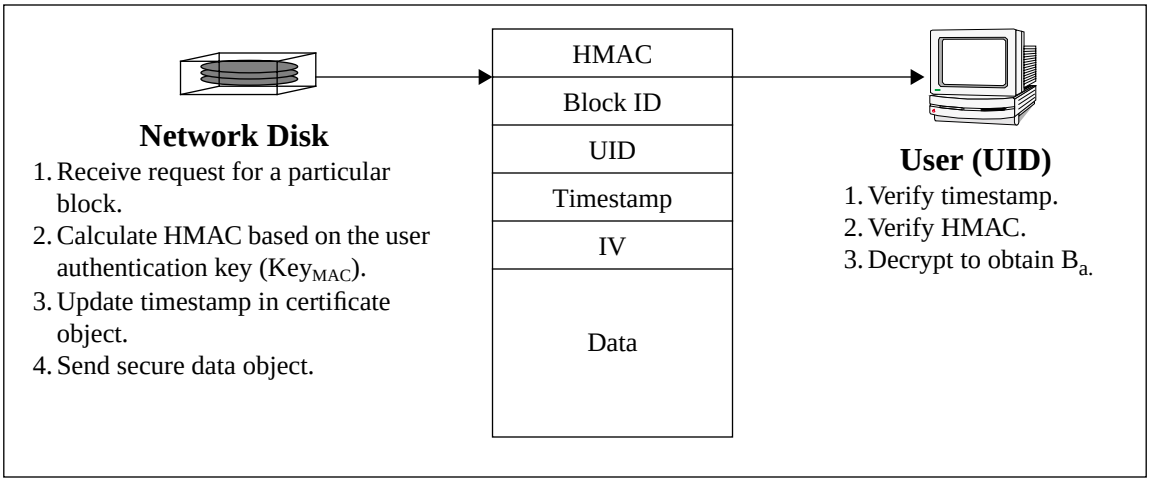

Figure 7. Reading a File

4.2.3 Block Delete

A delete request is handled basically the same way as a write. The drive verifies that the uid in the delete request is the owner of the file, that the signature on the request is valid, and the timestamp is in order. If these conditions are true, the data object is deleted. It is the responsibility of the file system software on The Final System.

5. System Performance

A test system was built to verify the feasibility of this system. This section describes the exact hardware and software of the prototype system, along with performance measurements. The cryptographic component performance, network performance, as well as the system performance is presented.

5.1 System Description

The system used for this study is comprised of 2 Motorola VME boards connected to a 100 Mbps hub as shown in Figure 8. The hub in use does not provide switching capabilities, so the aggregate performance is limited to 100 Mbps. An attempt to use a Cisco Catalyst 2900 switch was thwarted by unsolved device driver issues. Both of the computers in this test setup run the VxWorks real-time operating system. This operating system was chosen to limit the variables introduced with a large multi-user OS such as Unix or Windows.

5.2 Cryptographic Component Performance

The performance of various cryptographic algorithms on a variety of platforms has been tested to verify the feasibility of the proposed file system [5]. For this study, the performance of the hash algorithms to be used in an HMAC (MD4 or MD5) and the encryption algorithm RC5 is shown in Table 1.

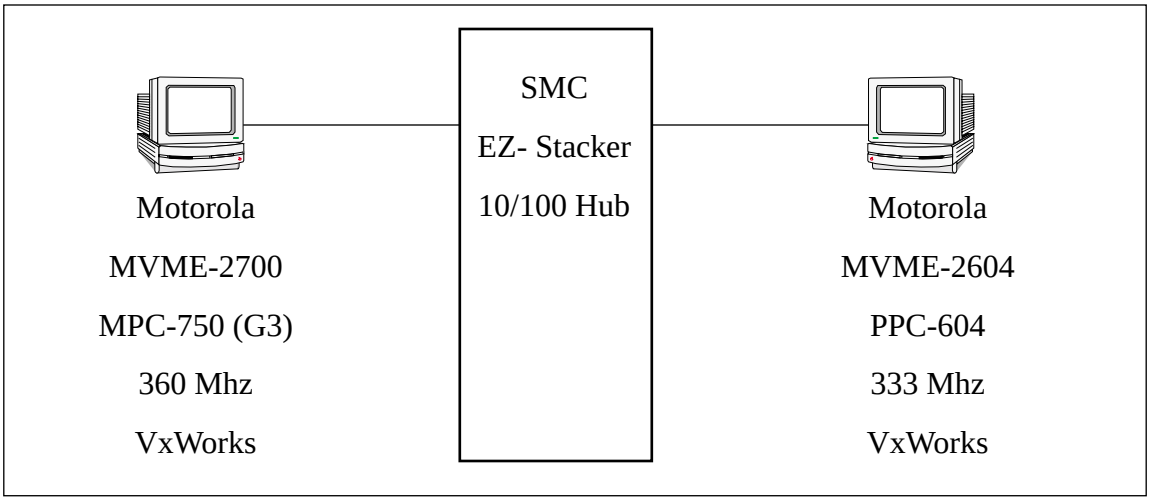

Figure 8. Hardware Configuration

Operation (64000 Bytes)	**MVME-2604 PPC604/333**	**MVME-2700 MPC750/360**
MD4	1.39 ms	1.31 ms
MD5	3.20 ms	2.30 ms
RC5 Encrypt	4.27 ms	3.20 ms
RC5 Decrypt	5.33 ms	4.60 ms
RSA-512 Sign	25 ms	21.2 ms
RSA-512 Verify	2.3 ms	2.0 ms

Table 1. Cryptographic Operation Time - 64000 Byte Block

5.3 Network Performance

The performance of the networking stack implemented in VxWorks for these Motorola boards needed to be investigated to ensure the network was not responsible for a bottle-neck. The network buffers and timings were tuned to obtain the results shown in Table 2 it is clear that using UDP, the network performance is not a limiting factor. The difference

Protocol	**MVME-2604 PPC604/333**	**MVME-2700 MPC750/360**
TCP	77 Mbps	80 Mbps
UDP	96 Mbps	96 Mbps

Table 2. Network Protocol Performance

between the UDP performance and the optimal 100 Mbps is simply the overhead of lower

level protocols. The VxWorks based computers are capable of generating the UDP datagrams significantly faster, the network is the bottleneck.

5.4 System Performance

The combined system performance for the block read and block write operation was tested for this scheme. The block write followed the protocol shown in Figure 6, thus involved the MPC750 performing a 64000 byte RC5-CBC encryption followed by an MD4 HMAC operation. Upon receipt of the block, the PPC604 acting as the network disk was required to verify the MD4-HMAC, and write the block into memory. The writing of the block to a actual disk was not tested at this point. However, the performance of the Seagate Cheetah drives is much greater than the 100 Mbps network, and should pose no bottleneck.

The read operation tested was similar to the protocol shown in Figure 7, and involved the PPC604 at the network disk performing an MD4 HMAC operation, and the MPC750 at the workstation performing the slower RC5-CBC decrypt operation followed by an MD4 HMAC operation. The difference was the client performing the read operation for these measurements send the datagram over the network instead of receiving it because the code to read from the network at the client was not yet finished. The performance of this read and write operation is shown in Table 3.

Operation (64000 Bytes)	**Overall Performance**
Block Write	66.4 Mbps
Block Read	56.5 Mbps

Table 3. System Performance

6. Conclusion

This paper presented the details of the Network Attached Storage with Decentralized Security system along with performance measurements. It is clear that this type of system is feasible with today's computing power, and will become even more attractive as processors become faster. The write operation was able to run at over 2/3 of the optimal performance, while the read was limited to 59% of optimal performance. This distributed file system solves many of the performance and security problems in existing systems today. This system protects user data confidentiality and integrity from the moment it leaves the client computer. The distributed disks should perform substantially better than centralized file servers, and provide better reliability. Having the security functionality decentralized will improve performance and scalability. It also removes the single point of failure that plagues many proposed centralized security schemes to date.

REFERENCES

[1] Thomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, and Randolph Y. Wang. *Serverless Network File Systems,* ACM Transactions on Computer Systems, Feb. 1996, pp 41 - 79.

[2] M. Bellare, R. Canetti and H.krawczyk. *Keying Hash Functions for Message Authentication,* Proc. Advances in Cryptology, pp 1-15, CRYPTO, 1996.

[3] Steven M. Bellovin and Michael Merritt. *Limitations of the Kerberos Authentication System.* Computer Communications Review, 1991, pp 1 - 15.

[4] Pei Cao, Swee Boon Lim, Shavakumar Venkataraman and John Wilkes. *The TickerTAIP Parallel RAID Architecture.* ACM Transactions on Computer Systems, August 1994, pp 236 - 269.

[5] William E. Freeman, Ethan L. Miller. A*n Experimental Analysis of Cryptographic Overhead in Performance-Critical Systems*. IEEE Mascots ‘99, pp 348-357.

[6] Garth A. Gibson, David F. Nagle, Khalil Amiri, Fay W. Chang, Howard Gobioff, Erik Reidel, David Rochberg and Jim Zelenka. *Filesystems for Network-Attached Secure Disks.* www.pdl.cs.cmu.edu/PDL-FTP/NASD/CMU-CS-97-118.pdf.

[7] N. Haller. *The S/KEY One-Time Password System.* IETF RFC-1760.

[8] Nelson E. Hastings. *TCP/IP Spoofing Fundamentals.* Proceedings of IEEE Fifteenth Annual International Phoenix Conference on Computer and Communications, 1996, pp 218 - 224.

[9] Jim Reid. *Plugging the Holes on Host-based Authentication.* Computers and Security, 1996, pp 661 - 671.

[10] R. Rivest. *The RC5 Encryption Algorithm,* RSA Labs’ CryptoBytes, Vol. 1 No. 1, Spring 1995. http://www.rsa.com/rsalabs/pubs/cryptobytes.html.

[11] Chandramohan A. Thekkath, Timothy Mann and Edward K. Lee. *Frangipani: A Scalable Distributed File System.* ACM Operating System Principles, 1997, pp 224 - 237.