# Integrity and Performance in Network Attached Storage

Howard Gobioff[1], David Nagle[2], Garth Gibson[1]

December 1998
CMU-CS-98-182

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

## Abstract

*Computer security is of growing importance in the increasingly networked computing environment.This work examines the issue of high-performance network security, specifically integrity, by focusing on integrating security into network storage system. Emphasizing the cost-constrained environment of storage, we examine how current software-based cryptography cannot support storage's Gigabit/sec transfer rates. To solve this problem, we introduce a novel message authentication code, based on stored message digests. This allows storage to deliver high-performance, a factor of five improvement in our prototype's integrity protected bandwidth, without hardware acceleration for common read operations. For receivers, where precomputation cannot be done, we outline an inline message authentication code that minimizes buffering requirements.*

1. School of Computer Science, can be reached via email at *{hgobioff,garth}@cs.cmu.edu*
2. Department of Electrical and Computer Engineering, can be reached via email at *bassoon@cs.cmu.edu*

This research is sponsored by DARPA/ITO through DARPA Order D306, and issued by Indian Head Division, NSWC under contract N00174-96-0002. Additional support was provided by the member companies of the Parallel Data Consortium, including: Hewelett-Packard Laboratories, Hitachi, IBM, Intel, Quantum, Seagate Technology, Siemens, Storage Technology, Wind Ricver Systems, 3Com Corporation, Compaq, Data General/Clariion, and LSI Logic.

**ACM Computing Reviews Keywords:** D.4.3 File systems management, C.3.0 Special-purpose and application-based systems, C.4 Design study, D.4.6 Cryptographic controls. D.4.4 Network communication

## 1 Introduction

The growth of network computing has made high-speed, secure network communication essential. The performance of secure systems, however, is lagging behind network speeds that are approaching 4 Gigabit/second full duplex [7]. Software-based security algorithms are often orders of magnitude slower than network speeds [16] and hardware implementations [8] can be costly, especially when deployed in low-cost computing devices.

One area where high-performance networking and security is becoming increasing important is data storage and retrieval systems, such as distributed file systems (e.g., AFS, NFS, NTFS), FTP, and the World Wide Web. Many modern distributed file systems, such as Transarc's DFS and Kerberized NFS, have the capability to provide secure communication but this option is often unused because it is considered too much of a performance penalty. Better support for high performance security could be achieved by incorporating hardware cryptography such as a secure co-processor [20, 21] or cryptographic accelerators [8]. However, servers have limited bus bandwidth and the extra transfers between memory and the security co-processor would further reduce the already precious bus bandwidth resource.

In order to deliver high-performance storage systems, we have investigated network attached storage, such as CMU's Network Attached Secure Disk (NASD) [9, 10]. Network attached storage proposes attaching storage devices directly to the network, removing the file-server bottleneck and providing scalable bandwidth from storage to client. Because the storage device is directly accessible on the network, the storage device (rather than the file server) is also responsible for security. To address this issue, NASD is designed to provide both integrity and privacy by incorporating security directly into the storage device.

To meet the bandwidth demands of current disk drives that burst at 1Gbit/second and stream at 250 Mbit/second, NASDs are designed to incorporate hardware-based cryptography. However, storage devices are extremely cost sensitive and must be carefully optimized to provide security without significantly raising the cost of the device. Therefore, their security system must be carefully designed to minimize implementation costs while maximizing performance [11].

This paper describes how we provide high-performance data transfer with integrity in a resource poor device. We present two new ideas: 1) an alternative message authentication code that enables precomputation of integrity information and 2) incremental verification of data integrity over large transfers that minimizes buffer requirements.

The next section provides background material on security and explains the network/storage performance issues that motivated this work. Section 3 describes the "Hash and MAC" solution and discusses the security implications of using a different structure for the MAC. Section 4 presents the performance of our prototype using “Hash and MAC” implementation. Section 5 discusses the importance of careful consideration of buffer requirements.

# 2 Background

## 2.1 Providing Integrity and Privacy

Informally, secure communications means “keep an adversary from observing or modifying communications”. Systems provide *integrity* if they protect message exchanges from being modified by an adversary. A system provides *privacy* if an observer of message traffic cannot learn the contents of the messages. Additionally, security behaviour often includes the *freshness* or timeliness of a messages arrival. In this paper, we focus on integrity because it is critical to the correctness of most applications and we briefly discuss the implications of privacy. Protecting the privacy or integrity of a message primarily requires a computation over all of the message’s bytes and varies little between applications using the same cryptographic functions. However, freshness guarantees are more application specific and depend on application managed state (e.g., timestamps, sequence numbers, challenge-response pairs).

Two approaches are used to provide *integrity*: 1) message digests with public key signature (aka “Hash and Sign”) and 2) message authentication codes. In both approaches, integrity is protected by binding the output back to a cryptographic key. A message digest (MD) is a strong cryptographic checksum that processes a variable length input string to produce a fixed-length output, called a hash or a digest, that is designed to detect malicious modifications (e.g., MD5 [17], SHA-1 [5], and Tiger [2]). However, since no secrets are used to generate the message digest, anyone can generate a valid message digest and it is insufficient, on its own, to protect integrity. To provide integrity, a message digest is normally signed with a private key of a public key cryptosystem so the receiver can verify who generated the digest. Using the corresponding public key, anyone can decrypt and verify the message digest matches accompanying message. Unfortunately, public key cryptographic operations are generally 10x-100x more expensive than message digest operations, making them ill-suited for performance-sensitive applications where only a small amount of data is processed.

The second approach, message authentication code (MAC), directly combines a variable length input string with a secret key to produce a fixed-length output and uses a shared secret key. The secret key ensures that only holders of the secret key can generate or verify the message authentication code. A good example of a modern MAC is HMAC-SHA1 [3] or XOR-MAC [4]. HMAC-SHA1 requires the key for linear processing over all the data bytes being authenticated. XOR-MAC allows parallel computation over data blocks, so it can exploit of hardware parallelism, but still requires the key be involved in all the processing.

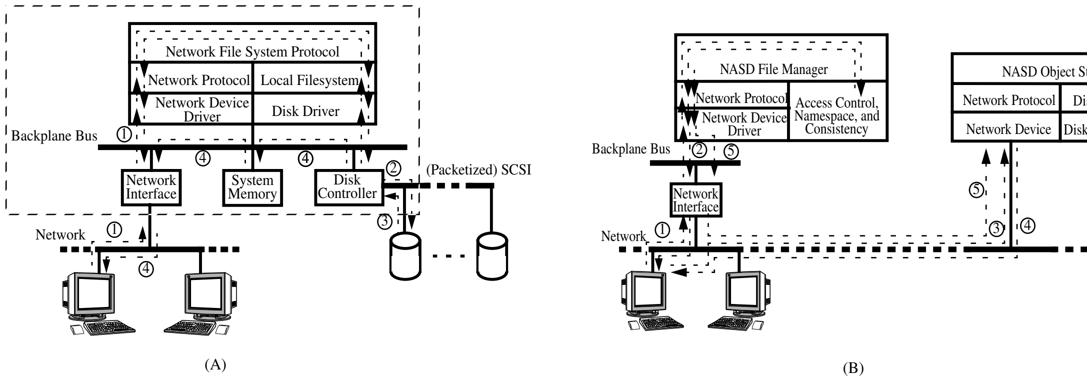


**Fig. 1:** SAD vs. NASD. Server Attached Disks (SAD) interposes a server between the storage and the network that copies data from the peripheral network onto the client network. In the SAD case, a client wanting data from storage sends a message to the file server (1), which sends a message to storage (2), which accesses the data and sends it back to the file server (3), which finally sends the requested data back to the client (4). In the NASD case, prior to reading a file, the client requests access to a file from the file manager (1), which delivers access credentials to the authorized client (2). So equipped, the client makes repeated accesses to the different regions of the file (3,4) without contacting the file manager again unless the filemanager choose to revokie the clients rights (5).The server also protects communication with the storage. NASD removes the fileserver from the data path. NASD clients infrequently consult a filemanager and, in the common case, go directly to the storage device thus avoiding the store-and-forward through the fileserver.

The other issue is *privacy* which is provided through encryption. Public key cryptography can be used to encrypt data but is extremely expensive. Symmetric key cryptography, such as 3DES, is significantly faster thus better suited to bulk encryption of data, but requires more complicated key management than public key cryptography. Frequently, a synergy of public key and symmetric key cryptography is used with key distribution done via public key and bulk data encryption done with symmetric key cryptography.

## 2.2 Network Attached Secure Disks

CMU's approach to network attached storage, Network Attached Secure Disks (NASD), promotes simple storage devices (e.g. disk drives) to first-class network entities that communicate directly with clients. In traditional distributed filesystems (Fig. 1a), storage is protected behind a fileserver that screens all requests and transfers data between an internal I/O bus and a general-purpose network. Often, the fileserver is the bottleneck because of the store and forward copying through the fileserver's memory. In previous work, we have shown that the NASD (Fig. 1b) architecture can reduce the fileserver workload by up to a factor of ten in a distributed filesystem [9]. We have also demonstrated a prototype system that removes the fileserver bottleneck and delivers almost linear scaling of bandwidth to clients [10]. As more and more high performance computing applications rely on distributed storage systems, NASD has the potential to deliver more scalable and cost-effective storage systems than conventional server-based solutions.

Unfortunately, even on an internal corporate network, the network is not safe for communication. When drives are promoted to first-class network citizens, they must become responsible for their own security [11]. This problem is distinct from previous work because it requires, both *security* and *performance* in a low-cost commodity *device*.

With current trends of decreasing feature size, we expect that future drives will have a 200MHZ StrongARM-class processor on-board to provide overall management of communication, security, and the media. Unfortunately, as we discussed in Section 2.2, this class of processor is unable to provide sufficient cryptographic performance necessary for a 100Mbps link without consuming most of the CPU and fails far short of a 1Gbps link.

NASDs retrieve data more often than the data is updated which creates an opportunity to reuse checksums or, in the case of security, message digests across multiple requests. With a normal MAC, the key is involved in the entire computation so the result can not be shared across multiple requests made with different keys. The “Hash and MAC” approach avoids this constraint by delaying the binding of the key into the computation. In the next sections, we show how this read-frequently write-rarely property along with the “Hash and MAC” can be used to deliver increased throughput from a prototype drive.

### 2.3 Security Performance

Cryptography’s computational requirements can have serious performance implications. Widely studied and standardized algorithms such as 3DES [6] for privacy and SHA-1 [5] for integrity are very expensive when done in software. For example, SHA-1 on a 90 MHz Pentium only delivers 54.9 Mbits/second while 3DES can only process 6.2 Mbits/second [16]. This is for hand-optimized implementations accessing cached data so the performance results represent peak performance; application performance will often be much lower. On Pentium machines, software implementations of SHA-1 are competitive with other hash functions while less standard encryption functions can deliver 2-5 times the performance of 3DES. Future cryptographic algorithms, such the proposals for the Advanced Encryption Standard [1], may provide performance a factor of 2-5 times faster than 3DES but are all slower than SHA-1. Table 1 shows our measurements of both SHA-1 and 3DES on a variety of processor. A 200 MHz StrongARM can only deliver 7.7 MB/sec using SHA-1 and a meager 819 KB/sec with 3DES.We expect these numbers to improve by a factor of 1.5 to 2 in hand-optimized StrongARM code but they will still fall short of a modern drive’s performance.

Hardware support can dramatically improve performance, especially in cryptographic algorithms (e.g., DES) that are specifically designed for high-speed hardware implementations. High-speed DES implementations (greater than 200 MB/second) have been available for many years [19]. Hardware support for

| Op & Machine/Operation Size | 8 bytes | 64 bytes | 256 bytes | 1024 bytes | 8192 bytes |
|---|---|---|---|---|---|
| SHA-1: Pentium II/300 | 2,029 | 9,692 | 17,547 | 21,969 | 23,709 |
| SHA-1: 500 MHZ Alpha | 2,133 | 9,501 | 15,883 | 19,310 | 20,615 |
| SHA-1: 200MHZ StrongARM | 800 | 3,759 | 6,218 | 7,430 | 7,877 |
| SHA-1: 133MHZ Alpha | 303 | 1,452 | 2,582 | 3,201 | 3,473 |
| 3DES: Pentium II/300 | 2,336 | 2,445 | 2,491 | 2,495 | 2,485 |
| 3DES: 500 MHZ Alpha | 2,183 | 2,341 | 2,356 | 2,352 | 2,359 |
| 3DES: 200 MHZ StrongARM | 787 | 814 | 817 | 819 | 819 |
| 3DES: 133 MHZ Alpha | 374 | 406 | 409 | 408 | 410 |

**Table 1:** Performance of cryptographic primitives in kilobytes per second. These values were measured using C code compiled with “gcc-O3” from the SSLeay 0.9.0b package of cryptographic algorithms [22]. All tests were performed by repeatedly preforming the operation over a single buffer for three seconds and the total number of iterations measured. Faster performance can be achieved via hand-coded optimizations. The Pentium versions included hand-coded assembly cores.

integrity algorithms has received less attention, in part due to encryption being a plausible replacement for a MAC along with a lack of strong demand. Encryption algorithms can protect integrity by relying on error propagation in the encryption system along with message redundancy or simply by taking the last cipher-text block. Relying on error propagation and message redundancy provides only limited protection against modification. Using cipher-text blocks as a MAC normally generates a small MAC, relative to the size of a dedicated message authentication code, and the encryption algorithms were designed for another purpose, making the exact relationship between a good cipher and a good MAC somewhat ill defined.

In some cases, the cost of extra hardware may be prohibitive or we may simply not be able to achieve the desired performance goals with available hardware. Rather than depending on hardware to provide all of the performance, this work exploits some application specific properties of storage to reduce the computational cost of protecting integrity. This allows software to deliver improved performance and for a drive to deliver greater performance from hardware support.

## 2.4 Prototype Performance

To illustrate the impact of security on performance, we implemented security in our NASD prototype. The NASD prototype is constructed from 3-generation old DEC Alpha workstations (133 MHz 21064) that are similar in performance to the class of processor we expect in NASDs [10]. Our client workstations are 233 MHz Alpha workstations that communicate with the NASDs through an OC-3 ATM DEC Gigaswitch. In our baseline prototype security implementation, we use HMAC-SHA1 for the message authentication code algorithm (and 3DES for privacy).

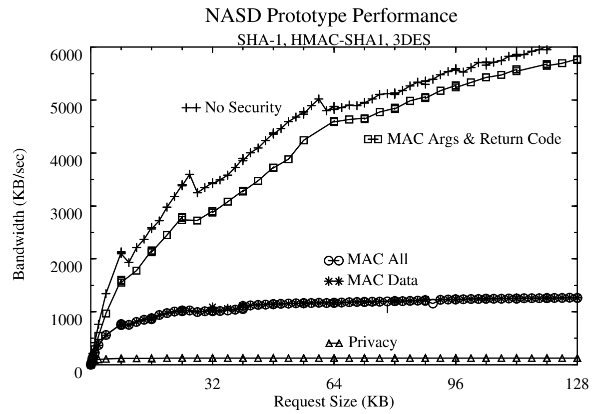


**Fig. 2:** All points average a minimum of 3 seconds of continuous requests and a minimum of 100 requests by a single client reading data from an in-memory object at the drive. The large bumps in the otherwise smooth curve are an artifact of the buffering behaviour of our RPC mechanism. Protecting the integrity of the arguments and return code impose a small fixed performance penalty while protecting the data exacts a much higher cost and saturates the drive CPU. Providing privacy quickly saturates the drive CPU because of the higher cost of 3DES compared to HMAC-SHA1.

Fig. 2 shows the NASD prototype's performance across a range of software-implemented security options. With no security, the prototype's maximum throughput, about 6 MB/second, saturates the CPU.

Adding the minimal meaningful security guarantee provides integrity of the request arguments and return codes, but no security on the actual data. This level of security permits some control over access to objects but does not prevent an adversary from tampering with the data payloads. With this level of protection, the amount of cryptographic work is a function of the request arguments and return code size and is independent the size of data transferred. Thus, we have a fixed cryptographic overhead that is amortized over the size of the data. For 1KByte reads, the fixed penalty reduces performance by 30% while for 128KByte reads performance is reduced by only 7%. However, integrity only on commands and return codes is a very weak notion of security and far less than is preferred in many systems.

Stronger security provides data integrity over arguments, return codes, and data, but has a fixed per-byte cost. For non-trivial request sizes, the per-byte cost of cryptography dominates the fixed cost of protecting the arguments and return codes. Fig. 2 shows that the prototype's maximum throughput is reduced by 46% for 1KByte reads and over 65% for 8KByte reads. At an 8KByte request size, the CPU saturates due to cryptography; larger transfer sizes do not provide any performance improvements.

Data privacy dramatically reduces performance to 126 KB/sec regardless of whether or not any integrity protection is used. Other algorithms may be faster in software but not by the factor of 100+ needed to make software implementations fast enough, so the problem can be expected to remain. One potential encryption technology, the Advanced Encryption Standard [1] will select a freely available and well studied encryption algorithm that is designed for both hardware and software implementations. It is unlikely,

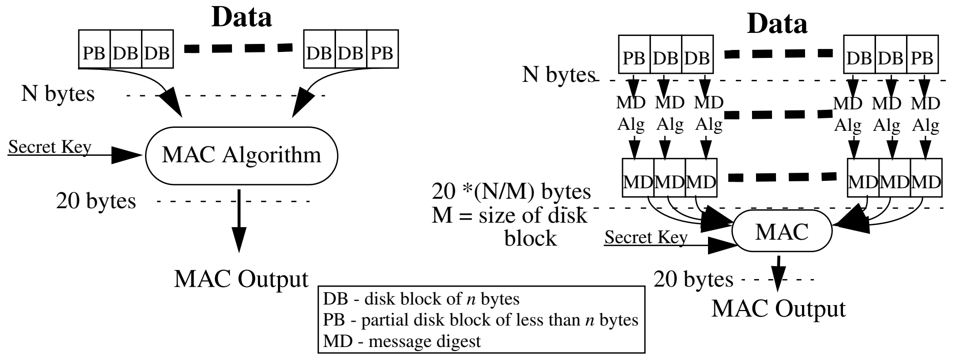


**Fig. 3:** On the left, most MAC algorithms involve the key in the computation over all the bytes of data and process the data linearly. On the right, "Hash and MAC" does not involve the key until late in the computation. This enables parallelization and precomputation for increased performance. The labeled dotted lines indicate the amount of data that passes in and out of the computation

however, to provide the factor of 100 performance improvement essential to high-throughput encrypted communications in low-cost devices.

# 3 Hash and MAC

For many applications, where reads are more frequent than writes (e.g. WWW, FTP, filesystems), the contents of the files do not change very often. Therefore, computation of information such as checksums is often performed repeatedly on the same data. To avoid this redundant computation, it is possible to *precompute* the TCP checksums across a set of data blocks and store each block's information with the original data [12]. When a read command is processed, both the data and checksums are read from disk (or cache RAM), thus avoiding the cost of on-the-fly computation.

While TCP checksums on data do not change from client to client, a MAC involves a key so the value can not be used for different clients using different keys. To exploit the performance gains of stored security information while providing good security, we propose a structure for generating MACs that *explicitly* delays the binding of the key to the computation. Based on existing message authentication code and message digest algorithms, our solution does the following:

- When a drive object is written for the first time, the drive precomputes a sequence of *unkeyed message digests* over each of an object's data blocks.
- For each read request, the drive generates a MAC of the concatenation of the unkeyed message digests corresponding to the requested data blocks.

Most MAC algorithms (Fig. 3a) involve the key in the entire process of generating a message authentication code. In contrast, we remove the key from the per-byte calculation and use it only in the final step (Fig. 3b). Because the key is not involved in the per-byte calculations, the results can be stored and used in response to multiple read requests to the same disk block, from different clients. Additionally, the keyless nature of the per-byte message digest computation is a simpler task for high-speed hardware because it doesn't require the identification of the appropriate key before it can begin processing.

The "Hash and MAC" approach is very similar to encrypting or signing a message digest. A public key signature would also provide non-repudiation of message origin which is not necessary in our network attached storage application. By entirely avoiding the use of encryption, the "Hash and MAC" approach can be used to provide integrity in the face of export restrictions on encryption algorithms.

### 3.1 Security of Hash and MAC

How does the "Hash and MAC" approach effect the security of the system? MACing the concatenation of hash values is very similar to signing them with a public key except it is much faster and does not provide the non-repudiability property associated with public key signatures.

If we assume the basic MAC function is secure, is the MAC of hash values secure? When something is considered "secure", it is normally secure for arbitrary inputs. If there was a class of inputs for which it was insecure then the MAC function as a whole would not be secure. An adversary breaks a MAC if they can recover the key or generate two inputs that have identical MAC values under an unknown key. Concretely, if the adversary breaks "Hash and MAC" by attacking the MAC function then they have defined a set of inputs, the concatenation of hash values, that can be used as an input to the MAC to break the original MAC. By our initial assumption, the MAC is secure, so this can not be true.

The compromise that "Hash and MAC" has made for performance is to allow an *off-line* attack against the message digest function. An adversary can apply arbitrary computational power to *precompute* two blocks that generate the same digest (i.e., a collision). Alternately, an adversary who observes requests and their associated message digests may attempt to find a second data block that generates the same digest (i.e., a second pre-image). As long as we use a large modern message digest such as SHA-1 or RIPEMD-160, which produce 160 bit values, this is a small risk. The best current attacks against both these message digests requires a brute force search of the input space. In order to find a second data block that collides with a given message digest, an adversary expects to compute digests of about $2^{160}$ inputs. A far simpler task, given large amounts of memory, is to find a pair of data blocks that generate the same hash by exploiting the birthday paradox which is still expected to require $2^{80}$ digest calculations [19].

Assuming an adversary finds a collision, they can only exploit the collision if one of the colliding blocks is within the system and the adversary can replace the in-system block with the colliding block. An adversary can potentially tabulate a large number of digests but the odds of an adversary holding a digest used in the system are:

$$2 \times NumOfCollisions \times NumOfDiskBlocksSeen / 2^{160}$$

which is small even when terabytes of data are transferred. Additionally, the adversary must be able to recognize that one of the collisions is useful quickly enough to explot it, which is difficult with a large number of on-hand collisions.

An adversary would have an easier task if they could insert one half of the collision into the system and then replace it with the other half of the collision. This would be a case of a malicious client attacking the system. However, in this case, the adversary could have already written the second of the two blocks rather than swapping the message blocks while they were being read. Thus, an adversary would primarily be able to exploit a collision in a system, such as a database system, where write operations are filtered through another host which decides which writes to pass on to storage. If a collision was found, then the adversary could swap a bad data block for an forwarded data block. Because a filtering host enforces some structure on requests that it passes on to storage, we limit the scope of useful collisions thus increasing security.

Because "Hash and MAC" generates multiple independent digests to create the final MAC, an adversary can parallelize an attempt to find a second pre-image. If the request is divided into $r$ different disk blocks, an adversary can attack $r$ different values when trying to find a second per-image. However, even for extremely large requests and heavily used storage devices, $r$ will not be large enough to substantially reduce the $2^{160}$ factor required to find a second pre-image. For example, if a terabyte of data were transferred and the digests were done on 8K blocks, $2^{22}$ attack message digests would be open to attack, only reducing the expected time to find a collision to $2^{138}$.

# 4 Integration of "Hash and MAC" with Network Attached Storage

## 4.1 Performance Benefit

To overcome the performance limitations of normal integrity solutions, we developed the "Hash and MAC" approach (Section 3). NASD's implementation of "Hash and MAC" uses SHA-1 to compute each disk block's message digest and HMAC-SHA1 for the overall message authentication code. We will refer to this specific instantiation of Hash and MAC as *HierMAC*. When data is read, the precomputed message digests are read from the drive and used as input to the HMAC-SHA1. If only a partial disk block is read, which can only occur in the first and last disk block of a request, a message digest of the partial disk blocks must be computed on the fly.

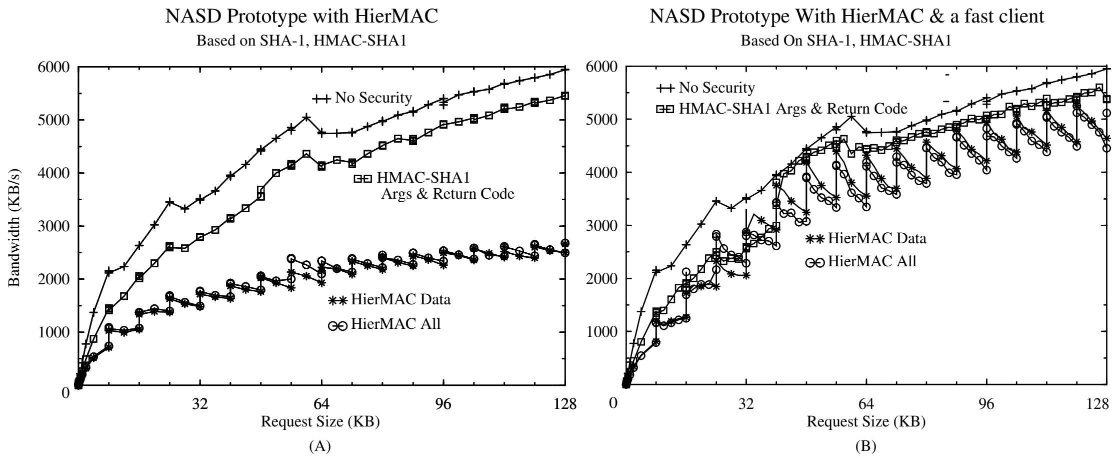


**Fig. 4:** "Hash and MAC" is used as the basic MAC construction rather than HMAC (Fig. 2) and digests are stored with data blocks. Otherwise, the experimental setup is identical to the experiment in Fig. 2. On the left, use of stored message digests has substantially improved the read bandwidth for large requests but fails to deliver the bandwidth we would expect because the client has now become the bottleneck. On the right, the clients have been modified to emulate the ability to perform cryptography at network data rates. This experiment shows the maximum amount of integrity-protected data that the drive can provide to a client. The data rates are much closer to the maximum available from the prototype without security.

With a normal MAC, the cryptographic costs were directly proportional to the number of bytes being transmitted. HierMAC reduces the costs to:

$$RequestHdr + (PrefixBytes + SuffixBytes) + NumOfFullDiskBlocks \times DigestSize$$

where *(PrefixBytes + SuffixBytes)* are the bytes in the partial data blocks. In our implementation, a disk block is 8KBytes while a message digest is 20 bytes. Thus, HierMAC is performing a MAC operation on 20 bytes per full disk block (8KBytes) transferred. This reduces (in the asymptotic case) the request time integrity processing to 0.2% of the number of bytes that a normal MAC would process. We are not changing the total number of bytes processed by the MAC algorithm. Instead, we are reordering the work in time and sharing work across multiple commands to reduce the on-the-fly cryptographic load.

Fig. 4a shows that reducing the on-the-fly cryptography load significantly increases throughput in the prototype. For large requests in multiples of 8KBytes, providing integrity across all of the data only decreases performance by 45% percent of our maximal performance. In fact, with stored digests, the drive can transmit secure data faster than the client is able to verify the data, shifting the bottleneck from the NASD drive to the receiving workstations.

It is important for disks to be cheap because they are high-volume commodity products and there may be hundreds or thousands in the organizational infrastructure. In contrast, client machines are likely to have high performance processors or dedicated hardware for special tasks, such as cryptography, that are critical to their regular functions. To understand the performance potential of HierMAC, we emulated one of these faster clients by disabling the verification of the MAC at the clients. Fig. 4b shows that HierMAC's low per-byte security overhead allows performance to closely follow the baseline NASD performance curve. Total cryptographic costs are now small enough that the additional cost of protecting the arguments in addition to the data is a noticeable performance difference rather than being masked by the overlapping of computation at both ends of the communication.

Fig. 4 shows that precomputing message digests on 8KByte blocks creates a pronounced saw-tooth behaviour. Between 8Kbyte block boundaries, performance declines because the drive must spend more time processing the prefix and suffix bytes. On 8KByte boundary, the drive only uses stored digests (*prefix* + *suffix* length returns to zero). For a uniform distribution of starting and ending bytes within a file, the average number of *prefix* + *suffix* bytes will be the size of one data block. Thus, the performance at the lowest points on the jagged edge, 1 byte before hitting a disk block boundary, will represent the "average performance" for a random workload. However, many filesystems will make requests that are aligned on disk block or VM page boundaries, resulting in significantly better performance.

Fig. 4b shows that by closely integrating security with storage and networking we have achieved a significant improvement in our read bandwidth. For many workloads, the majority of bytes are moved in large requests, making HierMAC a powerful optimization. However, HierMAC improves performance on neither writes nor small read operations.

### 4.2 Implementation

NASD uses an Inode structure similar to the Berkeley FFS [14]. Fig. 5 depicts how the stored digests are integrated with the direct pointers in the inodes. A significant advantage to storing the digests with the direct pointers is that the digests will always be available without additional I/O operations[1] whenever the data is accessed. However, the addition of a stored digest in each direct pointer reduces the total number of pointers that fit in a single inode, thus reducing the addressable storage at any given level of indirection and the overall addressable storage of a single NASD object. Within our NASD prototype, the addition of the stored SHA-1 digests reduced the number of bytes addressable via direct pointers from approximately 4 MB to 1.8 MB. Similarly, using our 8K inodes, the totally addressable storage in a single NASD object is reduced from over 4,000 PB down to approximately 8 PB. Because the Inode structure allows multiple

[1] For drives, the most expensive operation is to physically move the arm to another location on the media. In order to avoid this, we want to minimize the number of I/O operations required to service a client request. If retrieving stored digests introduces additional I/O operations or reduces the drive's ability to stream data off of the media then we will lose some, if not all, of the performance gain of stored message digests.

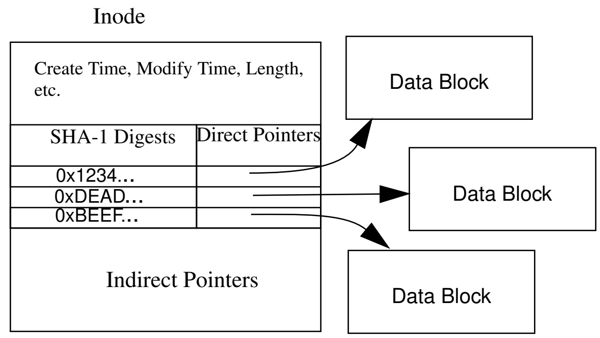


**Fig. 5:** In order for the drive to service an operation on an object, the drive must read the objects metadata into memory. We store the SHA-1 digests in the inode structure along with the direct pointers to the data so they are always available when we are accessing the data.

levels of indirection, a different implementation may select a different balance of direct and indirect pointers to trade off direct addressability against the total addressable storage.

Whenever a client writes new data, both the stored data and the stored digest must be updated. If they become mismatched, the drive will send clients erroneous MAC values and the clients will reject the data because they will believe that an adversary tampered with the data. This problem of keeping the stored digests consistent is very similar to the problem of avoiding errors with pointers to inodes and disk blocks, but there are some significant differences. Pointers are only modified when disk blocks are allocated (i.e., the file is extended) or deallocated (i.e., the file is shortened). In contrast, stored digests will be updated on every write operation, creating a bigger window for errors if the system crashes. Filesystems normally update the last modify time on every read but the last modify time is not critical to system integrity and does not exist in the indirect inode blocks.

A NASD drive can overcome this problem, maintaining stored digests and disk blocks consistent in the presence of drive failures through one of the following mechanisms.

- A log of dirty inodes stored in a small NVRAM. Before a disk block is written to the media, the associated inode is marked as dirty in NVRAM and the stored digest is updated. If the drive fails between receiving the write and having both the inode and data updated, the drive can recover by recomputing all the stored digests in dirty inodes. The recovery process will be more efficient if both the dirty inode and the potentially invalid digests are recorded, although the log will be slightly larger.
- The stored digest can be marked as dirty and its inode flushed to the media before any data is written out to the media. After the data is successfully written, the inode can be marked as clean and lazily written to disk. If the drive fails and a dirty digest is read, the drive will know that the digest may be inconsistent with the data and that it should recalculate the digest. This does not require any additional hardware support but it requires synchronous updates to metadata that could cause poor scheduling of the disk head.
- Optimistically, the drive could assume that the system will not fail and all the updates will eventually be flushed to the media. If a failure does occur, the client will detect the incorrect MAC when an out-of-date digest used during a client read. The client will request the drive to resend the data which, in this approach, will force recalculation of the digests. If the recalculated digest is the same as the stored digest, then the original reply to the client had been tampered with and a security violation should be

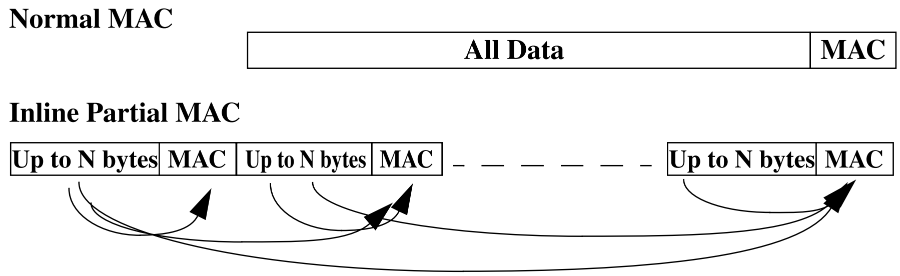


**Fig. 6:** A normal MAC protects “all” of the data. By injecting MACs of all data up to the current point into the transfer, the sender enables the receiver to buffer only N bytes before it can begin processing the data. All MACs are cumulative of all data up to the current point, preserving the relationship between chunks of data.

logged. Alternately, the drive either explicitly updates/verifies all the digests when crash-recovery occurs or flags all stored digests as invalid during crash recovery.

Of the three, NVRAM is the most powerful because it allows quick recovery without significantly hampering drive performance by requiring synchronous metadata update sequences at the price of extra hardware support. The optimistic case must either pay to recompute all the digests or involve the client in noticing a mismatched digest. If the client is involved, the client and the drive will pay the performance penalty at request time rather than the drive paying at restart.

## 5 Improving Receiver Buffering

For receivers (e.g. storage on writes, clients on reads), performing on-the-fly cryptographic work is unavoidable. Faster CPUs can improve the situation but networking already consumes up to 80% of the host’s cycles in protocol processing overhead, leaving few excess CPU cycles for cryptography [10]. Worse, to increase network transfer efficiency, applications often optimize for large transfers. With security added to the transfer, it is possible that a receiver would have to buffer an entire message before it could verify the data’s integrity. This can be a problem for storage servers or NASDs, where limited (and shared) resources make it difficult to dedicate large amounts of buffers. Further, delaying verification until all the data is received prohibits pipelining of data processing (e.g., overlapping writing to storage with the reception of the data).

In order to reduce the buffering requirement at the receiver and permit pipelining of the data transfers, we insert digests of prefixes of the requests into the data stream as illustrated in Fig. 6. At regular intervals, the sender inserts a MAC of all the data sent so far. Upon reception, the receiver can verify that *all data up to the MAC* is valid and then begin processing the data (e.g., writing to disk). This slightly changes the semantics of the MAC, from all or nothing to having prefixes be independently valid. Effectively, we are treating a large data transfer as a stream of data rather than a block of data. The primary

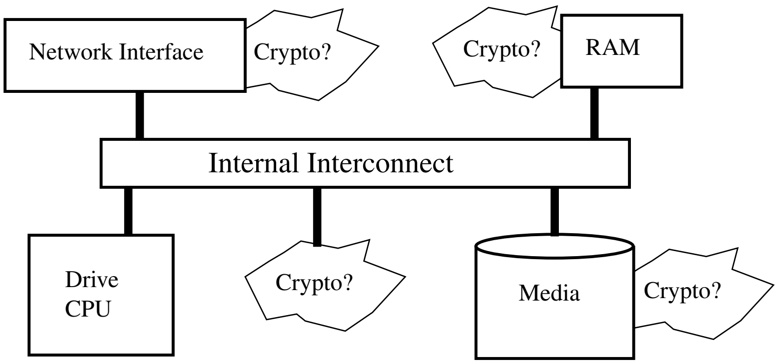


**Fig. 7:** Potential locations for hardware cryptographic support. A simplified view of a drive internal architecture illustrates the options that we explore in this section. Network attached storage devices will initially use an architecture very similar to a workstation to facilitate implementation.

advantage is to reduce the receiver's buffering requirement from a full request down to the amount of data sent between MACs. This is necessary for protecting the integrity of any large data transfers and is independent of the use of HierMAC. For NASDs, with their resource constrained environment, it is very important to process data as it arrives rather than buffering an entire write request because a single write may overflow all of the drive's available memory.

## 6 System Architecture: Placing Security Hardware

The software stored-digest approach substantially improves performance but cryptographic work can not be reused at the receiver nor cannot it provide privacy. Therefore, storage, other high throughput applications and a wide variety of clients using high throughput communication across the internet will benefit from special-purpose cryptographic hardware. This is similar to other bandwidth sensitive or timing sensitive problems, such as 3D graphics or audio, where application specific hardware support is often used to deliver the desired performance from low-cost devices.

Care must be taken when integrating hardware support into an overall system. In this section we examine how to best integrate special-purpose cryptographic hardware with storage, although some of the ideas are applicable to general purpose systems. Our goal is to understand the software, hardware and performance implications of alternate design decisions.

### 6.1 Placement of Cryptographic Hardware

At a high level, a network attached storage device can be pictured as shown in Fig. 7. The important components are the buffer controller/memory, the CPU (a 200 MHz StrongARM-class processor on NASD), the network interface, and the media. Each of these locations within a NASD are candidates for hardware support for cryptography. As we discussed earlier, using the CPU for the cryptography reduces system performance because many CPUs can not provide the cycles necessary to support high-speed data rates, so we avoid using the CPU for critical, data-intensive cryptographic computation.

### 6.1.1 At the Media

Cryptographic hardware could be placed close to media. The main motivation to do this would be to provide *location-independent* security where an adversary could physically steal a device yet not gain access to the data. We achieve this by placing a tamper-resistant device, such as a simplified version of secure coprocessor [20, 21], on the data path between the protocol processing and the media. Thus, data can be encrypted/decrypted as it goes to and from the media. The tamper-resistant device is necessary to prevent physical access, which would in turn provide access to the cryptographic keys thus access to data. In this situation, physical access and control of a drive does not provide access to stored data unless an adversary is able to violate the tamper-resistanct device and retrieve the cryptographic keys. However, placing cryptography near the media does not improve the drive's ability to communicate over the network. This location performs cryptography as the data is streaming off the platter which requires the keys be identified for privacy. Similarly, on a write request, the CPU must decide if a request is valid before data is sent to the media in order to maintain system integrity. Therefore, performing a message authentication code as the data goes to the media would be too late. Location independence is an interesting property that may be significant in high security problem domains but is not important in normal working environments.

### 6.1.2 On the Internal Interconnect

Another obvious place to put cryptographic support is directly on the internal interconnect. The primary drawback to placing the crypto on the interconnect is the additional bus traffic. Placing functionality at the most generally accessible point within the system requires it be reached over the shared interconnect. In a highly integrated special purpose device such as a NASD, the generality of the location provides little gain. Even within a workstation, the prevailing use of the cryptography is to protect communication which argues for placement of crypto near the network interface. The primary advantage of not integrating the cryptographic support with any of the other functional units in a system is *isolation.* By keeping it distinct, it is much simpler to associate secure state such as keys with the cryptographic support and delineate a clear security perimeter within a system.

If we place the cryptographic support on the interconnect, we can either have it actively managed by the CPU (i.e. a "dumb" crypto unit) or have it make decisions for itself about when to process data (i.e. a "smart" crypto unit). A dumb crypto unit would have the following sequence to process a request: the NIC receives an operation, the data is DMA'd into memory, the CPU decides what security processing is needed, the crypto unit is set up, data is DMA'd to the crypto unit, if encryption/decryption is used then the data is DMA'd back to memory, MACs are verified by the CPU, and finally data is DMA'd to the media. A smart crypto unit can snoop the bus and decide when to perform cryptography itself as data is

DMA'd directly into memory, thus eliminating the DMA operation to the crypto unit at the price of a significantly more complex crypto unit that understands significant portions of all the protocol processing. For a dumb unit, the storage device is moving the data either three or four times across the internal interconnect while the smarter unit can reduce this to two or three bus crossings. In either case, we also must also transfer MACs or message digests across the bus, increasing the interconnect traffic by 1-2% as long as MACs are generated on 1KByte or greater chunks of data. For small requests, the overhead of transferring the MAC or message digest values will be more substantial.

### 6.1.3 At the Buffer Controller

Within a disk drive, the buffer controller is on the data path between the interconnect and physical memory. All data handled by a drive will pass through the local buffer pool before the drive decides what is to be done with it. The semantics of the buffer manager can be roughly be approximated by those of a DMA engine. If we augment the buffer manager to optionally calculate a message digest on every read or write to memory and deposit the digest in another address, we can cleanly integrate integrity using Hier-MAC into the drive without introducing additional data movement. Similarly, we can augment the buffer manager to encrypt or decrypt under specified keys as data moves in or out of memory (although this requires that the proper key be identified). When data is moved to the network interface, the CPU will be able to provide the appropriate key for encryption. However, some part of the system needs to identify the proper key to use to decrypt data coming off of the network before the buffer manager can begin to decrypt data. The CPU could identify the proper key to use but this would require the CPU to take an interrupt on every packet reception. Alternately, the network interface could make the key decision if sufficient complexity was placed in the interface.

By placing the cryptography in the buffer controller, we are placing it closest to the data. However, the cryptographic operations must occur on the same block sizes with identical alignments to those at the sender so the CPU or the buffer controller must actively manage data movement to preserve the appropriate boundaries. Additionally, the buffer controller will run at the bus bandwidth rather than at slower network or media rates. This requires the cryptography at the buffer controller to run at bus rates in order to efficiently transfer data.

There is a strong parallel between the buffer controller with cryptographic functions and the IRAM work which moves computation to the memory [15]. An IRAM based approach could be used in both a NASD system and with future workstations to provide cryptography. IRAM also introduces parallelism among memory banks with independent processing that could be exploited for increased cryptographic performance if care was taken in how data is spread across the independent memory banks.

### 6.1.4 In the Network Interface

The final possible location to place cryptographic support is directly in the network interface. The primary use of cryptography is to secure communications between communicating parties, thus integration with the network interface is a logical target. Integration with the network interface also binds cryptographic and network performance together in a manner that allows them to be properly matched by network interface designers. However, network hardware manufactures are traditionally wary of adding any functionality to the interface because of both cost and complexity issues. If cryptography is to be integrated into a network interface, it is critical to reduce the security task of the interface hardware to a well simple well defined task.

We propose that we generate an unkeyed message digest of the payload of a request in the network interface for all incoming packets. This is similar to generating the IP checksum in hardware. This digest would cover at most an Maximum Transfer Unit (MTU) worth of data and perhaps much less. We do not require the network interface to involve any state across the packet for support of MACs, thus it has a straightforward task.

The per-MTU digests are then be placed in memory along with the packet for use later in a HierMAC computation. The network interface provides the bulk of the computation to protect integrity yet does not need to manage state across packets. This is similar to implementing IPSec at the network interface but does not require the network interface to manage connection state. By doing a HierMAC using the cooperation of the network interface, we are performing only 1-2% more cryptography than a simpler MAC would require for transfers of 8K or larger with an MTU of 1500 bytes or larger. Also, we are eliminating almost 1/3 of the bus traffic compared to sending the data to a cryptographic unit on the system bus. At the price of small amount of cryptography, we are freeing the network interface from the need to manage connection state such as buffering out-of-order packets and key lookup while still gaining the benefits of a simple integration of cryptographic functionality into the network interface.

## 6.2 Privacy

Privacy is more difficult to integrate into the network interface than integrity because we can not separate the calculation into an un-keyed and keyed phase as we have done with integrity. Privacy requires that the key be involved at all steps of the processing, otherwise the public steps could be skipped or undone by an adversary. Therefore, a simple function to be performed on all packets to help achieve privacy cannot be defined.

To provide high performance privacy at the network interface, we must perform key-lookup in the network interface. The latter will be necessary if integrating IPsec directly into hardware. While performing a lookup is certainly possible, it is potentially expensive in a low-cost device that may be concurrently interacting with a large number of parties.

## 7 Cryptographic Hardware

Over the last 10 years, a variety of vendors have provided products that include DES [19]. The DES core can readily be replicated to form a 3DES ASIC implementation. Implementations such as VLSI Technology's VM007 have demonstrated DES at 200 MB/sec data rates [19]. At the time this work began, no hardware SHA-1 implementations were available. Recently, HiFn 7711, targeted towards 100Mbps applications, has became available and can perform 3DES at 80 Mbps and SHA at 95 Mbps [8].

Recent research with FPGA implementations of cryptographic algorithms has demonstrated DES using 741 CLBs of a Xilinx 4028 @ 25.2 MHz performing at 402.7 Mb/sec [13]. In roughly 3 times the space, we could implement 3DES at equivalent data rates. Work at Carnegie Mellon University by Steve Schlosser has demonstrated an SHA-1 core in an FPGA using 719 CLBS, approximately 14,000 gates, on a Xilinx 4036XL @ 30 MHz performing at 125 Mbits/sec over 64 byte blocks and 142 Mb/sec over 8 kilobyte blocks [18]. The promising results of FPGA implementations demonstrate that, at reasonably low clock rates, these algorithms could achieve 100Mbps performance in an ASIC. Projecting out to an increased clock rate in a modern ASIC, we expect 1+ Gbps performance. Therefore, the cryptographic primitives are commercially available and there is no clear barrier to 1Gbps performance of the primitives in hardware.

FPGA shows promise of high performance in a package that can be reconfigured if an algorithm is broken. An ASIC implementation will fix an algorithm into silicon so a failure will require the abandonment of the ASIC. However, if an FPGA is used, the cryptographic primitive can be fixed or replaced on devices already in the field.

## 8 Conclusion

Computer security is of growing importance in today's increasingly networked environment. In order to efficiently provide integrity in a wide variety of systems, we need to consider many aspects of the problem in order to decide how best to provide security. By leveraging specific features of storage, such as writes being less frequent than reads and large transfers, we can significantly improve the amount of integrity-protected bandwidth a drive can provide without special hardware or a cutting edge processor. This helps to reduce the cost of a secure network attached storage system. Using a "Hash and MAC" approach, we deliver a factor of 5 improvement in integrity-protected bandwidth for large data transfers, closely following the no-security performance curve while sacrificing a small and well defined amount of security.

Modern ASICs can provide the high performance cryptographic primitives that can be integrated with network attached devices and commodity clients to provide high performance secure distributed systems. However, misapplication of the primitives can create additional buffering requirements (by using single message authentication code on large transfers), increased bus traffic (by moving data through external components), and increased amounts of integrity computation (by performing message authentication code on small data chunks). By integrating the security into the network interface using a hierarchical message authentication code, we avoid high-speed key lookup and state management when providing integrity with little extra cryptography or internal bus traffic.

## 9 Acknowledgements

We thank the entire Parallel Data Lab for their help and support in developing these ideas and making the research possible and enjoyable. We thank Bennet Yee for his helpful discussions on the security of Hier-MAC. Finally, we thank Joan Digney for helping to prepare this technical report.

This research is sponsored by DARPA/ITO through DARPA Order D306, and issued by Indian Head Division, NSWC under contract NO00174-96-0002. Additional support is provided by the member companies of the Parallel Data Consortium including: Hewlett-Packard Labs, Hitachi, IBM, Intel, Quantum, Seagate Technology, Siemens Microelectronics, Storage Technology Corporation, Wind River Systems, 3Com Corporation, Compaq, Data General/Clariion, and LSI Logic.

## 10 References

1 Advance Encryption Standard, http//www.nist.gov/aes

2 Anderson, R. and Biham, E. "Tiger: A Fast New Hash Function" *Proceedings of the Third Workshop on Fast Software Encryption,* 1996. Published as *Lecture Notes in Computer Science - 1039,* Springer-Verlag.

3 Bellare, M., Canetti, R., and Krawczyk, H., "Keying Hash Functions for Message Authentication", *Advances in Cryptology: Crypto '96 Proceedings*, Springer-Verlag, 1996.

4 Bellare, M., Guerin, R., and Rogaway, P., "XOR MACs: New methods for message authentication using finite pseudorandom functions". *Advances in Cryptology: Crypto '95 Proceedings*, Springer-Verlag, 1995.

5 Federal Information Processing Standards Publication 180-1, "Secure Hash Standard", April 1995.

6 Federal Information Processing Standards Publication 46-3 (draft), "Data Encryption Standard", January 15th, 1999.

7 Fibre Channel Association, http://www.fibrechannel.com

8 HiFn 7711 Data Sheet, http://www.hifn.com

9 Gibson, G., Nagle, D., Amiri, K., Chang, F., Feinberg, E., Gobioff, H., Lee, C., Ozceri, B., Riedel, E., Rochberg, D., Zelenka, J. "File Server Scaling with Network-Attached Secure Disks". *Proceedings of the SIGMETRICS 1997*. June, 1997.

10 Gibson, G., Nagle, D., Amiri, K., Butler, J., Chang, F., Gobioff, H., Hardin, C., Riedel, E., Rochberg, D., Zelenka, J. "A Cost-Effective, High-Bandwidth Storage Architecture", *Proceedings of ASPLOS VIII*, 1998.

11 Gobioff, H., Gibson, G., Tygar, J. D., "Security for Network Attached Storage Devices", Technical Report CMU-CS-97-185, 1997.

12 Kaashoek, M. F., Engler, D. R., Ganger, G. R., Wallach, D. A., "Server Operating Systems", *1996 SIGOPS European Workshop*. Connemara, Ireland, 1996.

13 Kaps, J., and Paar, C., "Fast DES Implementation for FPGAs and its Application to a Universal Key-Search Machine", 5th Annual Workshop on Selected Areas in Cryptography (SAC '98), Queen's University, Kingston, Ontario, Canada.

14 McKusick, M.K. et al., A Fast File System for UNIX, *ACM TOCS* 2, August 1984.

15 Patterson, D., Anderson, T., Cardwell, N, Fromm, R., Keeton, K. Kozyrakis, C., Thomas, R., Yelick, K. "A Case for Intelligent RAM: IRAM". *IEEE Micro*, April 1997.

16 Preneel, B., Rijmen, V., Bosselaers, A., "Principles and Performance of Cryptographic Algorithms", *Dr. Dobb's Journal*, December, 1998.

17 Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, Apr. 1992.

18 Schlosser, Steve, Personall Communication, 11/1998.

19 Schneier, Bruce. *Applied Cryptography.* John Wiley & Sons, Inc. 1998.

20 Smith, S., Weingart, S., "Building a High-Performance, Progammable Secure Coprocesor". IBM Research Report RC 21102, February 1998.

21 Yee, B. Tygar, J.D., "Secure coprocessors in electronic commerce applications". *Proceedings of the 1st USENIX Workshop on Electronic Commerce*. July 1995.

22 Young, E. SSLeay, http://www.psy.uq.oz.aus/~ftp/Crypto