Design, Implementation and Evaluation of Security in iSCSI-based Network Storage Systems

Shiva Chaitanya
Computer Science and Engineering
Pennsylvania State University
chaitany@cse.psu.edu

Kevin Butler
Computer Science and Engineering
Pennsylvania State University
butler@cse.psu.edu

Anand Sivasubramaniam
Computer Science and Engineering
Pennsylvania State University
anand@cse.psu.edu

Patrick McDaniel
Computer Science and Engineering
Pennsylvania State University
mcdaniel@cse.psu.edu

Murali Vilayannur
Mathematics and Computer Science Division
Argonne National Laboratory
vilayann@mcs.anl.gov

ABSTRACT

This paper studies the performance and security aspects of the iSCSI protocol in a network storage based system. Ethernet speeds have been improving rapidly and network throughput is no longer considered a bottleneck when compared to Fibre-channel based storage area networks. However, when security of the data traffic is taken into consideration, existing protocols like IPSec prove to be a major hindrance to the overall throughput. In this paper, we evaluate the performance of iSCSI when deployed over standard security protocols and suggest lazy crypto approaches to alleviate the processing needs at the server. The testbed consists of a cluster of Linux machines directly connected to the server through a Gigabit Ethernet network. Micro and application benchmarks like BTIO and dbench were used to analyze the performance and scalability of the different approaches. Our proposed lazy approaches improved throughput by as much as 46% for microbenchmarks and 30% for application benchmarks in comparison to the IPSec based approaches.

Categories and Subject Descriptors

D.4 [**Operating Systems**]: Storage Management; D.4.6 [**Security and Protection**]: Cryptographic controls

General Terms

Security, Design, Performance, Experimentation, Measurement

Keywords

iSCSI, IPSec, Encryption, Authentication

StorageSS'06, October 30, 2006, Alexandria, Virginia, USA.

1. INTRODUCTION

Data storage is an increasingly important asset in today's fast growing data-intensive network services. Given the overwhelming and mature Internet infrastructure, IP-based protocols for storage networking are beginning to emerge. The iSCSI protocol specifies how to access SCSI devices over a TCP network. The protocol encapsulates SCSI disk access requests (in the form of SCSI commands, data blocks and commands) into TCP packets and transmits them over IP networks. Although, implementing storage over traditional IP networks is economical and convenient, performance and security of the iSCSI protocol stack are equally important issues that need to be dealt with effectively. In this paper, we study the interplay of security and performance in the iSCSI protocol stack in a clustered network storage environment.

Ethernet speeds have been improving rapidly (multi-gigabit per second) so much so, that network throughput is no longer considered a bottleneck when compared to Fibre-channel based storage area networks. However, when security of the data traffic is taken into consideration, existing secure protocols that operate at the lower layers of the stack like IPSec prove to be detrimental to the overall throughput. On a typical IPSec implementation, we have measured a point-to-point throughput reduction of as much as 75%. Further, this throughput reduction is even more significant when multiple clients are active, resulting in poor scalability in a shared environment. The primary reason for this drastic reduction in throughput is the overhead of IPSec processing of all client connections and data at the central server. In this paper, we characterize the performance of the iSCSI protocol stack when used in conjunction with IPSec, demonstrate the critical path overheads of protocol processing and present the design and experimental results using techniques that reduce protocol overheads while still preserving security guarantees.

iSCSI-based storage systems are vulnerable to security attacks from various sources. Since they rely on TCP/IP based networks for data transmission, all the associated problems with these kinds of networks are inherited by the iSCSI protocol. iSCSI data that is sent in clear across the public networks is exposed to passive and active attacks by an

adversary snooping on the wire. Passive attacks involve gathering of information by eavesdropping on the physical medium and using the obtained knowledge to cause damage at a later point in time. This type of scenario is possible in situations where mission critical data of an organization traverses public networks. Leakage of such information, by itself, constitutes a major security breach. Active attacks involve more real-time intrusion by either modifying the packet contents, replacing them by new packets, denial-of-service attacks, etc. It also includes man-in-the-middle type where the adversary deceives the sender and receiver into believing that they are communicating with each other, but in reality, all messages are intercepted and modified appropriately for either end of the link.

Attacks in collusion with the iSCSI server are also possible where the adversary can simply obtain access to the critical data stored on the target disks. These kinds of attacks by an untrusted server can be prevented only by storing data encrypted on disk and relying on some trusted third party to do the management and secure distribution of encryption keys. The trusted entity provides the services for the clients and controls access to the server based on capabilities issued *apriori*. This scheme again raises the issue of placing the trust on a central authority that can be compromised in future. Analysis of this kind of problem usually results in trade-offs between developing a secure central solution and a distributed solution with key management among peer machines. Distributed solutions need to be based on a robust algorithm that remains foolproof against compromise of one or few number of clients. A large number of clients colluding and launching attacks is tough to protect against and is in general not considered a viable problem.

In this work, we only address security issues relating to the data traffic and make an assumption that the users of the storage network and servers are trusted parties. Some of the popular techniques like LUN Masking and Zoning that are employed in Fibre-Channel SANs can also be applied in iSCSI networks to guard against unfair usage of storage even by trusted clients. LUN Masking is a mechanism that allows nodes on a SAN to only see the LUNs that they are authorized to access. It works by mapping the LUNs to WWNs (World Wide Name used in FC networks, analogous to IP address) and only providing access to certain LUNs from certain WWNs. Zoning is another method where restrictions are placed on the communication between certain sets of WWNs. The access control is implemented either in the hardware driver on the initiators or incorporated on the Fibre-Channel switches. In our work, we assume that the initiators are trusted to perform fair accesses and hence this issue is not considered further. Also, in the normal scenario the initiators of a iSCSI network are application servers running in an organization and are typically trusted not to perform such illegal accesses.

The rest of this paper is organized as follows. The next section describes the IPSec protocol in greater detail, characterizes the performance overheads of the stack and motivates the need for lazy crypto techniques to ameliorate the performance drop. Section 3 outlines the design of the lazy decryption and authentications schemes. Implementation issues and performance evaluation with micro-benchmarks and real applications are presented in Section 4. We discuss related work in Section 5 before concluding the paper in Section 6.

2. ISCSI AND IPSEC

The iSCSI protocol definition provisions that the end points of communication be authenticated before starting an iSCSI session. An initiator must first establish a session with the target before obtaining access to storage data. A session is composed of one or more TCP connections. When the leading TCP connection of a session is setup, the login phase begins with exchange of authentication information between the involved parties through mechanisms such as the Challenge Handshake Authentication Protocol (CHAP) and the Secure Remote Password (SRP) protocol. After the session's operational parameters are negotiated, the full data transmission phase begins.

While the login process provides mutual authentication of the endpoints before the initiation of a session, iSCSI does not account for per-packet authentication and integrity of data during the full featured phase. In addition, privacy of the confidential data sent across the vulnerable IP networks is also not taken into consideration. One of the standard solutions is to run iSCSI in conjunction with some security scheme available in a lower layer of the protocol stack such as IPSec.

IPSec, an extension to IP, is a security protocol that can be established between two machines in order to secure data traffic. It provides confidentiality, authentication and integrity services to upper layers of the network stack. IPSec is used to secure tunnels against false data origins and encrypts traffic to combat active and passive intruders who can listen or modify the traffic. It consists of two separate protocols, IP Authentication Header (IPSec AH) and IP Encapsulating Security Payload (IPSec ESP).

IPSec AH provides connectionless integrity and data origin authentication, while IPSec ESP provides both those features and confidentiality. The Authentication Data field is computed by performing a one-way hash function (e.g. MD5, SHA-1) on the payload that includes all the data bits from the upper layers like TCP/UDP and network applications. The common hash function is used again at the receiver to compute the authentication data from the payload and is checked against the one provided in the AH header. Any modification to an original IP packet or a newly created one from an unauthorized host can be detected at the receiver. Figure 1 (a) shows the IP AH header format.

IPSec ESP provides confidentiality in addition to connectionless integrity and data origin authentication. The payload data field is computed by applying a symmetric encryption algorithm (e.g. Data Encryption Standard DES, Advanced Encryption Standard AES) to the original data comprising of TCP or UDP data. The encrypted data is followed by the authentication data computed in the same manner as IPSec AH. Figure 1 (b) depicts the format of the IPSec ESP header.

There are two ways by which the choice of cryptographic algorithms and the symmetric keys used by IPSec can be set on the two communicating machines. Manual configuration of the algorithms and setting the pre-shared keys is done by including the information in a system configuration file. Another option is to use the IKE (Internet Key Exchange) protocol. IKE works by allowing IPSec-capable devices to exchange security associations (SAs) to populate their security association databases (SADs). A Security Association (SA) is a set of security information that describes a particular kind of secure connection between two machines.

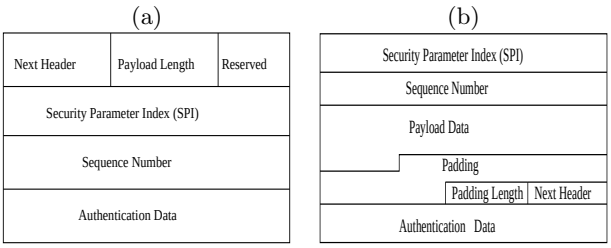

Figure 1: (a) IPSec AH Format, (b) IPSec ESP Format

It contains information like IP addresses, security protocol used, cryptographic algorithm, keys, etc. IKE also provides the benefits of dynamic SA establishment and dynamic rekeying.

2.1 IPSec performance

Ensuring high performance of the IPSec protocol is crucial to obtaining good throughput from the IPSec-based iSCSI stack. With the increase in available network throughput, thanks to the deployment of multi-gigabit Ethernet, the focus has shifted from the bandwidth offered by the physical medium to the network stack (CPU processing overheads) on either side of the link. Authors in [8] identify the time spent in TCP/IP processing and interrupt overheads as the two major bottlenecks in obtaining high iSCSI performance. With the inclusion of IPSec processing to the network flow, there is an additional slowdown to the overall throughput. Further, processes of encryption/decryption are highly CPU-intensive and affect other applications running on the system. One of the major factors hampering the performance of IPSec is the very fact that it is implemented at the IP layer (lower layer of the stack). Thus, application-level information and requirements cannot be exploited by the IPSec layer.

Let's take a closer look at the transmission of a single IP packet through the IPSec layer on the sender side. The following description is specific to the native IPSec implementation in Linux kernel 2.6. Though the exact processing times and overheads may vary slightly among different implementations of IPSec, all of them follow a similar set of packet processing rules. IPSec processes a packet based on IPSec Security Policy (SP) and IPSec Security Association (SA). SP indicates if a packet should be processed as an IPSec packet or dropped. The chosen packets are processed by the IPSec stack with some parameters included by SA. IPSec Security Policy is checked next to determine if the packet is required to be processed by IPSec. Then, for every IPSec packet, the SA database is queried to obtain the destination chain of output functions. The output functions perform cryptographic manipulations on the packet based on the type of the protocol used, AH or ESP. Note that there can be more than a single output function per packet if multiple security protocols are configured for a particular route. Finally, the IPSec packet is passed to the destination chain.

We conducted a small set of experiments to study the performance (throughput) of IPSec AH and ESP for the `ttcp` program for a single client accessing a server. The sender processes continuously transmit data of a fixed size to the receiver which is listening on a TCP socket. The average

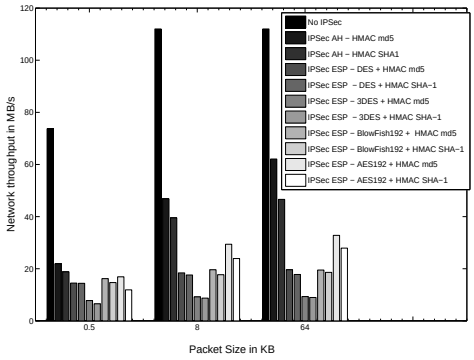

Figure 2: Maximum Throughput of IPSec for different packet sizes

throughput and latency for each packet size was computed. This experiment was repeated for a setup without IPSec (No-IPSec), IPSec with AH (IPSec-AH) and IPSec with ESP (IPSec-ESP). Measurements with IPSec AH and IPSec ESP were repeated for different algorithms supported in the Linux kernel 2.6 IPSec implementation. The test machines, each a 2-processor Intel(R) Xeon(TM) 3.06GHz (with hyper-threading turned on) running a 2.6.13 Fedora 4 kernel were directly connected via a Gigabit Ethernet Switch. The network interface cards on these machines were Intel 82546EB Gigabit Ethernet adapters. Figure 2 shows the maximum network throughput obtained by running `ttcp` on top of IPv4, IPSec (AH and ESP). Different algorithms were chosen for the two IPSec protocols. The maximum throughput attainable was determined by increasing the number of concurrent threads in the program till the host CPU utilization reaches 100% or network becomes saturated. For IPv4, network bandwidth was fully utilized for $\geq$ 8KB packet sizes. For a packet size of 512 bytes, the maximum throughput obtained was $\simeq$ 75 MB/s. For IPSec AH, the maximum throughput values obtained for packet sizes of 512 bytes, 8 KB and 64 KB was 22, 47, 62 MB/s respectively. Compared to IPv4, that amounts to a throughput reduction of 70%, 58% and 45% respectively. For IPSec ESP, the throughput reductions were more drastic with values of 77%, 74% and 71%. The point to point throughput suffers as well when multiple clients are active thus resulting in poor scalability in a shared environment. The primary reason for this degradation is the centralized bottleneck caused by the IPSec processing of all the client data at the server. This type of performance degradation of IPSec when running iSCSI can have a detrimental affect on the overall throughput of the I/O applications. In the critical path of IPSec processing, there are a pair of crypto processes (encryption/decryption and/or HMAC) that run on two machines. The next few sections describe how these overheads can be ameliorated while still preserving the security guarantees.

3. PROPOSED SOLUTIONS

Consider a typical scenario in an iSCSI environment, that comprises of a set of iSCSI clients called the initiators and an iSCSI target that is attached to the storage disks. These disks are exported at the block-level to the clients so that they can store huge quantities of data and retrieve them

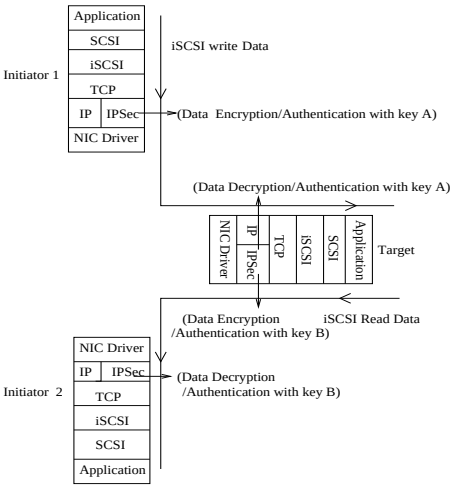

Figure 3: Path of data packets during a Write by Initiator 1 and a subsequent Read by Initiator 2. Keys A and B are IPSec symmetric keys of Initiator1-Target and Initiator2-Target respectively

when necessary. The storage management system at the iSCSI target takes care of all requirements like availability, maintenance, backup, disk space management etc. When an iSCSI initiator writes a data block to the target, it is stored on the disk as it is received. If IPSec was used to secure the network between the initiator and target, the data sent from the initiator is transformed using the symmetric IPSec keys and back again at the target before it reaches the iSCSI layer. This data when requested by the same client or some other client is manipulated again at the IPSec layer on the server.

Figure 3 depicts this flow of read-write requests for IPSec ESP and it is apparent that the crypto processing of the storage data blocks that take place at the server is superfluous and could be done away with if the same key is used for a data block throughout the entire path. In other words, since the server simply acts as a storage reservoir and doesn't use the data itself, it need not be involved in the data processing if the transformation keys are securely communicated to the clients. Note that the crypto processing of the control data (iSCSI headers) cannot be avoided at the server and needs to be performed in the critical path. Correct functioning of the iSCSI protocol requires the control data be immediately acted upon at the server. Since IPSec is implemented at the IP layer it is not possible to selectively act on certain portions of the IP payload (TCP header, iSCSI header) and leave the iSCSI data portion untouched. IPSec is designed to provide application independent security services and is oblivious of upper level characteristics and requirements. Consequently, crypto operations are performed in an identical manner on ALL traffic between two IP nodes. Thus our observation to improving performance is to move these operations to the iSCSI layer by which it is possible to reduce unnecessary processing from the critical path. We now explain the first of our two mechanisms namely *Lazy Decryption*.

3.1 Lazy Decryption

iSCSI defines its own packets that are referred to as iSCSI Protocol Data Units (PDUs). A iSCSI PDU consists of a header and possible data, where the data length is specified within the iSCSI PDU header. The basic segment header size is fixed and is 48 bytes. Additional headers can be attached to the basic segment and this information is specified in the `TotalAHSLength` field (see Figure 4). As reasoned above, when used in conjunction with IPSec ESP, the decryption of the iSCSI data is not really needed at the target for writes. Similarly during reads, data is already in encrypted form at the server and can be sent across the network without any transformations. iSCSI PDU header contains important control information like the iSCSI session id, SCSI command and other operational parameters that need to be protected from external attacks as much as the critical data. Delayed decryption of the header at the server does not work because the iSCSI target layer needs to process all the fields in it for proper functioning of the protocol.

In our lazy decryption approach, we aim to provide the confidentiality services to iSCSI data. IPSec AH can be used for providing iSCSI data origin authentication and its integrity. In addition, the IPSec Security Association Database will be populated with a IPSec ESP encryption algorithm and a symmetric key. The key is either pre-shared or IKE is used to dynamically establish and update it. However, the Security Policy on the two machines is not configured to use IPSec ESP. This allows for symmetric encryption key management between the two machines without actually using it for encrypting data traffic at the IPSec layer. This symmetric encryption key is used to encrypt the iSCSI PDU header at the iSCSI layer. The symmetric encryption algorithm used by the iSCSI initiator and target can either be hard coded in the implementation or negotiated as one of the operational parameters during the login phase. In the current work, we assume that they are set manually in the iSCSI configuration files. The novelty of our approach lies in the fact that we exploit the secret key of IPSec and use it to communicate between the machines at the application layer. The following subsections explain in detail how lazy decryption is incorporated at the iSCSI layers on the initiator and target software.

3.1.1 *At the Initiator*

For iSCSI operations that require only the transmission of the iSCSI PDU header (e.g SCSI READ Command), implementation of lazy decryption is straightforward. The symmetric encryption key obtained from IPSec is used to first encrypt the basic header (48 bytes). The set of all additional headers is encrypted as a single unit using the same key. This is followed by the transmission of the entire header. For operations that require subsequent transmission of data blocks following the PDU header (e.g. SCSI WRITE), things are a little more complicated. We cannot use the symmetric encryption key to encrypt the data blocks as well because decryption of the shared data at some other client would require knowledge of the secret key known only to the original initiator and the target. We overcome this by generating a random `Block Encryption Key (BEK)` for every data block about to be written to the server. These BEKs are used to encrypt the storage blocks in the iSCSI PDU data and are stored sequentially in a new Additional

Header Segment (AHS) appended to the end of iSCSI PDU header. Note that the BEKs are not one-time keys and persist only till some initiator generates a new key during the next write to the same block. Therefore, the BEKs need not be remembered by the client and can be disposed of. Figure 4 shows the modified iSCSI PDU header format used for this scheme. The confidentiality of the iSCSI PDU header (including the BEKs) is protected by the symmetric IPSec encryption key and that of the iSCSI PDU data is protected by the BEKs. This is an application of skip encryption [13] used in security schemes. When the initiator receives iSCSI PDUs from the target, it first decrypts the 48 byte header using the symmetric key and checks to see if there are additional headers and data segment. Additional headers are decrypted in the same manner using the symmetric key. Data blocks, if present are decrypted using the BEKs sent along in the last AHS.

3.1.2 *At the Target*

The implementation at the iSCSI target layer is quite similar to that of the initiator but with two important differences. The Block Encryption Keys received are not used to decrypt the data blocks. Rather, the encrypted data are stored directly on the iSCSI exported disks. Separate storage space, different from the iSCSI disks, is needed to accumulate the BEKs. The keys can either be located at fixed positions on a disk or stored in a file system. In the current implementation, we chose to use file storage for the keys and made sure that the ones belonging to adjacent logical block addresses are co-located in a file. This improves the access time of the keys during iSCSI data transfers. We also implemented a simple cache to store frequently accessed BEKs in memory.

Now we describe our second mechanism called *Lazy Authentication* that reduces the CPU load at the iSCSI server by delegating data integrity checks to the clients.

3.2 Lazy Authentication

In this approach, we intend to achieve origin authentication and integrity of data blocks by performing the checks only at the clients. IPSec key management service is used to share a symmetric key between a client and the server. Hashed message authenticated code (HMAC) of the iSCSI header is generated at the sender and checked immediately at the receiver to detect any modifications to control data. However, the server does not perform any HMACs on the data blocks. The following subsections describe the method in detail as implemented on the initiator and target ends.

3.2.1 *At the Initiator*

For operations involving transmission of only the iSCSI header, the initiator generates a HMAC for the entire header (including the Additional header segment) and appends it at the end. Typically the size of the generated output is about 16 bytes (HMAC-MD5) or 20 bytes (HMAC-SHA1). The symmetric HMAC key used is the one obtained from the IPSec layer. If data blocks are transmitted along with the header, then a random `Block Authentication Key (BAK)` is generated for every block and is used to generate the block HMAC. The BAKs are transmitted in encrypted form (using the symmetric IPSec key) and constitute a part of the iSCSI header. Figure 5 shows the modified iSCSI header to support lazy authentication. The block HMACs are appended

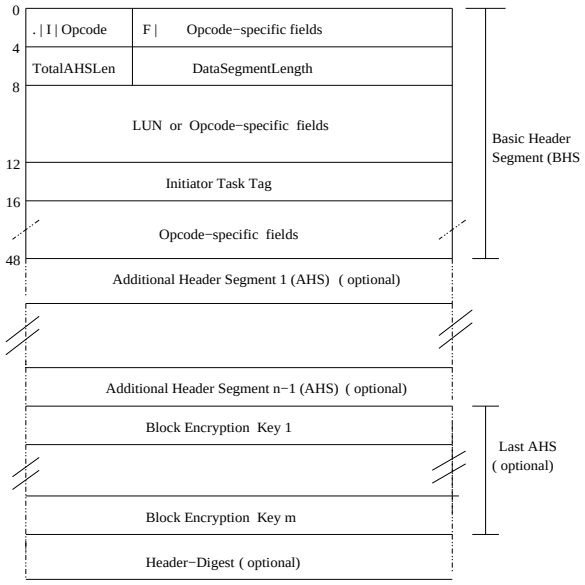

Figure 4: Modified Format of iSCSI PDU header for Lazy Decryption. iSCSI PDU header is encrypted using the IPSec symmetric encryption key and the Data Segment is protected using the BEKs.

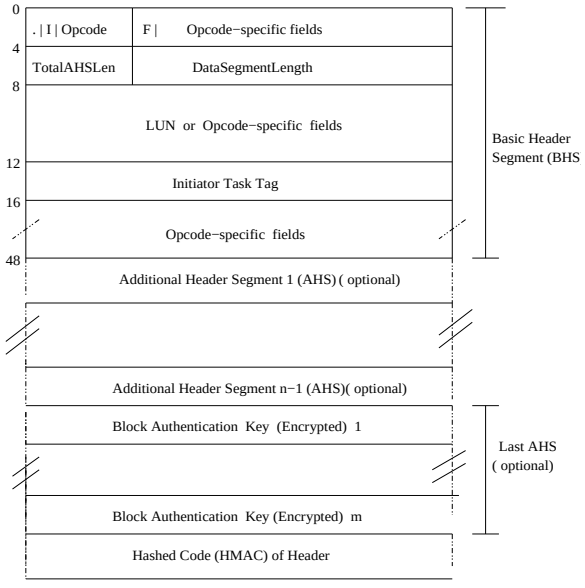

Figure 5: Modified Format of iSCSI PDU header for Lazy Authentication. The Block Authentication keys are encrypted using the secret IPSec key and sent in the AHS. They are followed by the HMAC of the header.

Table 1: Crypto transformations of iSCSI PDUs. (H) denotes Header of iSCSI PDU, (B) denotes Body/data segment of the iSCSI PDU, (Ck) is the secret communication key (symmetric IPSec key) and (Bk) is the random body key (BAK or BEK). E and D denote encryption and decryption respectively.

Scheme	iSCSI PDU on network	ClientWrite	ClientRead	ServerWrite	ServerRead
Plain iSCSI	$M = H + B$				
IPSec AH	$M = (H + B), HMAC_{Ck}(H + B)$	$HMAC_{Ck}(M)$	$HMAC_{Ck}(M)$	$HMAC_{Ck}(M)$	$HMAC_{Ck}(M)$
IPSec ESP	$eM = E_{Ck}(H + B)$ $M = eM, HMAC_{Ck}(eM)$	$E_{Ck}(M)$, $HMAC_{Ck}(M)$	$HMAC_{Ck}(M)$, $D_{Ck}(M)$	$E_{Ck}(M)$, $HMAC_{Ck}(M)$	$HMAC_{Ck}(M)$, $D_{Ck}(M)$
Lazy Auth.	$M = (H + B), HMAC_{Ck}(H)$, $HMAC_{Bk}(B)$	$HMAC_{Ck}(H)$, $HMAC_{Bk}(B)$	$HMAC_{Ck}(H)$, $HMAC_{Bk}(B)$	$HMAC_{Ck}(H)$	$HMAC_{Ck}(H)$
IPSec AH + Lazy Dec.	$eH = E_{CK}(H), eB = E_{Bk}(B)$ $M = (eH + eB), HMAC_{Ck}(eH + eB)$	$HMAC_{Ck}(M)$, $E_{Bk}(B)$	$HMAC_{Ck}(M)$, $D_{Bk}(eB)$	$HMAC_{Ck}(M)$	$HMAC_{Ck}(M)$
Lazy Auth + Lazy Dec.	$eH = E_{CK}(H), eB = E_{Bk}(B)$ $M = (eH + eB), HMAC_{Ck}(eH)$, $HMAC_{Bk}(eB)$	$HMAC_{Ck}(eH)$, $HMAC_{Bk}(eB)$ $E_{Bk}(B)$	$HMAC_{Ck}(eH)$, $HMAC_{Bk}(eB)$ $D_{Bk}(eB)$	$HMAC_{Ck}(eH)$	$HMAC_{Ck}(eH)$

to the end of the iSCSI PDU data segment. Since the block HMACs are created using the encrypted BAKs present in the header, they need not be associated with the header content. Attacks where the data blocks and their HMACs are replaced on the wire with previously seen ones can be detected since different BAKs are used for different blocks and the adversary does not know them. When receiving a header, the client computes its HMAC and checks to see if it matches with the one provided in the header itself. This ensures that the header originated from the iSCSI server and was unmodified in transit. If data blocks are also received along with the header, the BAKs are first decrypted and used to compute the block HMACs which are then checked against the ones received with the iSCSI PDU.

3.2.2 *At the Target*

The iSCSI header received is immediately authenticated (origin and integrity) using the shared IPSec HMAC key. Data blocks if received, are not checked for their authenticity at the server. The BAKs are however decrypted and stored on a separate file system in the same manner as described in the lazy encryption scheme. Since the authenticity of the data blocks is not checked until a subsequent read by a client, they cannot overwrite existing data on the disk. Therefore, there is a need for creating versions of unauthenticated data blocks till a client confirms the successful completion of a read. We implement block versioning by storing a sequence of unauthenticated blocks on separate storage. When a client performs a read, it receives the latest version from the server and if the received blocks cannot be authenticated, the read is re-tried with the previous version and so on. A variant of this technique was proposed in [14, 20] in the context of a fault tolerant Byzantine distributed storage system. The successful completion of a read by a client enables the server to commit the correct version on to the iSCSI disk and the previous and incorrect versions are destroyed.

The lazy approaches help in decreasing the CPU load at the server by reducing the crypto processing of the data blocks at the server. Table 1 shows the overall picture of the crypto manipulations of the iSCSI header and data at the client and server for reads and writes. The first column enumerates different schemes that differ in the manner in which the iSCSI PDUs are handled from the crypto perspective. The second column shows the format of the messages on the network. The last four columns show the actual inbound and outbound crypto processing on the clients and the server. Note that the crypto processing at the server end is significantly reduced in the lazy approaches.

4. IMPLEMENTATION ISSUES

For implementing the crypto operations at the iSCSI layer, we used the cryptographic API available in 2.6 Linux kernels. The scatter list based API takes page vectors as arguments and works directly on pages. The API supports three types of transforms, ciphers (Encryption), Digests (Hashing) and Compression. Since iSCSI software drivers are implemented as kernel modules, the API (kernel methods) was integrated directly without any overhead of library function calls. The XFRM interface was used to retrieve the symmetric encryption key from the IPSec layer's SAD.

If the basic unit of the crypto operations were a single block, reads and writes simply involve generating HMACs and encrypted blocks and transmitting them directly to the server. However, this requires a stream of kernel function calls for every data block. To reduce this overhead, a fixed size of consecutive blocks can be chosen as a block set and crypto operations can be performed at the granularity of a block set. However, block sets introduce complications when a subset of blocks are accessed. Suppose a set of consecutive blocks A, B, C and D form a single encryption unit. Every time a subset of these blocks (say B,C) is written by an iSCSI initiator, the remaining blocks in the set (A,D) must first be retrieved from the target and decrypted at the client, following which all the blocks in the set must be re-encrypted with the new key and sent back to the server. In the current work, we use a block as the granularity of crypto operations to avoid this problem.

One of the tunable parameters in the above implementation of lazy approaches is the mapping between Block Keys (BEK and BAK) and data blocks. A unique Block Key can be associated with a single block or multiple blocks. If the mapping were one-to-one, then the total number of keys needed for a iSCSI disk is the capacity of the disk in blocks. For a disk capacity of 36GB and key size of 16 bytes, the storage space required for Block Keys alone at the iSCSI server would be (36/512)*16 $\simeq$ 1.1 GB, assuming that the typical size of a block is 512 bytes. One-to-many mappings decrease the amount of storage needed for the keys but increase the security vulnerabilities of the data blocks.

In the lazy authentication approach described in the Section 3.2, versions need to be maintained for unauthenticated data blocks. The number of versions for each block increases with every consecutive write. This can cause an explosion

of storage requirements at the server for some applications. To avoid this, periodic garbage collection can be performed at the server that simply authenticates some versions of a block and commits the latest valid block.

With increasing Ethernet speeds, CPU overheads of protocol processing is a cause for concern in servers. By removing the encryption and decryption processing needs at the iSCSI server as shown in the lazy decryption and authentication schemes, a significant bottleneck has been removed. This not only increases the throughput of a single initiator to target, but also makes the system much more scalable with increase in number of simultaneous initiators. Of course, there is a new overhead of storage and retrieval of Block Keys but we estimate that this is much less compared to the CPU-intensive crypto operations and corroborated by our results. As mentioned before, one of our major focus in the current work is the performance of iSCSI protocol when deployed in a shared storage environment with potentially multiple initiators accessing the same blocks simultaneously from a trusted server. With this setup, key storage and management at the server provides an efficient mechanism to perform secure data block transfer.

4.1 Evaluation Space

In a TCP/IP network, the underlying physical medium can vary considerably. Some networks already support physical security while others may not have any such mechanisms. Also, the security requirements of applications running on top on iSCSI can differ significantly as well. We classify the security needed by iSCSI depending on the existing underlying infrastructure and requirements of applications into three levels. The first level is No-Security. Some iSCSI systems are deployed in controlled environments where the nodes operate under physically secure conditions and no additional security guarantees need to be provided. This is also the case in situations where the underlying network has already been provided with security channels like VPN. We name this level as `Security-0`.

The second level of security offers origin authentication and integrity of iSCSI data. This is applicable in situations where active attacks are possible by snooping on the public networks. Confidentiality of data may be unimportant in web servers where the content is readable by anybody but the users require the guarantee that the data did arrive from the server and that it was not modified on the way. Also, the server needs to assured of the identity of the user and integrity of user data received. Sometimes, even in a private network where the privacy of data is preserved, the end users at the periphery of the network still need to be authenticated. The security level, named `Security-1` addresses all these issues by providing authentication and data integrity. One might argue that the IPSec AH and Lazy Authentication do not exactly provide the same security gaurantees because the TCP and IP headers are not covered in the latter. However the spoof attacks potentially caused by this in Lazy Authentication can be detected easily because the two communicating parties share a IPSec symmetric key and use it to authenticate the iSCSI header. The provision against anti-replay attacks by IPSec AH which uses a monotonically increasing sequence number for the purpose is also present in lazy authentication with the initiator task tag in the iSCSI PDU header taking up the role of the nonce.

The third level provides the services of `Security-1` along with data confidentiality. This level of guarantee is needed in situations where the end hosts are communicating in a completely insecure underlying network. TCP/IP running on public networks comes under this category. Sensitive iSCSI data traversing such paths are susceptible to all types of security attacks discussed in this work. By providing origin authentication, data integrity and encryption, this level `Security-2` guards against the vulnerabilities exposed to iSCSI data traffic.

Table 2: Three levels of iSCSI security and their implementation methods

Name	Services Offered	Implementation Choices
`Security-0`	No security	S0 - No changes to iSCSI
`Security-1`	Data origin authentication and Integrity	S1-1 - IPSec AH (Soft) S1-2 - Lazy Auth (Soft)
`Security-2`	Data origin authentication, Integrity and Confidentiality	S2-1 - IPSec ESP (Soft) S2-2 - IPSec ESP (Hard) S2-3 - Lazy Decryption (Soft) +IPSec AH (Soft) S2-4 - Lazy Decryption (Hard) +IPSec AH (Soft) S2-5 - Lazy Decryption (Soft) +Lazy Auth (Soft) S2-6 - Lazy Decryption (Hard) +Lazy Auth(Soft)

Table 2 summarizes the three security levels. The third column in the table describes the different implementation choices that we have at our disposal to satisfy the requirements of the three iSCSI security levels. The labels SN-M indicates the design choice with N representing the security level and M being the index within the group.

For `Security-0`, we use the software implementations of iSCSI initiator and target in their original form. Both of them are configured to run on a traditional TCP/IP stack with no added functionality. `Security-1` is provided by running iSCSI on top of IPSec AH or using Lazy Authentication in software. To avoid design space explosion, we restricted ourselves to using *hmac-md5* uniformly across all implementations of the one-way cryptographic hash function and *aes192-cbc* as the encryption algorithm. Six different implementations were chosen for `Security-2`. The first two are iSCSI running over IPSec ESP implemented in software or the encryption/decryption offloaded to hardware. The details of the crypto hardware used in our work are presented in the next section. The next two involve the combination of Lazy Decryption and IPSec AH with the encryption/decryption implemented in software or hardware. Similarly the last two are using the combination of Lazy Decryption and Lazy Authentication.

4.2 Experimental Testbed

Our iSCSI testbed consists of x86 based SMP machines. Each client machine is equipped with dual Hyper-threaded Intel Xeon processors clocked at 3.06 GHz equipped with 1 GB RAM. All the nodes are connected directly via a Gigabit Ethernet network with Intel 82546EB Ethernet adapters. The iSCSI server is equipped with dual AMD Opteron processors clocked at 1.6 GHz with 1 GB RAM. The server is connected to a nStor Fibre-Channel storage array containing 12 Seagate 36GB drives of 15K RPM speed. The external transfer rate of each disk is 200 MB/s, Average la-

tency is 2 msec and Average Read/Write seek time is 3.5/4.0 msec. The operating system running on the client machines is Fedora Core 4 with a kernel version 2.6.13-1.1532, while the server uses Fedora Core 2 with kernel version 2.6.10-1.771. Open source iSCSI initiator `open-iscsi-0.4-408` [2] and Ardis target implementation `iscsitarget-0.4.11` [1] were used for the software versions of iSCSI initiator and target. In order to run our applications, we use Redhat's open source clustering SAN file system (GFS version 6.1) which allows applications to simultaneously read and write to a shared file system on the SAN.

For hardware acceleration of crypto functions, we used the `PowerCrypt 5x` PCI card. It supports encryption algorithms such as AES, DES, 3DES and RC4. Based on Hifn's 7956 encryption processor, PowerCrypt 5x can provide AES cipher speeds of approximately 80 MB/s. The card is supported on Linux systems through OCF (Open Cryptographic Framework) drivers. The drivers support hardware acceleration of general purpose crypto functions. We instrumented the 2.6 kernel to incorporate OCF asynchronous offloading of ESP processing for the native IPSec stack.

4.3 Benchmarks Used

To evaluate the performance of the iSCSI system with security overheads, we used a set of micro-benchmarks and applications. *Rawio* is a workload generator that is used to extensively evaluate I/O subsystems. The application benchmark *BTIO* is a scientific workload that is used in the study of MPI-IO performance. We also used *dbench* which simulates netbench, an industry standard for testing file servers.

- **Rawio:** Rawio is a I/O benchmarking tool that is used to measure the raw block read and write performance of disks. The parameters that can be varied include access pattern, number of outstanding requests and the block size. We choose the combinations of the following parameters: Random/Sequential and block size = 512 bytes, 8K , 64K and Reads/Writes. For each of the combination, we test the performance of the nine design choices by running Rawio concurrently from 1,2,4 and 8 clients. The maximum number of outstanding I/O requests was set to 16 in each of the clients. The measured metric is the average point to point throughput as seen by a client in MB/sec.

- **BTIO:** The BTIO benchmark (Version 2.4) from NASA Ames Research Center [5] simulates the I/O required by a time-stepping flow solver that periodically writes its solution matrix. The solution matrix is distributed among processes by using a multi-partition distribution in which each process is responsible for several disjoint sub-blocks of points (cells) of the grid. The access pattern in BTIO is non-contiguous in memory and on file and uses MPI derived data-types to describe this non-contiguity. We ran the Class A problem size for our experiments on 1,4 and 9 clients. It uses a 64x64x64 element array to generate a file of size 400 MB.

- **dbench:** dbench is an application that simulates netbench which is an industry standard for testing file servers. Since the requirements of running netbench are 60 to 150 Windows PCs on switched fast Ethernet, dbench is often chosen as an alternative to generate the file-system load typically seen from a netbench client. The file-system load is created with the same IO calls that the smbd server in Samba would invoke in a netbunch run. dbench does not exercise any networking calls.

4.4 Results

Figure 6 shows the performance of Rawio for different sequential and random workloads on varying block sizes. First, we consider the throughput values obtained for sequential reads (the leftmost bar stack in the bar charts of Fig 6 (a)) for a single client case with no hardware crypto acceleration. When the block size is 512 bytes it was observed that the maximum throughput of a single running client without any security is around 5.6 MB/sec. It is limited by the network I/O calls and iSCSI processing overheads of small packets. When IPSec AH (the bar labeled S1-1) and IPSec ESP (S2-1) are running in conjunction with iSCSI, the throughput drops down by 13.4% and 24.8% respectively. When the block size is increased, the iSCSI layer and network I/O processing are no longer the bottleneck and the maximum throughput in the No-Security case continues to increase peaking at the network bandwidth at 64K bytes. However, IPSec processing becomes a bottleneck at larger block sizes due to the CPU overhead of crypto processing with throughput reductions of 52.5% and 71% for S1-1 and S2-1.

The bars also show that the lazy approaches and their combination with IPSec outperform the pure IPSec performance. Lazy authentication (S1-2) achieves better throughput in the security level S1 with significant improvements over IPSec AH at larger block sizes (46.2% for 64K bytes). The combination of Lazy decryption and IPSec AH (S2-3) also performs well compared to the pure IPSec ESP case although the improvements are not as large as before (17.3% for 64K bytes) The full combination of Lazy decryption and Lazy authentication (S2-5) performs even better with improvements of 11.6% and 40.4% for the small and large block sizes. The reduction in crypto processing at the server end results in better throughput for the lazy approaches.

When more than one client are set to request sequential blocks simultaneously, Figures 6 (a) and (b) show the average point to point throughput seen by a client is reduced by a factor only slightly less than the number of clients. This shows that the disk transfer bandwidth at the server is not much of a overhead for sequential workloads and that the maximum throughput obtained when a single client is running scales well when more clients are added. Also note that the lazy approaches are much more scalable than IPSec at larger block sizes with the increasing clients. For sequential reads with eight clients running (the rightmost bar stack in the bar charts of Figure 6(a)), Lazy authentication achieves a 92.4% improvement over IPSec AH for a block size of 64K bytes. The corresponding values for the combinations of Lazy decryption/IPSec AH and Lazy decryption/Lazy authentication over IPSec ESP are 44.4% and 170%. The tremendous increase in the relative throughput values of the lazy approaches compared to IPSec is due to the fact that there is very little crypto processing overhead at the server. All the data packets are authenticated or encrypted/decrypted at the client side resulting in a very scalable design.

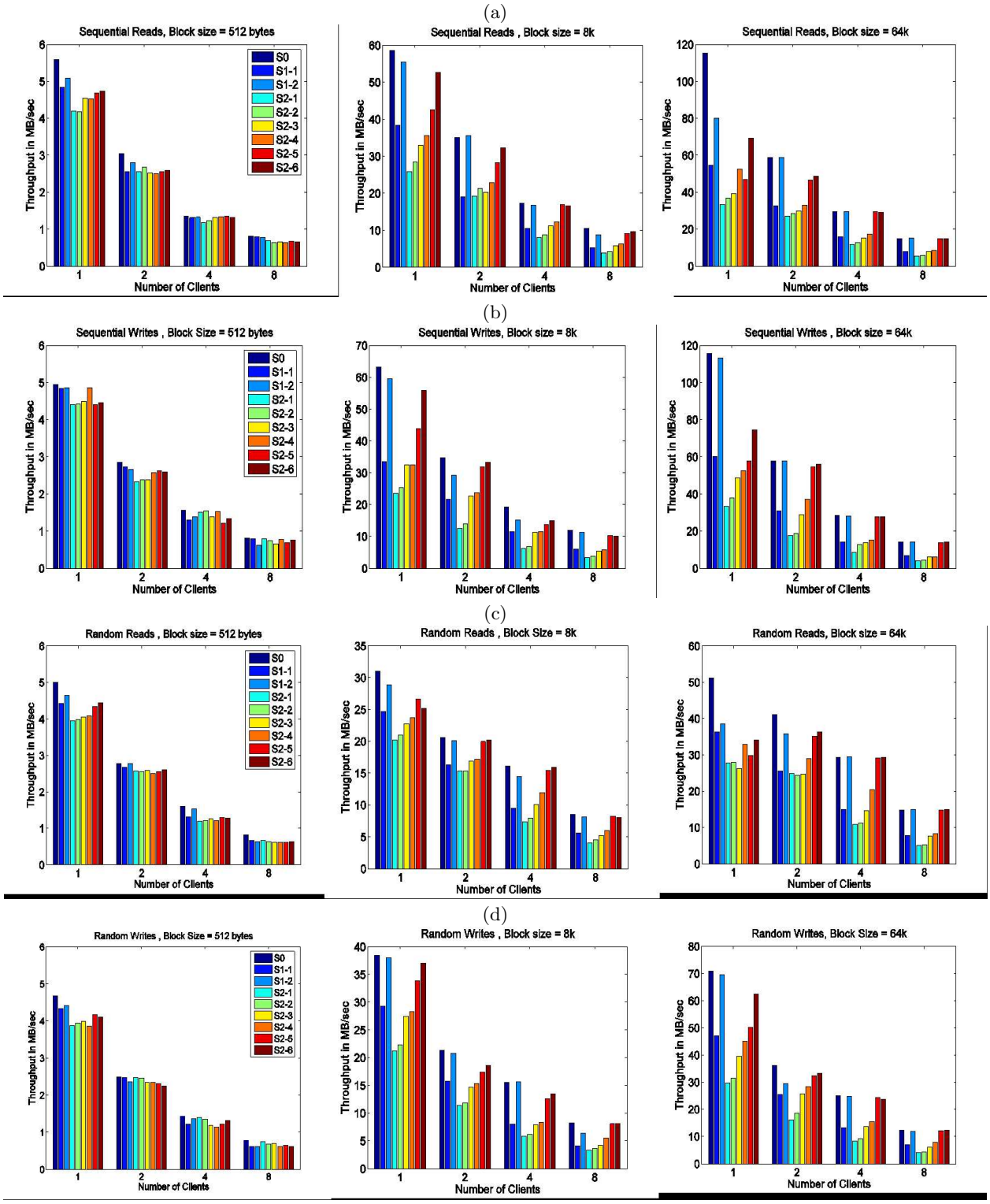

Figure 6: Point to Point throughput of Rawio for (a) sequential reads, (b) sequential writes, (c) random reads, (d) random writes

Table 3: Completion time of BTIO measured in seconds

	S0	S1-1	S1-2	S2-1	S2-3	S2-5
One client	444.84	446.27	445.36	462.15	454.32	449.13
Four clients	144.9	147.28	145.25	152.16	149.51	146.38
Nine clients	131.02	133.74	131.96	136.21	134.33	133.6

Table 4: Point-to-point throughput of dbench measured in MB/second

	S0	S1-1	S1-2	S2-1	S2-3	S2-5
One client	45.55	39.88	45.35	32.2	36.82	41.74
Two clients	27.72	26.75	27.16	21.11	23.33	24.48
Four clients	9.45	9.15	9.22	8.23	8.8	9.01
Eight clients	3.13	3.04	3.11	2.23	2.75	2.91

For the random workloads (Figures 6(c) and (d)), similar results can be inferred. The lazy approaches performed better in all cases compared to the IPSec based ones. For the single client case, the relative improvements are less compared to the sequential workloads. This is because of the lower bandwidth seen from the disk for random accesses. The maximum throughput for the No-security case is about 33 MB/sec for 8K and 52 MB/sec for 64K block sizes and does not saturate the Gigabit Ethernet bandwidth. Therefore, apart from the crypto processing, the bottleneck is limited by the disk transfer bandwidth.

The three bars labeled S2-2, S2-4 and S2-6 in the figures are lazy approaches implemented by performing encryption/decryption in hardware. It was found that the performance improvements by using the hardware was not much compared to the respective software implementations S2-1, S2-3 and S2-5. For smaller block sizes, the overhead of asynchronous processing, numerous hardware calls made via the PCI bus and interrupt processing become a bottleneck. With larger block sizes, there are minor performance benefits by off-loading to the crypto hardware but this does not scale with increasing number of clients

Table 3 shows the execution time of BTIO for the design spectrum outlined earlier in Table 2. We omit the design choices involving the hardware accelerator because the micro-benchmark results using Rawio indicate that there is not much benefit in off-loading crypto especially for smaller block sizes and increasing number of clients (since BTIO writes its data to the GFS file system which in turn uses a small block size). As seen from the results, BTIO yields poor performance benefits with lazy authentication and encryption schemes because of fewer, smaller and well-orchestrated I/O transfers and the reduced proportion of time spent in sending and receiving data blocks over the network.

Table 4 shows the throughput for dbench which is a more I/O bound workload and hence shows better performance results than BTIO. In particular we find that for the Security-1 design space, performing lazy authentication (S1-2) boosts throughput to as much as the scheme which does not do any crypto (S0). Likewise for the Security-2 design space, a combination of lazy decryption and authentication (S2-5) boosts throughput to about 90% of the no crypto scheme (S0) and 30% improvement over IPSec ESP.

5. RELATED WORK

Much of the work on storage security has been done at the file system level. A framework for evaluating security of diverse systems is discussed in [21], where they classify file systems into either encrypt-on-wire or encrypt-on-disk systems depending on where the encryption/decryption takes place. Examples of systems which implement encrypt-on-wire include PASIS [25] and S4 [23], where each transmission requires encryption or decryption. On the other hand, file systems such as CFS [6], SFS-RO [12], Plutus [19], SiRiUS [15] perform encrypt-on-disk. As pointed out in [21], the latter category provides confidentiality even in an untrusted server model and can also incur less overheads in data transfer. In our work, the lazy decryption approach is an encrypt-on-disk mechanism with the data stored in an encrypted form on the disks.

At the device level, disk drives have traditionally not provided any security mechanisms. It is usually left to the higher level software to provision the security mechanisms - either access control or confidentiality/authenticity/integrity. The OSD model [9] with embedded security [17] is one of the first proposals to incorporate smartness in the drive for security functionality. A discussion of security considerations when deploying Fibre Channel SANs can be found in [7]. The IP versions of these protocols - iSCSI,iFCP,FCIP are typically expected to use the underlying network layers (IPSec, SSL) for authentication, integrity and confidentiality. The work by Aguilera et al [4] addresses the access control mechanisms needed at the block level for network attached disks. The mechanisms proposed are similar in concept to the OSD model which provides security at object granularity.

There has been some work in evaluating the performance of these secure storage systems that are attached to the network. Authors in [17] describe one such system emphasizing on performance and scalability of network attached storage systems. The file management functions occur at a different location than the storage device. The file managers issue capabilities to client managers which then access the files stored on the device directly. A follow up report [16] shows how software-based cryptography cannot support storage's gigabit/sec transfer rates.

There is a need to secure SAN systems like the iSCSI systems, where the disk is directly attached to the network and block level accesses are made over the network. The security of the blocks at rest and in transit must be achieved with minimal performance overheads. [10] performs encryption at the block level. The proposed framework SNAD relies upon several standard public-key cryptographic tools. However the use of public-key cryptography decreases the throughput appreciably and the evaluation does not include high-level or standardized benchmarks. [8] performs a detailed study of iSCSI performance using different micro-benchmarks like IOmeter and real applications. However the study does not include any measurements of security overheads and scalability of iSCSI with increase in number of clients. In [22] the authors analyze the security requirements of iSCSI storage systems and examine the integration of two popular security schemes, IPSec and SSL in iSCSI. They also note that the two mechanisms show different throughput trends with different data sizes.

Although the performance of existing iSCSI implementations without any security considerations have been well studied ([3, 24, 26, 18, 27, 11]), we analyze the performance of iSCSI with and without security considerations for a set of micro and application benchmarks. To our knowledge,

ours is the first proposal which exploits the secret key of the lower level secure protocol to secure communication between machines at the application level without sacrificing performance.

6. CONCLUDING REMARKS

In this paper, we analyze the security and performance requirements of iSCSI based network storage systems. Existing techniques like IPSec that provide security at the network layer incur large overheads for iSCSI which are based on the fast growing Ethernet technology. We identify the the server end crypto processing of data packets as a significant bottleneck in the overall system throughput. To alleviate the processing overheads, we propose lazy crypto approaches which reduce the amount of work that needs to be done at the server. Our results show that the lazy techniques perform better than the IPSec based ones and result in better scalability with the increase in number of clients.

7. ACKNOWLEDGEMENTS

This work has been supported by NSF grant 0621429. This work was partially supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-Eng-38.

8. REFERENCES

[1] Linux iSCSI Target Implementation, ARDIS Technologies. http://www.ardistech.com/iscsi.

[2] Open-iSCSI Project, A Multi-Platform Implementation of iSCSI RFC. http://www.open-iscsi.org.

[3] A. Chadda and A. Palekar and R. Russel and N. Ganapathy. Design, implementation and performance analysis of the iSCSI protocol for SCSI over TCP/IP. In *Proceedings of the Internetworking 2003 conference*, 2003.

[4] M. Aguilera, M. Ji, M. Lillibridge, J. Maccormick, E. Oertli, D. Andersen, M. Burrows, T. Mann, and C. Thekkath. Block-level Security for Network-Attached Disks. In *Proceedings of the Conference on File and Storage Technologies*, 2003.

[5] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS Parallel Benchmarks. *The International Journal of Supercomputer Applications*, pages 63–73, Fall 1991.

[6] M. Blaze. A Cryptographic File System for UNIX. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 9–16, 1993.

[7] E. D. Burgh. SAN Security Beyond Segmentation, 2004. http://www.sans.org/rr/whitepapers/honors/1489.php.

[8] D. Xinidis and M. D. Flouris and A. Bilas. Performance Evaluation of Commodity iSCSI-based Storage Systems. In *Proceedings of the Twenty Second IEEE, Thirteenth NASA Goddard Conference on Mass Storage Systems and Technologies*, 2005.

[9] T. W. Draft. Information Technology - SCSI Object Based Storage Device Commands (OSD). http://www.t10.org/ftp/t10/drafts/osd/osd-r10.pdf.

[10] E. Miller and W. Freeman and D. Long and B. Reed. Strong Security for Network-Attached Storage. In *Proceedings of the Conference on File and Storage Technologies*, 2002.

[11] F. Tomonori and O. Masanori. Performance of Optimized Software Implementation of iSCSI. In *Proceedings of the Workshop on Storage Network Architecture and Parallel I/Os (SNAPI)*, 2003.

[12] K. Fu, M. F. Kaashoek, and D. Mazieres. Fast and Secure Distributed Read-only File System. In *Proceedings of the Usenix Symposium on Operating Systems Design and Implementation*, pages 181–196, October 2000.

[13] G. Caronni and H. Lubich and A. Aziz and T. Markson and R. Skrenta. SKIP-securing the Internet. In *Proceedings of the 5th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'96)*, page 62, Washington, DC, USA, 1996. IEEE Computer Society.

[14] G. R. Goodson and J. J. Wylie and G. R. Ganger and M. K. Reiter. Efficient Byzantine-Tolerant Erasure-Coded Storage. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, page 135, Washington, DC, USA, 2004. IEEE Computer Society.

[15] E. Goh, H. Shacham, N. Modadugu, and D. Boneh. SiRiUS: Securing Remote Untrusted Storage. In *Proceedings of the Internet Society Network and Distributed Systems Security Symposium*, 2003.

[16] H. Gobioff and D. Nagle and G. Gibson. Integrity and Performance in Network Attached Storage. Technical Report CMU-CS-98-182, Carnegie Mellon University - School of Computer Science, 1998.

[17] H. Gobioff and G. Gibson and D. Tygar. Security for Network Attached Storage Devices. Technical Report CMU-CS-97-185, Carnegie Mellon University - School of Computer Science, 1997.

[18] H. M. Khosravi and A. Joglekar. Performance Characterization of iSCSI processing in a server platform. In *Proceedings of the Twenty Fourth IEEE International Performance Computing and Communications Conference*, 2005.

[19] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus – Scalable Secure File Sharing on Untrusted Storage. In *Proceedings of the Conference on File and Storage Technologies*, pages 29–42, March 2003.

[20] M. Abd-El-Malek and G. R. Ganger and M. K. Reiter and J. J. Wylie and G. R. Goodson. Lazy Verification in Fault-Tolerant Distributed Storage Systems. In *Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pages 179–190, Washington, DC, USA, 2005. IEEE Computer Society.

[21] E. Riedel, M. Kallahalla, and R. Swaminathan. A Framework for Evaluating Storage System Security. In *Proceedings of the Conference on File and Storage Technologies*, pages 15–30, 2002.

[22] S-Yi Tang and Y. Lu and D. Du. Performance Study of Software-based iSCSI Security. In *Proceedings of*

the First International IEEE Security in Storage Workshop, 2002.

[23] J. D. Strunk, G. R. Goodson, M. L. Scheinholtz, C. A. N. Soules, and G. R. Ganger. Self-Securing Storage: Protecting Data in Compromised Systems. In *Proceedings of the Usenix Symposium on Operating Systems Design and Implementation*, pages 165–180, October 2000.

[24] W. T. Ng and H. Sun and B. Hillyer and E. Shiver and E. Gabber and B. Ozden. Obtaining high performance for storage outsourcing. In *Proceedings of the 2001 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 2001.

[25] J. J. Wylie, M. W. Bigrigg, J. D. Strunk, G. R. Ganger, H. Kilite, and P. K. Khosla. Survivable Information Storage Systems. *IEEE Computer*, 33(8):61–68, August 2000.

[26] X. He and Q. Yang. A Caching Strategy to improve iSCSI performance. In *Proceedings of the Twenty Seventh Annual IEEE International Conference on Local Computer Networks*, 2002.

[27] Y. Shastry and S. Klotz and R. Russell. Evaluating the effect of iSCSI protocol parameters on performance. In *Proceedings of the Parallel and Distributed Computing and Networks*, 2005.