An Experimental Analysis of Cryptographic Overhead in Performance-Critical Systems

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

[1]Dept. Computer Science, University of Maryland, Baltimore MD
[2]Laboratory for Telecommunication Sciences, Adelphi MD

wef@lts.ncsc.mil, elm@csee.umbc.edu

ABSTRACT

This paper studies the performance implications of using cryptographic controls in performance-critical systems. Full cryptographic controls beyond basic authentication are considered and experimentally validated in the concept of network file systems. This paper demonstrates that processor speeds have recently become fast enough to support cryptographic controls in many performance-critical systems. Integrity and authentication using keyed-hash and RSA as well as confidentiality using RC5 are tested. This analysis demonstrates that full cryptographic controls are feasible in a distributed network file system, by showing the performance overhead for including signature, hash and encryption algorithms on various embedded and workstation computers. The results from these experiments are used to predict the performance impact using three proposed network disk security schemes.

1 Introduction

With the growing connectivity of the world, many systems that were immune from remote attacks due to network isolation are now vulnerable. Many embedded systems are now connected to the Internet in some manner which gives rise to the need for security on these devices. Many of these embedded systems perform real-time or performance-critical functions making system performance the most important characteristic. Lack of computer power has prohibited cryptographic controls from being used to secure these systems. Cryptographic authentication via a kerberos system (if anything at all) is the most that is implemented in the majority of current performance-critical systems.

There is a strong movement towards transport layer security for devices connected to the Internet. However, one study has shown that implementing the secure sockets layer (SSL) protocol on a web server has resulted in a two orders of magnitude decrease in throughput [1]. This paper shows that using an e-mail type security scheme on data transactions of 64 KBytes imposes an approximate overhead of 30 ms for writing and 10 ms for reading. To put this in perspective, modern disk drives take 8 ms to seek to a block and 5 ms to read or write a 64 KByte block. Using a simpler keyed-hash (MD4) with encryption (RC5) imposes an overhead of only 6.5 ms on the host computer and 1.6 ms on the network disk for reads or writes.

To determine the amount of time needed to perform cryptographic controls, several cryptographic algorithms were tested on a variety of embedded and workstation computers. The results from the performance timings of these algorithms are combined with three proposed security schemes for securing a network file system to determine the overhead of cryptographic controls on modern processors. This study also quantifies current beliefs that cryptographic controls impose a great penalty on systems with small data transactions. The projected speed of computers over the next few years is predicted using growth numbers from the previous ten years. This shows the time to perform cryptographic controls will take less than 1/2 of the time in 2002 as was required in this study.

Computer power has been growing at an exponential rate for many years now. This paper demonstrates that computing power has recently become fast enough to allow cryptographic controls in performance-critical systems. Cryptographic controls will be feasible in real-time systems with more stringent time requirements in the near future.

This paper is organized as follows: Section two briefly presents the fundamentals of data protection. Section three describes the cryptographic design and presents three schemes for providing cryptographic controls in a distributed file system environment. Section four presents the results from testing various encryption algorithms on five different hardware platforms and quantifies the total cryptographic overhead. This section also discusses historic processor performance growth and expected future performance. Section five presents the conclusions drawn from this study.

2 Cryptographic Control Basics

Data security is defined by four attributes: confidentiality, data integrity, authentication, and non-repudiation. Confidentiality is the means to protect against disclosure of the data.

Data integrity is the assurance that the data received was exactly the data sent. Authentication is the mechanism for verifying the identity of the requestor. Non-repudiation is the ability to prove transmission and receipt of data. These attributes and the cryptographic tools used to provide them are briefly described below. For a more complete description of basic cryptographic tools, see [13].

2.1 CONFIDENTIALITY

Confidentiality is needed whenever the data in transit or in storage must be protected from unauthorized disclosure. This is typically provided by some form of data encryption. There are two main types of data encryption schemes: public-key cryptography and secret-key cryptography.

Secret-Key cryptography:

- Common algorithms: DES, TDES, IDEA and RC5.
- Secret key must be exchanged between users.
- Very fast to perform.
- Same key for encrypting and decrypting.

Public-Key cryptography:

- Common algorithm: RSA.
- Separate encrypt (public) and decrypt (private) keys.
- Very slow to perform.

This study will use the RC5 algorithm for the secret-key encryption. This algorithm has a variable length key, with lengths between 56 and 128 bits commonly used. The RSA challenge to break a 56-bit RC5 code was completed in 1998 using tens of thousands of computers. For this system, RC5 will be used with a 128-bit key, which should be more than sufficient. The 128-bit key is 2^{72} times harder to break than a 56-bit key. RC5 was chosen because it is believed to be quite strong and the algorithm is very fast to perform.

Public-key cryptographic algorithms avoid this secret-key exchange problem. RSA is the most common public-key cryptographic algorithm, but is far too slow for encrypting user data in a performance-critical system. RSA is used for key exchange and digital signatures in this study. Each user of RSA will have a public and private key forming a key-pair. The public key is used to encrypt data and verify signatures. The private key is used to decrypt data and create signatures.

2.2 AUTHENTICATION

User authentication is needed in any environment where the system must restrict who may or may not have access to system resources or information. User authentication is often done by a simple password sent in the clear which can be intercepted by an adversary. Some systems attempt to provide authentication services based on a workstation IP address, but this has been shown to be very weak [7]. Strong user authentication can be performed using a keyed-hash or digital signature. The keyed-hash works by combining a secret key with the message before performing the hash. The recipient can then take the transmitted message, combine their copy of the secret key and rehash the message. If this hash matches the hash the sender provided, then the message's authenticity and integrity are validated. HMAC is a common keyed-hash scheme for message authentication, defined in RFC-2104 that uses MD5 or SHA. Many schemes for keyed-hash authentication are believed to be strong, but the HMAC scheme has a proof of its strength [2]. A digital signature can be applied to the entire message or a hash of the message to prove the identity of the creator and the integrity of the data.

2.3 DATA INTEGRITY

Data integrity is the means of ensuring that the data received was indeed the data transmitted. For a network file system, it must detect modification in transit to the drive, while on the drive, and in transit to the receiver. Using the keyed-hash or digital signature over the data (or hash of the data) for user authentication automatically guarantees the data integrity, since any modification will cause the authentication to fail. Data integrity can be attacked by retransmitting an older write command after the data has been updated. This would result in the old data overwriting the new data. This type attack is called a *replay-attack* and is prevented by including a timestamp or one-up counter with each block to write.

One of the first commonly used hashes is called MD4. It was developed by Rivest, and is described in RFC 1320 [9]. This takes a message and performs a one-way function that results in a 128 bit hash value. This hash algorithm suffers from a serious collision problem that allows one to easily find two messages that hash to the same value [4]. There still are no known ways to produce a second message that hashes to the same value as a given first message which is the most important requirement for a hash to have in the keyed-hash system. MD5 and SHA are basically variants of the MD4 hash algorithm. MD5 solves the collision problem of MD4, and is fully described in RFC 1321 [10]. The only known problem with MD5 is the 128 bit key length is not considered sufficient for some systems [5]. The so-called birthday attack means that you only need 2^{64} messages to find two strings that hash to the same value with a 128 bit hash. The secure hash algorithm (SHA) is a very strong algorithm that produces a 160 bit hash value. For the purposes of user authentication and data integrity, MD5 is sufficient for the file-system design this paper presents. In the opinion of the authors, MD4 is also sufficient. MD5 takes about 50% longer to computer than MD4 and SHA is about 40% slower than MD5.

2.4 NON-REPUDIATION

Non-repudiation is needed when verification of both the transmission and receipt of information is required. An example of this is with electronic stock trading. The person initiating the trade may try to deny making the transaction if the stock price plunges. The broker may attempt to deny the transaction if they fail to perform the trade. Non-repudiation is usually achieved by a two-way transaction. One user sends some information protected by a digital signature. The receiver then returns a signed receipt of that information. This cannot be provided by the keyed-hash authentication scheme described above since both the sender and receiver share the same secret key, thus both could have generated the message. Non-repudiation is typically provided via a cryptographic signature using the private key. The drawback of providing non-repudiation is that public-key cryptographic algorithms are very slow. Current implementations of RSA are over 1000 times slower than DES. The time needed to perform a RSA operation is dependent on modulus length (M). The time to encrypt or verify a signature is $O(M^2)$. The time to decrypt or generate a signature is $O(M^3)$. It is currently believed that 512 bits is the minimum modulus length to provide sufficient security.

3 Cryptographic System Architecture

An example of a performance-critical system is a network file system. When a request to read or write a block of a file is made, a response is expected very quickly. The system is only acceptable if the information is delivered correctly and quickly. Cryptographic security has not been widely implemented in distributed file systems due to performance concerns. The systems in use today clearly show performance is more important than security.

The primary reason cryptography has not been used in file systems and other performance-critical systems is that system performance was limited by processor speed. With the rapid increase in processing power recently, many of these systems have become limited by other components. With file servers, the primary performance limit is caused by the back-plane and disk speed. This leaves the CPU idle much of the time; time that could be used for cryptographic processing.

This study proposes three schemes for performing cryptographic controls on a distributed file system that uses network disks. A network disk is basically a hard drive coupled with a board computer with a network interface. In a fully implemented system, there would be many of these disks, with the files spread across several in a RAID configuration [6]. The file system discussed in this study provides a raw block read/write/create service. Each of these schemes is analyzed to determine what cryptographic functions are performed by the host and network disk for a read and write operation. Combining this information with the performance analysis that follows will show the cryptographic overhead for this system.

3.1 File System Characteristics

Modern file systems must be fast, robust, and secure. The requirement for speed means that file systems must provide megabytes of data per second, potentially to many different clients simultaneously. However, most of the time spent in servicing a particular distributed file system request is spent in hardware delays: network transmission and (if the data is not found in cache) reading the file from disk. A file system CPU typically needs fewer than 40,000 instructions to service a file request. This allows a modern file server to handle 2500 - 5000 requests per second if the network protocol stack is offloaded to a different processor. Since file service is embarrassingly parallel, though, file servers often include multiple processors to speed up the rate of file delivery.

Robustness, on the other hand, is not affected greatly by CPU speed. The main problem that file systems must deal with is preventing data loss. Data loss can be avoided by intelligently designing the on-disk data structures and the sequence of operations in disk requests. Avoiding data loss does not typically cause a large increase in the number of instructions required to execute a single request.

Even though current file system security is very weak, security of a user’s files is important in many commercial and government installations. In NFS, the most popular distributed file system, the main security function was not designed for security, but is a technique for preventing two clients from simultaneously accessing different versions of the file [8]. Some more advanced systems use a centralized authentication scheme such as Kerberos, but still store and transmit data in the clear.

Secure architectures should be designed so one component failure does not compromise the entire system [3]. For example, if one user’s private key is compromised, data that user did not have access to should still be protected. For systems using a Kerberos authentication system, all of the security lies in the authentication server. If this is compromised, the entire system is compromised.

3.2 Cryptographic Initialization

All of the security schemes that follow require the network drives to store each user’s public key or hash key. This key is used for signature verification and writer-authentication key exchange. Each user of the system is assigned a user identification number (UID) that has an associated public and private key. For group access, each group is assigned a group identification number (GID). Each member of the group needs a copy of the group’s private key. The means of generating these key-pairs and the means of securing the private keys and other data on the local workstation is beyond the scope of this study.

When a new drive is added to this system, a certificate object is written to the disk. All future writes to this disk are checked using the public keys stored in these certificates. The certificate object then can be added to by the owner of that file (the system administrator.) This process is illustrated in Figure 1.

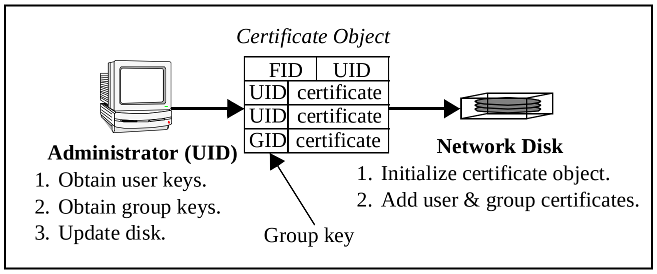


Figure 1. System Initialization

3.3 Cryptographic Schemes to Support Performance-Critical File System Access

This section presents three methods for providing data security in a network file system. The first scheme is very similar to cryptographic support in e-mail systems such as PGP. Each block is independently encrypted, hashed and signed before it is written. The symmetric key used to encrypt the block is encrypted with each private key whose holder needs access to the block. A large number of blocks and perhaps files will share the same symmetric encryption key. The second scheme removes the burden of calculating a signature verification on the network drive at the cost of adding a hash function to the host computer for each write operation. This would be used in a situation where the workstation computers are significantly faster than the drive computers. The third scheme replaces the public-key support for proof-of-origin with a keyed-hash scheme.

3.3.1 Scheme 1: Best Security / Worst Performance (Digital Signature)

This scheme provides security on each block of data similar to e-mail security schemes. To perform a write operation, each block is encrypted, hashed and signed. The network disk verifies the signature and hash of the block before it is written. A timestamp must be present on each block to prevent an old block being retransmitted to the drive to overwrite a new block (replay.) Writing a block with this scheme is illustrated in Figure 2.

There is no cryptographic security utilized by the network disk to service a read request. The drive simply provides the data block requested as shown in Figure 3. The data block is protected by the writer in such a way that it is useless without proper private keys. An important characteristic of this system is no sensitive data is stored on the network disk or transmitted over the network. This comes at the cost of the computationally expensive signature generation and verification. The signature verification is much simpler to compute, but this is done on the network disk which will have a low-cost processor. The signature generation is only performed on the more powerful workstations.

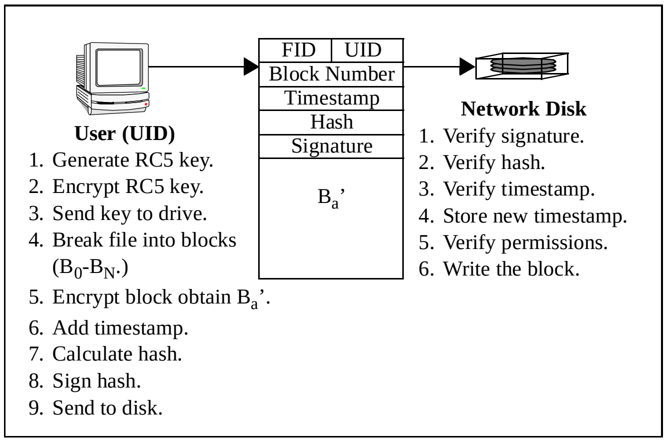


Figure 2. Writing a File - Scheme 1

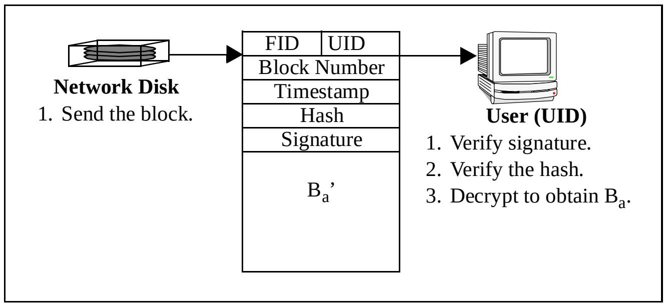


Figure 3. Reading a File - Scheme 1 & 2

The cryptographic controls provide the following services:

- **Confidentiality** - Each file is written in 64KB blocks; the last block may be smaller. Each of these blocks is encrypted with a fast encryption algorithm. This analysis uses the RC5 encryption algorithm [11] with a 128-bit key. 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 disk.
- **Data Integrity** - Each file has a cryptographic hash calculated over the data. For the purpose of this analysis, MD4 [9] and MD5 [10] were investigated. This ensures that if the data was modified in any way, the hash will not match the one provided. The provided hash is protected by the authentication / non-repudiation mechanism described below.

- **Non-repudiation** - A cryptographic signature is calculated over the hash using the user's private key. By using an individual key, non-repudiation can be provided since only that user could have signed the data. For this service, the reader of the information would need to store a signed copy of the hash on the drive, and provide a signed-receipt to the originator.
- **Authentication** - User authentication is provided by encrypting the RC5 key with only authorized user's public keys. Any user not possessing the proper private key will not be able to decrypt the RC5 key or the data. Authentication for the write operation is provided by the signature of the hash.

The data is provided confidentiality protection before it leaves the host computer by encrypting all of the sensitive data using the RC5 algorithm. This protection is only as strong as the RC5 key used and the RC5 algorithm. It is believed that the 128 bit strength of this algorithm is very strong, so only the key security needs to be looked at.

There are two problems that need to be dealt with. First, how hard the key is to guess without direct access to the key data? Computers are inherently bad at making random numbers because they are largely deterministic devices. Zimmerman overcame this problem by injecting some non-deterministic information into the key generation process. The period of time between a user's key strokes was measured to increase the entropy of the key. A similar process could be used to create the "random" number for the RC5 key in this scheme. Hardware key generation using techniques such as a noisy diode or multiple free-running oscillators could also be used.

The second problem with the RC5 key is that the key needs to be given to each and every reader of the data. This is true of any symmetric cryptographic algorithm, and this class of algorithm is the only one fast enough for this type of system. The key is exchanged using public-key cryptography, in particular RSA. Each user of this system has access to the other user's public keys using some certificate scheme. This allows each user to send data to another that is encrypted in such a way that only the holder of the private key can read it. This encryption algorithm is not acceptable for general use because it is very slow. It can be used however, for sharing small amounts of secret information. This leads to its most common use - secret key exchange. In this system, the RC5 key is exchanged between users using this public key cryptography. The key is encrypted with each user's public key that needs access to the file, and these encrypted keys are stored on the network storage system. Many files can be encrypted with the same RC5 key if desired, but they must have the same access list.

The next service that is provided is data integrity. Since the data in transit is encrypted, an adversary would have a very difficult time creating useful modified plain-text at the receiver. Since RC5 is a block cipher, modifying the cipher-text would result in gibberish in the corresponding plain-text block. Nevertheless, for many systems, data integrity must be ensured. This system provides data integrity using a digital signature and a hash function.

The digital signature uses the same public key cryptography as the RC5 key exchange, but uses the user's private key. The entire block could be protected using a digital signature, but RSA is simply too slow. A way to speed this process up is to reduce the size of the block using a cryptographic hash function. This produces some fixed-size block of information that can be recalculated quickly. The integrity is provided by performing a digital signature on the hash of the data block. The digital signature not only ensures that the appended hash value is authentic, but verifies the originator of the hash signature. Only the holder of the private key whose public key verifies the signature could have generated the appended signature. This approach is not as strong as signing the entire block since the security is no stronger than the weaker of the signature algorithm and the hash algorithm. A failure in either one will compromise this scheme.

The primary concern with this scheme is the performance. The slowest operations in this system are the public key operations (key-exchange and signature.) The cryptographic overhead is needed for each operation is shown in Figure 4.

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

Block Write

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

Block Read

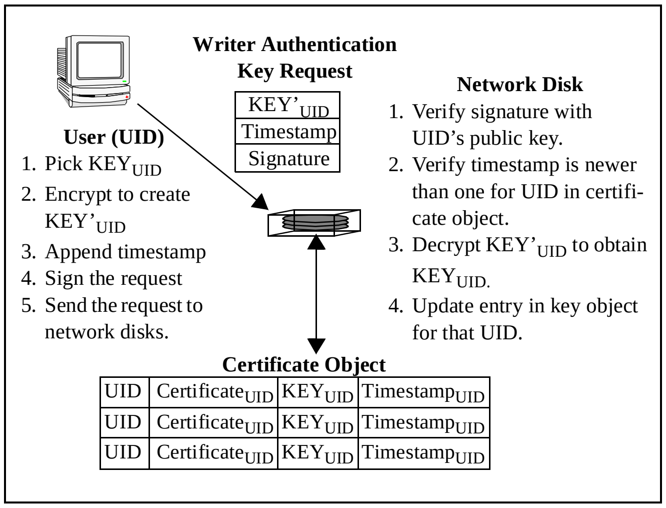
*note: A key-exchange operation is needed upon file creation for each user that needs access. Since this is only done once for each file (or group of files) regardless of the number of blocks, it is omitted.

Figure 4. Number of Cryptographic Operations - Scheme 1

3.3.2 Scheme 2: Good Performance / Good Security (keyed-hash / digital signature)

The problem with the first scheme is network disk must perform a hash and a signature verification on each block before writing. These are fairly slow operations to perform on lower performance disk processors, and both may not be necessary for many systems. This scheme replaces the signature verification at the drive by the following algorithm:

1. Initially (and perhaps at some regular interval) each user generates a random number (KEYUID.) This number is encrypted with the network disk's public key (multiple network disks would share the same public key), signed by the user's private key and sent to the network disk. A timestamp is appended before the signature to prevent a replay attack. Upon receipt of this key, each disk updates it's key object for that UID with this KEYUID, see



User (UID)
1. Pick KEY_{UID}
2. Encrypt to create KEY'_{UID}
3. Append timestamp
4. Sign the request
5. Send the request to network disks.

Network Disk
1. Verify signature with UID's public key.
2. Verify timestamp is newer than one for UID in certificate object.
3. Decrypt KEY'_{UID} to obtain KEY_{UID}.
4. Update entry in key object for that UID.

UID	$Certificate_{UID}$	KEY_{UID}	$Timestamp_{UID}$
UID	$Certificate_{UID}$	KEY_{UID}	$Timestamp_{UID}$
UID	$Certificate_{UID}$	KEY_{UID}	$Timestamp_{UID}$

Figure 5. Writer Authentication Key Initialization

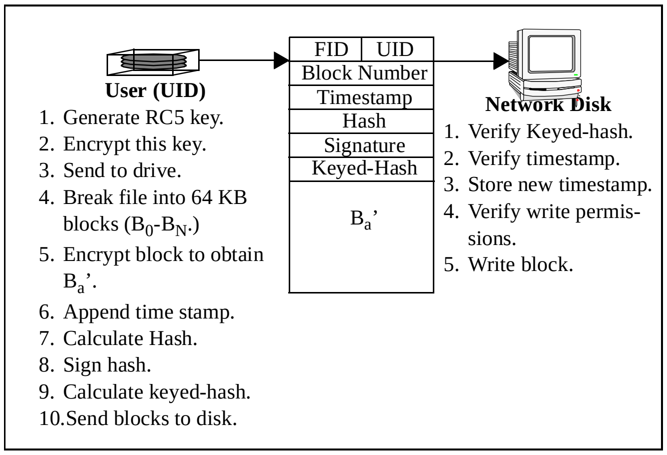
Figure 5.

2. The data block is prepared as in the first scheme, encrypting, hashing, and signing the data.
3. The KEY_{UID} is then combined with the hash of this block, and a new hash is computed (perhaps an HMAC.) The block from step 2 along with this new hash is sent to the drive. Since this only speeds up the writing at the disk, the potential weakness may not be justified for some systems. Which scheme to use depends on the level of assurance needed. The analysis of this method used just one hash function in the cryptographic overhead calculations. This is a reasonable estimation because the second hash is only over two 128 bit words - the original hash and the key, so the real performance will be slightly slower. If HMAC is used, there is an additional 2 word hash.
4. Upon receipt of this block, the drive can calculate the keyed-hash using the UID's key to authenticate the sender and verify the integrity of the block. If this passes, and the timestamp is newer than the block being replaced (for rewrite), the block is written See Figure 6.

The read operation is the same as scheme 1, illustrated in Figure 3. This scheme loses the property that no sensitive data is stored on the drives since a write-authentication key needs to be protected. There is still no sensitive data transmitted over the network. If this writer-authentication key is compromised, an adversary can only write blocks to the drive, but the reader of the blocks would be able to easily determine that they are bogus since the hash or signature would fail.

This scheme provides the following services:

- **Confidentiality** - Provided via RC5 encryption just as scheme 1.
- **Data Integrity** -For the reader of the data, integrity is provided via a signed cryptographic hash just as scheme 1. Integrity for writing the data is provided by having the drives recalculate the appended keyed-hash over the file. If this matches, that data integrity is ensured.



User (UID)
1. Generate RC5 key.
2. Encrypt this key.
3. Send to drive.
4. Break file into 64 KB blocks (B_0-B_N.)
5. Encrypt block to obtain B_a'.
6. Append time stamp.
7. Calculate Hash.
8. Sign hash.
9. Calculate keyed-hash.
10. Send blocks to disk.

Network Disk
1. Verify Keyed-hash.
2. Verify timestamp.
3. Store new timestamp.
4. Verify write permissions.
5. Write block.

Figure 6. Writing a File - Scheme 2

- **Non-repudiation** - Can be provided via a signature of the data hash just as scheme 1.
- **Authentication** - Reader authentication is provided by encrypting the RC5 key just as in scheme 1. Authentication for the write operation is provided by the keyed-hash.

As seen in Figure 7 the signature verification performed by the network disk for a write operation is removed at the cost of adding another hash operation on the host computer. This will improve system performance when the host computer is more powerful than the processor on the network disks. The read operation is the same as scheme 1.

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

Block Write

Operation	Host	NAS
En/Decrypt:	1	0
Hash:	1	0
Signature:	0	0
Verification:	1	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 7. Number of Cryptographic Operations - Scheme 2

3.3.3 Scheme 3: Good Security / Fast Performance (Full Keyed-hash)

The previous two schemes use a private-key signature to identify the originator of a data block. This scheme uses a keyed-hash approach to authenticate a writer of a data block. This has the disadvantage that the network disk contains sufficient information to forge a data block. The keyed-hash function has the property that the verifier of the keyed-hash can also create the keyed-hash since it is a symmetric function. If a network disk is compromised with this scheme, it is possible that the adversary could write information to the drive. This would require that they were able to obtain the writer-authentication keys from the drive. These keys could be stored encrypted for greater security. The system still prevents an adversary from accessing any encrypted data stored on the drive or any data in transit.

This scheme is the fastest presented thus far. It works by including a keyed-hash with each encrypted block and simply leaving the signature out. The drive is able to determine that the user or group 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 keyed-hash on the block.

Performing a keyed-hash 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 farther ensure data integrity.

Just as in the other schemes, the write operation starts with encrypting the block. Then a keyed-hash value is appended to the timestamped encrypted block. This block is sent to the network disk as shown in Figure 8. The read operation is more complex for the network disk. 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 9.

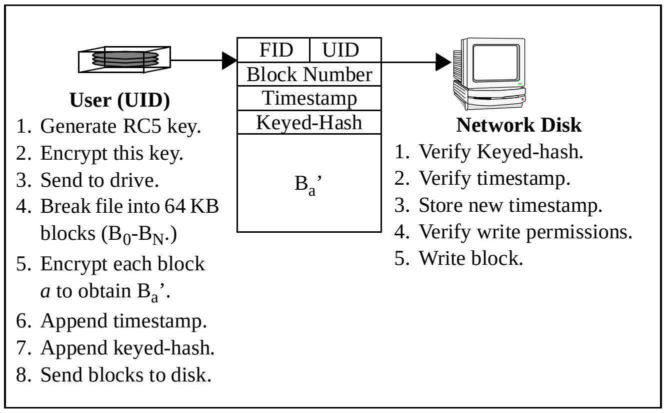


Figure 8. Writing a File - Scheme 3

This system provides the following cryptographic services:

- **Confidentiality** - Provided via RC5 encryption just as scheme 1.
- **Data Integrity** - Provided by the keyed-hash over the data. The key used for this hash can be exchanged with the drive using RSA.
- **Non-repudiation** - True non-repudiation can not be provided with this scheme.
- **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 signature of the hash.

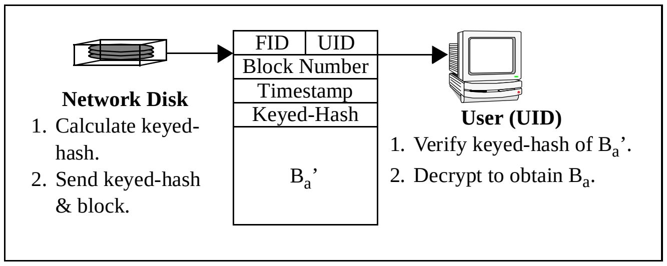


Figure 9. Reading a File - Scheme 3

The number of cryptographic functions performed is clearly reduced as shown in Figure 10. Most noteworthy is the removal of public-key operations for reading and writing a block. Note that public-key operations are still used upon creation of a file or file group, as the RC5 key used needs to be encrypted with the key for each user and group that needs file access

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 10. Number of Cryptographic Operations - Scheme 3

4 Experimental Results

The performance of some popular cryptographic algorithms on current host and drive hardware needed to be analyzed to determine the performance impact of implementing security described in the above schemes. For the cryptographic hash algorithm, MD4 and MD5 were tested. RC5 was tested for the secret-key cryptographic algorithm. The public-key cryptographic algorithm analyzed was RSA with a 512 bit modulus.

The hardware analyzed for possible host architectures starts with a Sun Sparc Ultra/60 with a 360 MHz U2 processor. This is the fastest available Sun workstation as of the time of this report. A Pentium-II 266 MHz Gateway Solo-9100 laptop and AMD K6-300 were the selected Intel-type platforms.

For the embedded computers, a Motorola MVME-147 with a 16 MHz 68030 was tested, as an outdated platform for a baseline. The new Motorola MVME-2600 with a 333 MHz PowerPC (604) was the high-end embedded system. Note that the AMD K6 used in the desktop is a viable embedded processor, and only costs $33 retail in May 1999.

The Sparc was running Solaris 2.6, and had the cryptographic programs compiled with gcc. As seen in the table below, this processor was quite fast, but only marginally faster than the $33 AMD chip computer.

The MVME boards were running the VxWorks real-time operating system. The cryptographic programs for these systems were compiled with the Wind River version of gcc. Both of the Intel-type platforms were running the Linux operation system. The cryptographic programs were compiled with gcc.

Overall, the PowerPC was the fastest of the lot. Only the Pentium-II was faster with MD5, which resulted from an optimized implementation. The MD5 speeds for the un-optimized program run on the other systems was twice as slow on the P-II.

The RSA Laboratories claims RSA private-key operations of 21.6 Kbits/second on a 90 MHz Pentium [12]. This would result in a 128 byte block being signed in 47.4 ms. A Pentium-II/266 should be at least 3 times as fast (although for this C implementation it is the same speed), which suggests a heavily optimized implementation of the RSA algorithms would result in a 3x speed up from the numbers shown below. The amount of time the cryptographic routines take for the read and write operation for the proposed security schemes can now be analyzed. The increase in time for RSA operations as key length increases can be seen in Figure 11.

Operation	MVME-147 16 MHz 68030 VxWorks	MVME-2600 333 MHz 604 VxWorks	Pentium II 266 MHz Linux	AMD K6 300 MHz Linux	Sun Sparc Ultra/60 - 360 Mhz Solaris
MD4	128 ms	1.6 ms	2.2 ms	2.0 ms	1.8 ms
MD5	196 ms	3.2 ms	2.0 ms	3.2 ms	3.2 ms
RC5	426 ms	4.9 ms	9.1 ms	8.0 ms	5.3 ms
Sign Verify	1878 ms 172 ms	25 ms 2.3 ms	40 ms 3.5ms	34 ms 3.1 ms	57 ms 5.3 ms

Table 1. Cryptographic Algorithm Performance

The overhead for the first scheme is calculated by simply multiplying the times each operation is performed by the time that operation took in the experiment. These results are shown in Table 2. The older 68030 is clearly inadequate for this type

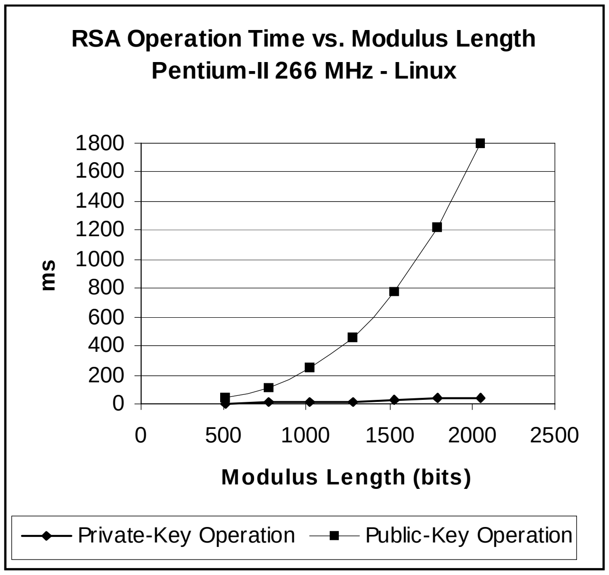


Figure 11. RSA Operation Time

of system, with writing a block taking 2.4 seconds on the host. The fastest hardware tested was the 333 MHz 604 PPC in the Motorola MVME-2600 VME board. If this PPC was used for the host computer the block write takes 31.5 ms. This would limit the system to 31.7 blocks writes per seconds. For these 64 KB blocks the throughput is limited to 2.0 MBps. This is the limit due to the cryptography, the actual limit is certainly lower. This may be acceptable for systems on slower (10 Mbit/second) networks, or systems where reading performance is much more important than writing performance. The time to read a block with the a cryptographic overhead of 8.8 ms (PPC) would allow a maximum throughput of 113.6 blocks per second. For the 64 KB blocks, this yields 7.2 MBps. This is approaching the maximum throughput of a current hard disk, so this speed may be acceptable.

Processor	Read - Host	Read - Disk	Write - Host	Write - Disk
68030	726 ms	0 ms	2432 ms	300 ms
Pentium II	14.8 ms	0 ms	51.3 ms	5.7 ms
AMD K6	13.1 ms	0 ms	44.0 ms	5.1 ms
Sun Ultra 2	12.4 ms	0 ms	64.1 ms	7.1 ms
PPC - 604	8.8 ms	0 ms	31.5 ms	3.9 ms

Table 2. Cryptographic Overhead for 64 KB block - Scheme 1

The goal of the second scheme was to off-load some of the processing overhead of the complex signature verification function from the network disk. This is particularly suitable for environments where the processors in the workstations are significantly faster than the processors in the disk drives. With this scheme, the cryptographic overhead required for the disk is reduced by one signature verification at the cost of greater complexity and an additional two-word hash. This reduces time the PowerPC takes to perform the cryptographic functions for a write to 1.6 ms at the network disk as shown in Table 3. There is still no cryptographic overhead for a read operation on the disk. The theoretical maximum throughput (cryptographic performance limited) is now 555.5 blocks per second for each disk. With the 64KB blocks, this equates to 35.5 MBps. A modern 100 Mbps LAN has an upper bound of 12.5 MBps. This scheme is certainly reasonable with respect to the network disks.

This scheme's bottleneck is likely with the host systems. Even with the PowerPC, a write operation has 31.5 ms of cryptographic processing overhead. This limits the system to 31.75 blocks per second. Again with the 64 KB block, this equates to only 2 MBps. This assumes that the processor is idle, and can devote 100% of its time to cryptographic functions. The read performance is still sufficient for most systems. Read speed is limited to 113.6 blocks per second (7.2 MBps.)

Processor	Read - Host	Read - Disk	Write - Host	Write - Disk
68030	726 ms	0 ms	2560 ms	128 ms
Pentium II	14.8 ms	0 ms	53.5 ms	2.2 ms
AMD K6	13.1 ms	0 ms	46.0 ms	2.0 ms
Sun Ultra 2	12.4 ms	0 ms	65.9 ms	1.8 ms
PPC - 604	8.8 ms	0 ms	31.5 ms	1.6 ms

Table 3. Cryptographic Overhead for 64 KB block - Scheme 2

The third scheme was designed for maximum performance while maintaining cryptographic protection of the data. With modern hardware such as the PowerPC, this scheme is quite fast. The slowest operation is reads or writes at the host computer. With the PowerPC, this took a mere 6.5 ms. See Table 4. This limits host read and write speed to 153.8 blocks per second. With the 64 KB blocks this equals 9.8 MBps. This is over 78 Mbps, around the speed of today's high speed networks. As networks get faster, so will computer hardware. The network disks are only limited to 625 blocks per second, or 40 MBps. This is significantly faster than any disk drive today, so should not pose a bottleneck.

As Figure 12 shows, processor speed is increasing at a rapid rate, allowing inexpensive processors to handle the cryptographic chores for individual disk drives. While today's processors are barely able to handle the task, future processors will be much better equipped to do so. Disk bandwidths are not increasing as rapidly as processor speeds — processor speed more than doubles every two years, while disk bandwidth takes longer than two years to double. Thus, an inexpensive processor on each disk drive will be able to handle the encryption chores necessary to protect the data on that drive.

Processor	Read - Host	Read - Disk	Write - Host	Write - Disk
68030	554 ms	128 ms	554 ms	128 ms
Pentium II	11.3 ms	2.2 ms	11.3 ms	2.2 ms
AMD K6	10.0 ms	2.0 ms	10.0 ms	2.0 ms
Sun Ultra 2	7.1 ms	1.8 ms	7.1 ms	1.8 ms
PPC - 604	6.5 ms	1.6 ms	6.5 ms	1.6 ms

Table 4. Cryptographic Overhead for 64 KB block - Scheme 3

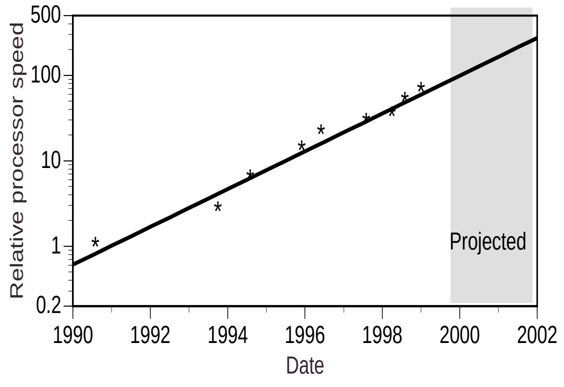


Figure 12. Processor speeds over time.

5 Conclusions

Past studies have shown that cryptographic controls are too costly for performance-critical and real-time systems. This study showed that modern processors have recently become fast enough to allow full cryptographic controls in systems that perform large network data transfers (64 KBytes or more.) This study also showed that the time for performing cryptographic processing is too great to use with small data transfers if these transfers are made hundreds of times a second. The demonstrated exponential growth of computer power showed the time to perform cryptographic routines is decreasing at an exponential rate. Current system designs that can not implement cryptographic controls due to performance issues will be able to include them in the near future. The amount of time to perform cryptographic controls clearly depends on the security scheme chosen. This paper showed that a keyed-hash authentication scheme coupled with RC5 encryption provides sufficient security for most systems and has a low overhead. This scheme is particularly suited for systems that need a fast write capability. For systems that can not trust the storage device to maintain data integrity, a signature based security scheme was presented. This scheme was significantly slower for the write operation, but allowed the storage device to hold no sensitive data.

The obvious next step is to build various embedded systems that utilize full cryptographic controls. The actual performance of these systems can then be measured and compared to systems with weak or no security.

6 Acknowledgments

We thank Steve Borbash for his cryptographic expertise.

References

[1] G. APOSTOLOPOULOS, V. PERIS AND D.SAHA, “Transport Layer Security: How much does it really cost?,” In Proc. IEEE INFOCOM, March 1999.

[2] M. BELLARE, R. CANETTI AND H.KRAWCZYK, “Keying Hash Functions for Message Authentication,” In Proc. Advances in Cryptology, pp 1-15, CRYPTO, 1996.

[3] W. FREEMAN, S. BORBASH AND D. MAUGHAN, “Simpler and More Secure Architectures for SNMPv3,” IETF Network Working Group, draft-freeman-snmpv3-sec-00.txt, November, 1998.

[4] H. DOBBERTIN, “Cryptanalysis of MD4,” Fast Software Encryption Workshop, Lecture Notes in Computer Science, vol. 1039, Springer Verlag, 1996, pp. 53-69.

[5] H. DOBBERTIN, “The status of MD5 After a Recent Attack,”, RSA Labs’ CryptoBytes, Vol. 2 No. 2, Summer 1996. http://www.rsa.com/rsalabs/pubs/cryptobytes.html

[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,” http://www.pdl.cs.cmu.edu/PDL-FTP/NASD/CMU-CS-97-118.pdf.

[7] NELSON E. HASTINGS, “TCP/IP Spoofing Fundamentals,” Proc. IEEE Fifteenth Annual International Phoenix Conference on Computer and Communications, 1996, pp 218-224.

[8] JIM REID, “Plugging the Holes on Host-based Authentication,” Computers and Security, 1996, pp 661-671.

[9] R. RIVEST, “The MD4 Message-Digest Algorithm,” IETF Network Working Group, RFC 1320, April 1992.

[10] R. RIVEST, “The MD5 message-digest algorithm,” IETF Network Working Group, RFC 1321, April 1992.

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

[12] RSA Laboratories, “How Fast is RSA,” RSA FAQ 4.0, Question 3.1.2, http://www.rsa.com/rsalabs/faq/html/3-1-2.html.

[13] B. SCHNEIER, “Applied Cryptography,” Wiley, New York, 1994.