ES-MPICH2: A Message Passing Interface with Enhanced Security

Xiaojun Ruan, Qing Yang, Mohammed I. Alghamdi†, Shu Yin,
Zhiyang Ding, Jiong Xie, Joshua Lewis, and Xiao Qin‡
Department of Computer Science and Software Engineering
Auburn University, Auburn, AL 36849-5347
Email: {xzr0001, yangqin, szy0004, dingzhi, jzx0004, jtl0003}@eng.auburn.edu
†Department of Computer Science
Al-Baha University,
Al-Baha City, Kingdom of Saudi Arabia
‡xqin@auburn.edu
‡http://www.eng.auburn.edu/∼xqin

***Abstract*—In largely distributed clusters, computing nodes are geographically deployed in various computing sites. Information processed in a distributed cluster is shared among a group of distributed processes or users by virtue of messages passing protocols (e.g. message passing interface - MPI) running on the Internet. Because of the open accessible nature of the Internet, data encryption for these large-scale distributed clusters becomes a non-trivial and challenging problem. To address this issue, we enhanced the security of the MPI (Message Passing Interface) protocol by encrypting and decrypting messages sent and received among computing nodes. In this study we focused on MPI rather than other protocols because MPI is one of the most popular communication protocols for cluster computing environments. From among a variety of MPI implementations, we picked MPICH2 developed by the Argonne National Laboratory. The design goal of MPICH2 - a widely used MPI implementation - is to combine portability with high performance. We integrated encryption algorithms into the MPICH2 library so that data confidentiality of MPI applications could be readily preserved without a need to change the source codes of the MPI applications. since we provide a security enhanced MPI-library with the standard MPI interfact, data communications of a conventional MPI program can be secured without converting the program into the corresponding secure version. We used Sandia Micro Benchmark and Intel MPI Benchmarks to evaluate and compared the performance of original MPICH2 and Enhanced Security MPICH2. According to the performance evaluation, ES-MPICH2 provides secured Message Passing Interface by sacrificing reasonable system performance.**

I. Introduction

Large cluster computing systems have been widely deployed and utilized by national laboratories, corporations, and government research centers. Some clusters were usually built upon local area networks, which are physically isolated from outside networks like the Internet, consequently, the security issue of information exchanging among computing nodes in a locally operated cluster is not a major concern. Without any security mechanism to preserve confidentiality, messages transferred among the computing nodes are just plain-texts that can be easily tempered and manipulated. However, due to the fast development of the Internet, an increasing number of universities and companies are connecting their cluster computing systems to public networks to provide high accessibility. Those networks connecting to the Internet can be accessed by anyone from anywhere, thereby opening a possibility for security leakages if there is no information assurance protection on classified or confidential data transmitted to and from cluster computing nodes.

Recently, a geographically distributed cluster system proposed by Sun Microsystems has attracted many interests from both academia and industry communities. In largely distributed clusters, computing nodes are geographically deployed in various computing sites. Information processed in a distributed cluster is shared among a group of distributed processes or users by the virtue of messages passing protocols (e.g. message passing interface - MPI) running on the Internet. Because of the open accessible nature of the Internet, data encryption for these large- scale distributed clusters becomes a non-trivial and challenging problem. To address this issue, we enhanced the security of the MPI (Message Passing Interface) protocol by encrypting and decrypting messages sent and received among computing nodes.

In this study we focus on MPI rather than other protocols, because MPI is one of the most popular communication protocols for cluster computing environments. Numerous scientific and commercial applications running on clusters were developed using the MPI protocol. From among a variety of MPI implementations, we picked MPICH2 developed by the Argonne National Laboratory. The design goal of MPICH2 - a widely used MPI implementation - is to combine portability with high performance [12]. We integrated encryption algorithms into the MPICH2 library, thus, data confidentiality of MPI applications can be readily preserved without a need to change the source codes of the MPI applications. Data communications of a conventional MPI program can be secured without converting the program into the corresponding secure version, since we provide a security enhanced MPI-library with the standard MPI interfact. In what follows, we summarize the four major contributions of this study:

- We implemented a standard MPI mechanism called ES-

978-1-4244-9328-9/10/$26.00 ©2010 IEEE

MPICH2 to offer data confidentiality for secure network communications in message passing environments. Our proposed security technique incorporated in the MPICH2 library can be very useful for protecting data transmitted in open networks like the Internet.

• The ES-MPICH2 mechanism allows application programmers to easily write secure MPI applications without additional code for data-confidentiality protection. We seek a channel-level solution in which encryption and decryption functions are included into the MPICH2 library. Our ES-MPICH2 maintains a standard MPI interface to exchange messages while preserving data confidentiality.

• The implemented ES-MPICH2 framework provides a configuration file that enables application programmers to selectively choose any cryptographic algorithm and key integrated in ES-MPICH2. This feature makes it possible for programmers to easily and fully control the security services incorporated in the MPICHI2 library. To demonstrate this feature, we implemented the AES and Triple DES algorithms in ES-MPICH2. We will also show in this paper how to add other cryptographic algorithms in to the ES-MPICH2 framework.

• We have used ES-MPICH2 to perform a detailed case study using the Sandia Micro Benchmarks and the Intel MPI benchmarks. We focus on runtime performance overhead introduced by the cryptographic algorithms.

The paper is organized as follows: Section II presents the reason why we choose MPICH2 implementaion. Section III presents the motivation of this work by showing why secured MPI is an important issue, and it also outlines the design of ES-MPICH2 - the message passing interface with enhanced security. Section IV describes the corresponding implementation details of ES-MPICH2. Section V discusses some experimental results and compares the performance of ES-MPICH2 with that of MPICH2. Section VI presents previous research related to our project. Finally, Section VII states the conclusions and future work of this study.

II. MPICH2 Overview

MPICH is one of the most popular MPI implementations developed at the Argonne National Laboratory [12]. The early MPICH version supports the MPI-1 standard. MPICH2 - a successor of MPICH - not only provides support for the MPI-1 standard, but also facilitates the new MPI-2 standard, which specifies functionalities like one-sided communicaiton, dynamic process management, and MPI I/O [10]. Compared with the implementation of MPICH, MPICH2 is completely redesigned and developed to achieve high performance, maximum flexibility, and good portability.

Fig. 1 shows the hierarchical structure of the MPICH2 implementation, where there are four distinct layers of interfaces to make the MPICH2 design portable and flexible. The four layers, from top to bottom, are the message passing interface 2 (MPI-2), the abstract device interface (ADI3), the channel interface (CH3), and the low-level interface. ADI3 - the third generation of the abstract device interface - in the

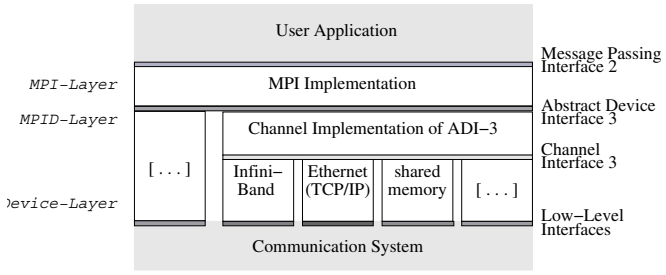

Fig. 1. Hierarchical Structure of MPICH2

hierarchical structure (see Fig. 1) allows MPICH2 to be easily ported from one platform to another. Since it is non-trivial to implement ADI3 as a full-featured abstract device interface with many functions, the CH3 layer simply implements a dozen functions in ADI3 [14]. As shown in Fig. 1, the TCP socket Channel, the shared memory access (SHMEM) channel, and the remote direct memory access (RDMA) channel are all implemented in the layer of CH3 to facilitate the ease of porting MPICH2 on various platforms. Note that each one of the aforementioned channels implements the CH3 interface for a corresponding communication architecture like TCP sockets, SHMEM, and RDMA. Unlike an ADI3 device, a channel is easy to be implemented since one only has to implement a dozen functions relevant for with the channel interface.

To improve the security of MPICH2, we implemented a standard MPI mechanism or ES-MPICH2 to offer data confidentiality services in message passing environments.

III. Description of ES-MPICH2

A. Motivation

Computing nodes in a distributed cluster are geographically deployed in several computing sites on an open network. Processes within a communication group need to transmit security-sensitive messages through unsecured communication channels. Preserving data confidentiality in a message passing environment over an untrusted network is critical for a wide spectrum of security-aware MPI applications because unauthorized access to the security-sensitive messages by untrusted processes can lead to serious security breaches. Hence, it is imperative to protect confidentiality of messages exchanged among a group of trusted processes.

There are three common approaches to improving security of MPI applications. In the first approach, application programmers can add source code to address the issue of message confidentiality. Such an application-level security approach not only makes the MPI applications error-prone, but it also reduces the portability and flexibility of the MPI applications. In the second approach, the MPI interface can be extended in the same way that new security-aware APIs are designed and implemented (see, for example, MPISec I/O [15]). This MPI-interface-level solution enables programmers to write secure MPI applications with minimal changes to the interface.

Although the second approach is better than the first, this MPI-interface-level solution typically requires extra code to deal with data confidentiality. The third approach, a channel-level solution, is proposed in this study to address the drawbacks of the aftermentioned two approaches. The channel-level solution aims at providing message confidentiality in a communication channel that implements the Channel Interface 3 in MPICH2 (see Fig. 1).

B. The Design of ES-MPICH2

The goal of the development of the ES-MPICH2 mechanism is to enable application programmers to easily implement secure enhanced MPI applications without additional code for data-confidentiality protection. In our programming model, processes belonging to a communication group trust each other. Any process that is outside of this communication group is not trusted by the member processes in the group. In other words, trusted processes within a communication group (i.e., a group of processes spawned in an MPI application) can efficiently encrypt and decrypt messages delivered among the trusted processes. The communication group as a whole will have to control the encryption and decryption procedures. With ES-MPICH2 in place, secure MPI application programmers are able to flexibly choose a cryptographic algorithm, key size, and data block size for each MPI application that needs data confidentiality protection.

ES-MPICH2 offers message confidentiality in an MPI programming environment by incorporating MPICH2 with encryption and decryption algorithms. In the process of designing ES-MPICH2, we integrated the AES (Advanced Encryption Standard) and 3DES (Triple Data Encryption Standard) algorithms into the MPICH2 library. Unlike the application-level and MPI-interface-level security solutions, ES-MPICH2 encrypts and decrypts messages at the transmission layer without modifying the MPI interface or application source code. ES-MPICH2 maintains a standard MPI interface to exchange messages while preserving data confidentiality. Thus, this feature of ES-MPICH2 allows application programmers to easily write secure MPI applications with no requirement of additional code to deal with message confidentiality protection.

The ES-MPICH2 implementation has the following four design goals:

• **Message Confidentiality:** ES-MPICH2 aims to preserve message confidentiality from unauthorized accesses by untrusted processes. We leverage AES to protect the confidentiality of messages, because AES is an encryption standard adopted by the U.S. government. For comparison purpose, we also considered 3DES in the design of ES-MPICH2. AES with 128-bit keys can provide adequate protection for classified messages up to the SECRET level. The implementation of AES in products intended to protect national security systems and/or information must be reviewed and certified by NSA prior to their acquisition and use [1]. In the next section, we will explain why we paid particular attention on block ciphers like AES. In this study, we integrated data confidentiality services with MPICH2 by implementing the cryptographic algorithms in a CH3 channel. We will also discuss in Section IV the integration of security services with the TCP socket channel in the CH3 layer of MPICH2.

• **Complete Transparency:** ES-MPICH2 is intended to improve the security of any conventional MPI program without adding extra code to perform message protection. Thus, preserving message confidentiality in MPICH2 is entirely transparent to application programmers. Such confidentiality transparency is feasible and the reason is two-fold: first, the encryption and decryption processes can be built in the MPICH2 library at the channel transmission layer, and second, we can maintain the same interface as the APIs of the MPICH2 implementation. Therefore, we can enhance the security of conventional MPI applications by running the applications in the ES-MPICH2 environment without modifying the source code.

• **Compatibility and Portability:** Ideally, ES-MPICH2 needs to be easily ported from one platform to another with no addition to the application source code. ES-MPICH2 is an extension of MPICH2, therefore, thus, ES-MPICH2 should have the same level of portability as MPICH2. However, it is challenging to achieve high portability in ES-MPICH2 because we have to implement a cryptographic subsystem in each channel in the CH3 layer in MPICH2. In the current version of ES-MPICH2, only a secured TCP socket channel has been implemented and tested. Thus, ES-MPICH2 should work without modifications on any parallel and distributed computing system that supports TCP communications and the MPICH2 library. The SHMEM and RDMA channels have not been completely supported in ES-MPICH2; therefore, the focus of this paper is the TCP socket channel in MPICH2. Nevertheless, one can take a similar approach described in Section IV to integrating data confidentiality services with the SHMEM and RDMA channels in the CH3 layer of MPICH2.

• **Extensibility:** ES-MPICH2 must allow application programmers to selectively choose any cipher techniques and keys incorporated in MPICH2. This design goal makes it possible for programmers to flexibly select any cryptographic algorithm implemented in ES-MPICHI2. Although we implemented AES and 3DES in the channel layer of MPICH2, we will show in the next section how to add other cryptographic algorithms (e.g., Elliptic Curve Cryptography, [2]) to the ES-MPICH2 environment.

IV. Implementation Details

During the implementation of ES-MPICH2, we have addressed the following implementation questions: (1) Among the multiple layers in the hierarchical structure of MPICH2, in which layer should we implement cryptographic algorithms? (2) Which cryptographic subsystems should we choose to implement? (3) How to implement secure key management? (4) How to use the implemented ES-MPICH2?

Ciphers in the channel layer: Fig. 2 shows that the message passing implementation structure in MPICH2. In this hierarchical structure of MPICH2, messages are passed from a sender to a receiver through the abstract device interface

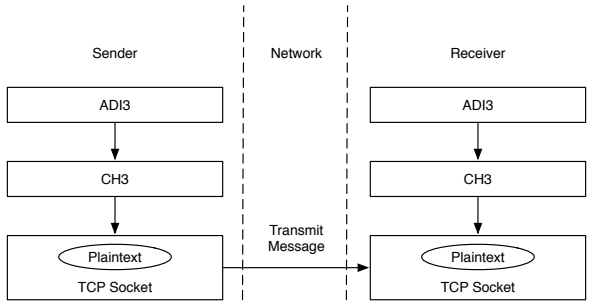

Fig. 2. Message Passing Implementation Structure in MPICH2

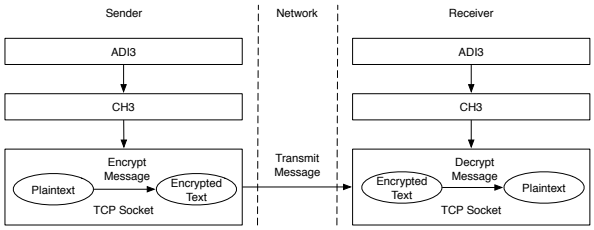

Fig. 3. Message Passing Implementation Structure in ES-MPICH2 with Encryption and Decryption Processes

(ADI3), the channel interfact (CH3), and the TCP socket channel. Cryptographic subsystems can be implemented in one of the three layers (i.e., ADI3, CH3, or TCP socket channel). To achieve the design goal of complete transparency, we chose to implement the cryptographic algorithms in the TCP socket channel. Fig. 3 depicts the implementation structure of ES-MPICH2, where the encryption and decryption processes were implemented in the TCP socket layer. Thus, messages are encrypted and decrypted in the TCP socket channel rather than the ADI3 and CH3 layers.

Fig. 4 shows that the encryption and decryption functions in ES-MPICH2 interact with the TCP socket to provide message confidentiality protection in the TCP socket layer. Before a message is delivered through the TCP socket channel, certain encryption algorithm like AES and 3DES encrypt data contained in the message. Upon the arrival of an encrypted message in a receiving node, the node invokes the corresponding decryption algorithm to decrypt the encrypted message. Fig. 4 demonstrates that ES-MPICH2 maintains a standard MPI interface by implementing the encryption and decryption algorithms in the TCP socket level. All modifications and configurations are carried out in the MPICH2 libraries, thereby being totally transparent to application programmers.

Block ciphers: In ES-MPICH2, the AES and 3DES cryptographic algorithms are implemented in MPICH2-1.0.7. We focused on block ciphers in the implementation of ES-MPICH2, because a block cipher transforms a fixed-length block of plaintext into a block of ciphertext of the same length. In the case where ciphertext and plaintext differ in length, MPI applications have to be aware of such difference. Therefore, we chose to focus on block ciphers so that MPI applications

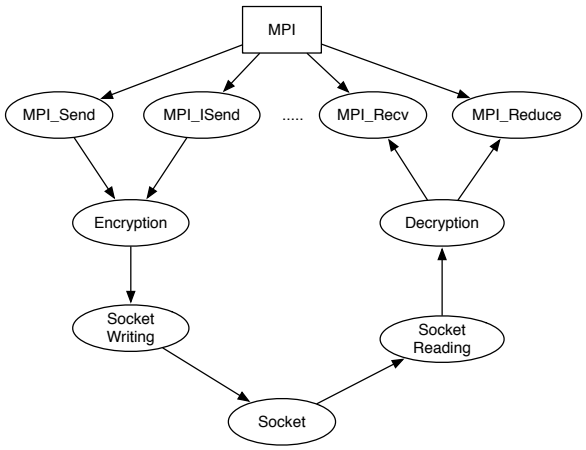

Fig. 4. The Interface between the Encryption/Decryption Processes and the TCP socket

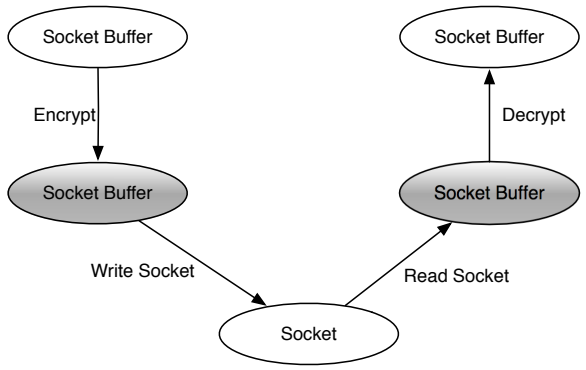

Fig. 5. ES-MPICH2 Socket Details

would not be aware of the lengths of plaintext and ciphertext.

Key management: In order to ensure secure key management, we are going to use a Key Management Infrastructure to share communication keys in ES-MPICH2. Key Management Infrastructure in ES-MPICH2 is different from MPISec I/O because the communication keys are set up in configuration files. Fig. 6 presents the key management infrustructure in ES-MPICH2. Communication keys are shared by using public-private key cryptographic algorithms through unsecure network connections. When the master node tries to share a new

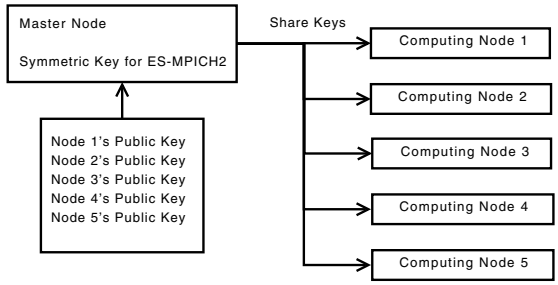

Fig. 6. Key Management Infrastructure

key with other nodes, it uses the master nodes' public key to encrypt the communication key and send the encrypted key to computing nodes. Each computing node receives the encrypted key and decrypts it with its own private key. This guarantees the security of the communication keys.

Socket programming: In socket programming, there is a buffer to contain data for sending and receiving data through TCP socket channel. Fig. 5 presents the ES-MPICH2 encryption and decryption process in detail. Originally, MPICH2 fills the socket buffer with data, then writes the data to the socket on the sending node; the receiving node the fills the receiving buffer with the data read from the socket. Thus, in ES-MPICH2, the sending buffer receives plain data first, then ES-MPICH2 encrypts the data in the buffer and replaces the buffer with encrypted data. Finally, the data written in socket is encrypted data. On the receiving node, receiving buffer is filled with the encrypted data read from the socket directly. Then MPICH2 decrypts the data and replaces receiving buffer with decrypted data. Because the plaintext and cipher text have the same length in block cipher algorithms, we do not need to worry about the buffer size changing after encryption and decryption at this point.

Usage: There are some configurations we need to set up before we can run ES-MPICH2. First, security features could be turned on or off. If a security feature is turned off, ES-MPICH2 will work exactly as the original MPICH2. Second, we need to set up which cryptographic algorithm we want to use. At this point, we could choose either AES or Triple DES. It is also easy to implement other block cipher algorithms in ES-MPICH2. This feature makes ES-MPICH2 very extendable. Third, we need to set up the encryption and decryption keys. Because both AES and Triple DES are symmetric key cryptographic algorithms, the encryption key and decryption key are the same. After the setup, users can compile or run their programs in the same way as in MPICH2. If the programs are able to run on MPICH2, they are also able to run on ES-MPICH2 without any modification.

V. Experimental Evaluation

A. Experimental Testbed

We evaluated the performance of both original MPICH2 and ES-MPICH2 on a 6-node cluster. Table I shows the cluster configuration. All six nodes have exactly the same configuration. Operating System running on the cluster is Ubuntu 9.04 Jaunty Jackalope. The nodes in the cluster are connected by Ethernet adapters, Ethernet cables, and one 1Gbit switch. All nodes share disk space on master node through Network File System (NFS). [17] We choose MPICH2-1.0.7 as our Message Passing Interface on our cluster. Then we use Sandia Micro Benchmark and Intel MPI Benchmarks to evaluate the performance difference among original MPICH2, AES version of ES-MPICH2, and Triple DES version of ES-MPICH2. The key length of Enhanced Security MPICH2 is 192-bit.

TABLE I
Cluster Configuration

	Node ×6
CPU	Intel Celeron 450 2.2GHz
Memory	2GB
OS	Ubuntu 9.04 Jaunty Jackalope
Kernel version	2.6.28-15-generic
Network	1000Mbps

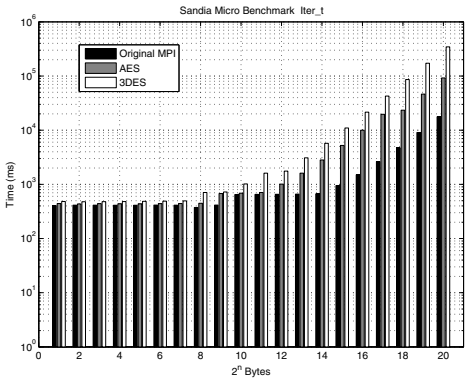

Fig. 7. Sandia Micro Benchmark iter_time

B. Sandia Micro Benchmark

Sandia National Laboratory develops Sandia Micro Benchmark (SMB) to evaluate and test high-performance networks and protocols.

Fig. 7 presents iter_t consumption by the three MPI implementations when the message size veries from 1 byte to 2MByte. Fig. 8, Fig. 9, Fig. 10, Fig. 11, and Fig. 12 present the result of iter_t, work_t, overhead_t, and base_t. The message sizes are set to 2KB, 16KB, 128KB, 512KB, and 1024KB respectively. Each experiment iterates 1000 times. According to the results, AES MPI and 3DES MPI cost more time than original MPI. When messages get larger and larger, the time consumed by AES MPI and 3DES MPI grows faster than that of orignal MPI. As block cipher algorithms, AES is more efficient than 3DES. This is the reason why 3DES MPI consumes even more time than AES. Because the workload only depends on message size at this point no matter what system we use, the encryption workload will be the same if the message size is the same. Hence, the time difference between ES-MPICH2 and original MPI could be much smaller in a cluster with powerful computing nodes.

C. Intel MPI Benchmarks

IMB has benchmarks for both MPI-1 [7] and MPI-2 [9] standards. In this section, we will evaluate the performance of ES-MPICH2 and original MPI by using the benchmarks for both MPI-1 and MPI-2 standards. There are three main classifications of IMB-MPI1: Single Transfer, Parallel Transfer, and Collective. Single Transfer benchmarks expect undisturbed results since they run dedicated. In Parallel Transfer Benchmarks, there are concurrent actions. Collective benchmarks are implemented to test higher level collective functions.

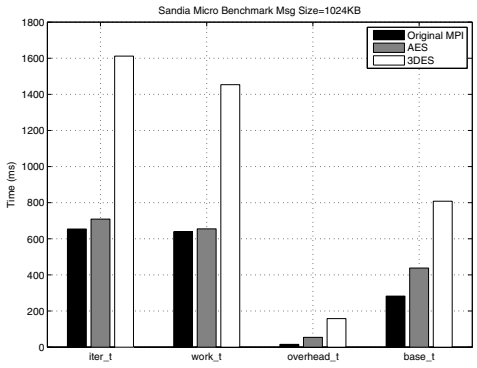

Fig. 8. Sandia Micro Benchmark Message Size is 2KB

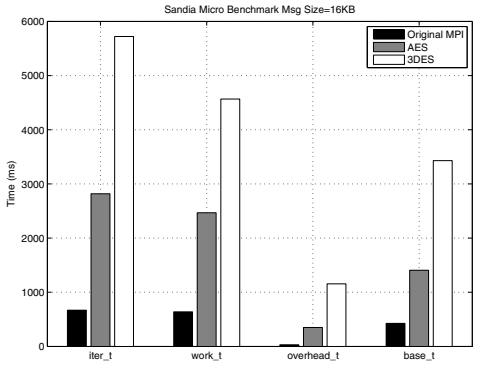

Fig. 9. Sandia Micro Benchmark Message Size is 16KB

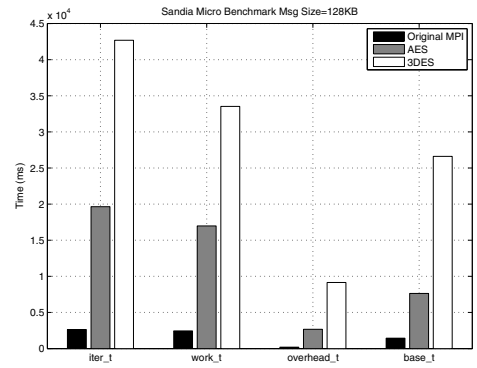

Fig. 10. Sandia Micro Benchmark Message Size is 128KB

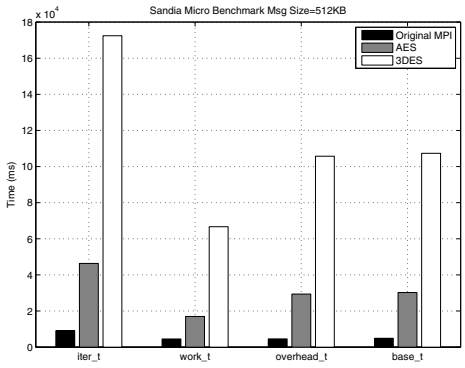

Fig. 11. Sandia Micro Benchmark Message Size is 512KB

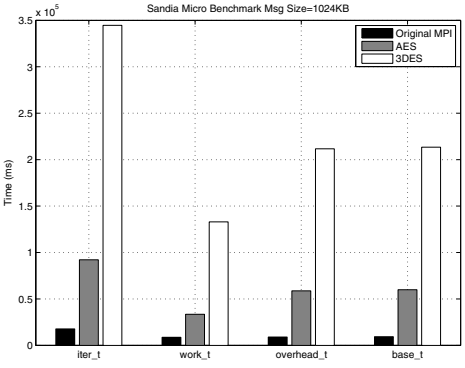

Fig. 12. Sandia Micro Benchmark Message Size is 1024KB

Fig. 13 and Fig. 14 present the performance of benchmark PingPong and PingPing. PingPong and PingPing are both Single Transfer benchmarks. Single Transfer benchmarks only ran with 2 active processes. In this experiment, they only run with 2 nodes. When the message size increases, the time consumption increases as well. Because a larger message means more encryption and decryption workload, AES and 3DES time consumption increases faster than original MPI. The performance of AES MPICH2 is closer to the original MPI than 3DES MPI.

The benchmarks Sendrecv and Exchange are Parallel Transfer benchmarks. Sendrecv is based on MPI_Sendrecv function and the processes form a periodic communication chain. Exchange is based on MPI_ISend, MPI_Waitall, and MPI_Recv. Unlike Single Transfer benchmarks, transfer operations in Parallel benchmarks could be in parallel.

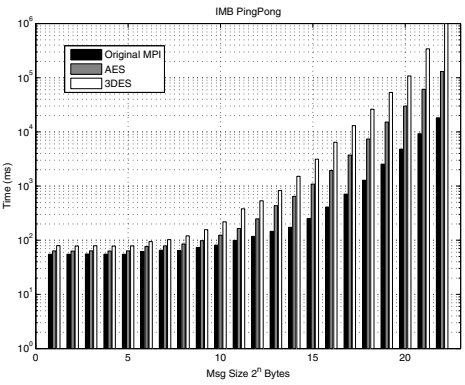

Fig. 13. Intel MPI Benchmarks PingPong

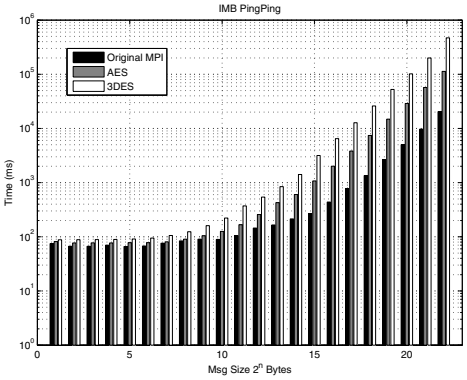

Fig. 14. Intel MPI Benchmarks PingPing

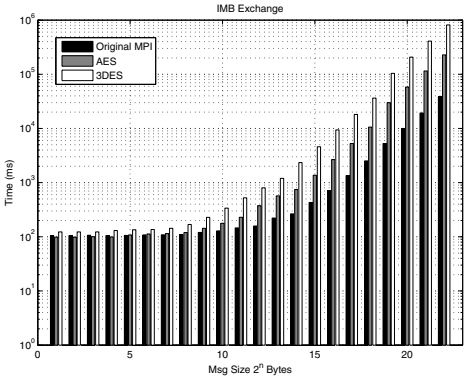

Fig. 16. Intel MPI Benchmarks Exchange

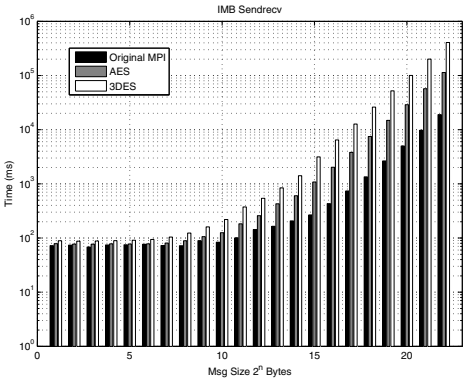

Fig. 15. Intel MPI Benchmarks SendRecv

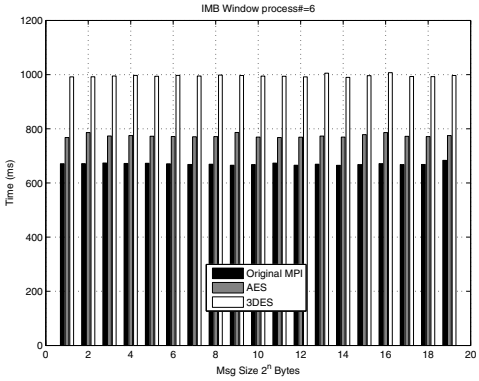

Fig. 17. Intel Micro Benchmark Window, 6 nodes

Fig. 15 presents the results of SendRecv benchmark in the cluster. One node receives data from its left neighbor then sends data to its right neighbor. The values in the figures with different node numbers are similar. The reason is that no matter how many nodes are applied, the transfer operations are in parallel. The trends shows that performance of AES is always close to original MPI. The consumption of 3DES increases faster than the other 2 MPI implementations.

Fig. 16 presents the results of Exchange benchmark. Exchange consumes more time than Sendrecv under the same condition, the reason being that nodes exchange data with both left and right neighbors in the chain. Hence, the values in Exchange are roughly doubled of that in Sendrecv.

Window is a benchmark to test MPI-2 standard functions MPI_Win_create, MPI_Win_fence, and MPI_Win_free. Windows are created on local nodes based on different Window sizes. So even though Window sizes are different in the experiment, the messages passed among nodes are in the same length.

VI. Related Work

As early as 1997, Brightwell *et al.* of Sandia National Laboratory first pointed out the barriers to creating a secure MPI. [3] The barriers include control and data, and cryptographic issues. For control and data, they state both data and control information need security and authentication. For cryptographic issues, they mention a few negative performance impacts.

There are many implementations of MPI and MPI-2 standard. However, there is not sufficient work that has been done for secured MPI. Prabhakar, et al. proposed a secure interface for MPI I/O, called MPISec I/O. [15] MPISec I/O preserves both the advantages of Parallel I/O and data confidentiality. Programmers can manually set encryption rules in MPISec I/O. The data will be first encrypted and written onto disks, then read from disks and decrypted.

There are two disadvantages for MPISec I/O. First, MPISec I/O programmers need set up encryption and decryption rules very carefully in their codes. Otherwise, it is possible that some data are stored without encryption or read without decryption. Either way will spoil the data. Second, MPISec is not completely compatible with non-secure MPI libraries. All previous projects have to be updated in order to read/write encrypted data from/to disks. Previously stored data also need to be marked as NOT encrypted, or encrypted the data on the disks before apply MPISec, which adds additional workloads.

To apply encryption and decryption algorithms in this paper, we adopt the following implementations and algorithms:

MPI, or Message Passing Interface, is a specification for a standard library for message passing. [12] It is used on

clustered computers to let them communicate with each other. MPICH2 is one of the most popular and widely applied MPI implementation developed by the Argonne National Laboratory for cluster computing. It is aims at the combination of portability and high-performance and fully supports MPI-2 standard. [11] MPICH2 can be applied on high-performance switches, shared-memory architectures, and networks of workstations.

The Advanced Encryption Standard, or AES, is a block cipher and symmetric-key cryptographic algorithm. It is an encryption standard adopted by the U. S. government. National Institute of Standards and Technology chose AES to replace Data Encryption Standard (DES). AES was originally published as Rijndael. The expected effort of exhaustive key search of AES is 2191 applications of Rijndael for a 24-byte key. [6] [AES: RijndaelProposal] From a performance perspective, AES is faster than 3DES in software implementation [8].

Triple DES (3DES) applies Data Encryption Standard (DES) three times for each data block. [4] DES is also a block cipher and symmetric-key algorithm.3DES is designed to work more efficiently in hardware than software. The speed on hardware is significantly faster than the best software implementations. [8] [13] We use 3DES to show the effect of a slow cryptographic algorithm on the performance.

There are two groups of benchmarks are chosen to evaluate ES-MPICH2 and MPICH2 performance: Sandia Micro Benchmark (SMB) and Intel MPI Benchmarks (IMB). SMB is a micro benchmark written by Sandia National Laboratory for high-performance network test and evaluation. IMB is a benchmarks suit including about 10,000 lines codes to measure the most important MPI functions. [5] [16]

VII. Conclusion

The security of Message Passing Interface is becoming more and more important today. In this paper, we proposed Enhanced Security MPICH2 implementation and corresponding performance evaluation. ES-MPICH2 is a secure, compatible, portable, and completely transparent Message Passing Interface. It is based on MPICH2 developed by Argonne National Laboratory. Compared with original MPICH2, ES-MPICH2 provides security in Message Passing Interface by encrypting messages in sending nodes and decrypting messages in receiving nodes. We integrated two encryption algorithms, AES and Triple DES, into the MPICH2 library on TCP socket layer. ES-MPICH2 offers data confidentiality for secure network communications in message passing environments. ES-MPICH2 should work without modifications on any parallel and distributed computing system that supports TCP communications and MPICH2 library.

In the future, we may implement some stronger and more efficient cryptographic algorithms like Elliptic Cureve Cryptography in ES-MPICH2. In additional, we may integrate encryption algorithms on other communication channels, like SHMEM and InfiniBand, because more and more clusters are built on high speed networks.

Acknowledgment

The work reported in this paper was supported by the US National Science Foundation under Grants CCF-0845257 (CAREER), CNS-0757778 (CSR), CCF-0742187 (CPA), CNS-0917137 (CSR), CNS-0831502 (CyberTrust), CNS-0855251 (CRI), OCI-0753305 (CI-TEAM), DUE-0837341 (CCLI), and DUE-0830831 (SFS), as well as by Auburn University under a startup grant and a gift (Number 2005-04-070) from the Intel Corporation. The corresponding source code for ES-MPICH2 described in this paper can be found at http://www.eng.auburn.edu/∼xqin/software/es-mpich2/.

References

[1] National Security Agency. National policy on the use of the advanced encryption standard (aes) to protect national security systems and national security information cnss policy no. 15 fact sheet no. 1, June 2003.

[2] Ian F. Blake, G. Seroussi, and N. P. Smart. *Elliptic curves in cryptography*. Cambridge University Press, New York, NY, USA, 1999.

[3] Ron Brightwell, David S. Greenberg, Brian J. Matt, and George I. Davida. Barriers to creating a secure mpi, 1997.

[4] D. Coppersmith, D. B. Johnson, S. M. Matyas, T. J. Watson, Don B. Johnson, and Stephen M. Matyas. Triple des cipher block chaining with output feedback masking, 1996.

[5] Intel Corporation. Intel mpi benchmarks user guide and methodology description, 2008.

[6] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael, 1999.

[7] Jack J. Dongarra, Steve W. Otto, Marc Snir, and David Walker. An introduction to the mpi standard. Technical report, Knoxville, TN, USA, 1995.

[8] Adam J. Elbirt, W. Yip, B. Chetwynd, and C. Paar. An fpga-based performance evaluation of the aes block cipher candidate algorithm finalists. *IEEE Trans. Very Large Scale Integr. Syst.*, 9(4):545–557, 2001.

[9] Al Geist, William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing L. Lusk, William Saphir, Tony Skjellum, and Marc Snir. Mpi-2: Extending the message-passing interface. In *Euro-Par '96: Proceedings of the Second International Euro-Par Conference on Parallel Processing*, pages 128–135, London, UK, 1996. Springer-Verlag.

[10] R. Grabner, F. Mietke, and W. Rehm. Implementing an mpich-2 channel device over vapi on infiniband. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 184–, April 2004.

[11] Ren Grabner, Frank Mietke, and Wolfgang Rehm. Implementing an mpich-2 channel device over vapi on infiniband. *Parallel and Distributed Processing Symposium, International*, 9:184a, 2004.

[12] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel Comput.*, 22(6):789–828, 1996.

[13] P. Hamalainen, M. Hannikainen, T. Hamalainen, and J. Saarinen. Configurable hardware implementation of triple-des encryption algorithm for wireless local area network. In *ICASSP '01: Proceedings of the Acoustics, Speech, and Signal Processing, 200. on IEEE International Conference*, pages 1221–1224, Washington, DC, USA, 2001. IEEE Computer Society.

[14] J. Liu, W. Jiang, P. Wyckoff, D.K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen. Design and implementation of mpich2 over infiniband with rdma support. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 16–, April 2004.

[15] Ramya Prabhakar, Christina Patrick, and Mahmut Kandemir. Mpisec i/o: Providing data confidentiality in mpi-i/o. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:388–395, 2009.

[16] Subhash Saini, Robert Ciotti, Brian T. N. Gunney, Thomas E. Spelce, Alice Koniges, Don Dossa, Panagiotis Adamidis, Rolf Rabenseifner, Sunil R. Tiyyagura, and Matthias Mueller. Performance evaluation of supercomputers using hpcc and imb benchmarks. *J. Comput. Syst. Sci.*, 74(6):965–982, 2008.

[17] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. Network file system (nfs) version 4 protocol. 2003.