A Secure Mechanism for MPI in Parallel Computing

Uday Kishore Pusuluri, G. Satyanarayana
1,2Dept. of CSE, Sir C.R.R College of Engg, Eluru, AP, India

Abstract
An increasing number of commodity clusters are connected to each other by public networks, which have become a potential threat to security sensitive parallel applications running on the clusters. To address Interface (MPI) implementation to preserve confidentiality of messages communicated among nodes of clusters in an unsecured network. We focus on MPI rather than other protocols, because MPI is one of the most popular communication protocol for parallel computing on clusters. Our MPI implementation—called ES-MPICH2 was built based on MPICH2 developed by the Argonne National Laboratory. Like MPICH2, ES-MPICH2 aims at supporting a large variety of computation and communication platforms like commodity clusters and high-speed networks. We integrated encryption and decryption algorithms into the MPICH2 library with the standard MPI interface and; thus, data confidentiality of MPI applications can be readily preserved without a need to change the source codes of the MPI applications. MPI-application programmers can fully configure any confidentiality services in MPICH2, because a secured configuration file in ES-MPICH2 offers the programmers flexibility in choosing any cryptographic schemes and keys seamlessly incorporated in ES-MPICH2. We used the Sandia Micro Benchmark and Intel MPI Benchmark suites to evaluate and compare the performance of ES-MPICH2 with the original MPICH2 version. Our experiments show that overhead incurred by the confidentiality services in ES-MPICH2 is marginal for small messages. The security overhead in ES-MPICH2 becomes more pronounced with larger messages. Our results also show that security overhead can be significantly reduced in ES-MPICH2 by high-performance clusters. The executable binaries and source code of the ES-MPICH2 implementation are freely available at The executable binaries and source code of the ES-MPICH2 implementation are freely available at http://www.eng.auburn.edu/~xqin/software/es-mpich2/

Keywords
Parallel Computing, Computer Security, Message Passing Interface, Encryption

I. Introduction
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 clusters connecting to the internet can be accessed by anyone from anywhere. For example, computing nodes in a distributed cluster system proposed by Sun Microsystems are geographically deployed in various computing sites. Information processed in a distributed cluster is shared among a group of distributed tasks or users by the virtue of message passing protocols (e.g., message passing interface—MPI) or confidential data transmitted to and from cluster computing nodes. 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. It is a nontrivial and challenging problem to offer confidentiality services for large-scale distributed clusters because there is an open accessible nature of the open networks. To address this issue, we enhanced the security of the MPI protocol by encrypting and decrypting messages sent and received among computing nodes.

A. Possible Approaches
There are three possible approaches to improving security of MPI applications. In first approach, application programmers can add source code to address the issue of message confidentiality. For example, the programmers may rely on external libraries (e.g., SEAL [26] and Nexus [11]) to implement secure applications. Such an application-level security approach not only makes the MPI applications error prone, but also reduces the portability and flexibility of the MPI applications. In the second approach, the MPI interface can be extended in the way that new security-aware APIs are designed and implemented (see, for example, MPI Sec I/O [22]). 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 one, this MPI-interface-level solution typically requires an 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 above two approaches.

B. Contributions
In what follows, we summarize the four major contributions of this study.
We implemented a standard MPI mechanism called ES-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 secured configuration file that enables application programmers to selectively choose any cryptographic algorithm and symmetric-key in ES-MPICH2. This feature makes it possible for programmers to easily and fully control the security services incorporated in the MPICH2 library. To demonstrate this feature, we implemented the AES and 3DES algorithms in ES-MPICH2. We also show in this paper how to add other cryptographic algorithms into 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.
C. Roadmap

The paper is organized as follows: Section II demonstrates the vulnerabilities of existing MPI implementations by describing a security threat model for clusters connected by public networks. Section III not only provides a reason for focusing on the confidentiality issue of MPICH2 rather than other MPI implementations, but also gives an overview of the MPICH2 implementation. Section IV presents the motivation of this work by showing why secured MPI is an important issue and also outlines the design of ES-MPICH2—the message passing interface with enhanced security. Section V describes the corresponding implementation details of ES-MPICH2. Section VI discusses some experimental results and compares the performance of ES-MPICH2 with that of MPICH2. Section VII presents previous research related to our project. Finally, Section VIII states the conclusions and future work of this study.

II. Threat Model

A geographically distributed cluster system is one in which computing components at local cluster computing platforms communicate and coordinate their actions by passing messages through public networks like the Internet. To improve the security of clusters connected to the public networks, one may build a private network to connect an array of local clusters to form a large-scale cluster. Building a private network, however, is not a cost-effective way to secure distributed clusters. The Internet—a very large distributed system—can be used to support large-scale cluster computing. Using a public network, the internet becomes a potential threat to distributed cluster computing environments. We first describe the confidentiality aspect of security in clusters followed by three specific attack instances. We believe new attacks are likely to emerge, but the confidentiality aspect will remain unchanged. Confidentiality attacks attempt to expose messages being transmitted among a set of collaborating processes in a cluster. For example, if attackers gain network administrator privileges, they can intercept messages and export the messages to a database file for further analysis. Even without legitimate privilege, an attacker still can sniff and intercept all messages in a cluster on the public network. Such attacks result in the information leakage of messages passed among computing nodes in geographically distributed clusters. Cryptography and access control are widely applied to computer systems to safeguard against confidentiality attacks.

We identify the following three confidentiality attacks on MPI programs running on distributed clusters:

Sniffing message traffic. Message traffic of an MPI program can be sniffed. For example, when MPICH2 is deployed in a cluster connected by a Gigabit Ethernet network, attackers can sniff plaintext messages transmitted through the TCP socket. Message sniffing can reveal confidentiality-sensitive data, metadata, and information.

Snooping on message buffer. In an MPI program, buffers are employed to send and receive messages. Regardless of specific MPI implementations, message buffers are created before the send and receive primitives are invoked. Attackers who snoop into the message buffers in memory can access data and information without being given specific access privileges.

III. MPICH2 Overview

MPICH—one of the most popular MPI implementations—were developed at the Argonne National Laboratory [14]. 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 communication, dynamic process management, and MPI I/O [13]. Compared with the implementation of MPICH, MPICH2 was 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 CH3, and the low-level interface. ADI3—the third generation of the abstract device interface—in the hierarchical structure (see fig. 1) allows MPICH2 to be easily ported from one platform to another.

Since it is nontrivial to implement ADI3 as a full-featured abstract device interface with many functions, the CH3 layer simply implements a dozen functions in ADI3 [18]. 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 implement since one only has to implement a dozen functions relevant for with the channel interface. To address the issues of message snooping in the message passing environments on clusters, we seek to implement a standard MPI mechanism with confidentiality services to counter snooping threats in MPI programs running on a cluster connected an unsecured network. More specifically, we aim to implement cryptographic algorithms in the TCP socket channel in the CH3 layer of MPICH2 (see fig. 2 and Section V for details of how to construct a cryptosystem in the channel layer).

IV. THE Design of ES-MPICH2

A. Scope of ES-MPICH2

Confidentiality, integrity, availability, and authentication are four important security issues to be addressed in clusters connected by an unsecured public network. Rather than addressing all the security aspects, we pay particular attention to confidentiality services for messages passed among computing nodes in an
unsecured cluster. Although preserving confidentiality is our primary concern, an integrity checking service can be readily incorporated into our security framework by applying a public-key cryptography scheme. In an MPI framework equipped with the public-key scheme, sending nodes can encode messages using their private keys. In the message receiving procedure, any nodes can use public keys corresponding to the private keys to decode messages. If one alters the messages, the cipher text cannot be deciphered correctly using public keys corresponding to the private keys. Thus, the receiving nodes can perform message integrity check without the secure exchange of secret keys. Please refer to Section 5.6 for details of how to add integrity checking services in our MPI framework.

V. Implementation Details
During the implementation of ES-MPICH2, we addressed the following five development questions:

- Among the multiple layers in the hierarchical structure of MPICH2, in which layer should we implement cryptographic algorithms?
- Which cryptosystem should we choose to implement
- How to implement secure key management?
- How to use the implemented ES-MPICH2?
- How to add integrity checking services to ESMPICH2?

A. Ciphers in the Channel Layer
MPICH2, messages are passed from a sending process to a receiving process through the abstract device interface, the CH3, and the TCP socket original version of MPICH2. In such a hierarchical structure channel. Cryptographic subsystems may be implemented in one of the three layers (i.e., ADI3, CH3, or the TCP socket channel). To achieve the design goal of a message passing implementation structure in the complete transparency, we chose to implement cryptographic algorithms in the TCP socket channel. Compared with ADI3 and CH3, the TCP socket channel is the lowest layer of the MPICH2 hierarchy. Implementing cryptosystems in the lowest layer can preserve message confidentiality in any conventional MPI program without adding extra code to protect messages. Fig. 3 depicts the implementation structure of ES-MPICH2, where a cryptosystem is 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, data contained in the message are encrypted by a certain cryptographic algorithm like AES and 3DES. Upon the arrival of an encrypted message in a receiving node, the node invokes the corresponding decryption function to decrypt the message. Fig. 4 demonstrates that ES-MPICH2 maintains the same application programming interface orAPI as that of MPICH2 by implementing the encryption and decryption algorithms in the TCP socket level. The confidentiality services of ES-MPICH2 were implemented in the MPICH2 libraries, thereby being totally transparent to MPI application programmers.

B. Block Ciphers
We have no intention of reinventing a cryptographic library, because it is very costly to guarantee that the security of your own implementation is higher than that of existing tested and audited security libraries. In the EMSPICH2 framework, we adopted the implementation of the AES and 3DES cryptographic algorithms offered by the Polar SSL library in MPICH2 version 1.0.7. Polar SSL is an open-source cryptographic library written in C. We focus on block ciphers in the implementation of ES-MPICH2, because a block cipher transforms a fixed-length block of plaintext into a block of cipher text of the same length.

C. Integrity Checking Evaluation Module
In this module we implement the integrity checking and evaluation process, while you can count on data integrity being addressed through multiple mechanisms for data. In this module we also show how we address the data integrity issues that require attention but occur after the data is sent. After data has been sent to destination, confirming the integrity of data is sent to destination only.

1. Signature Generation
For signing a message m by sender A, using A's private key dA
- Calculate e = HASH (m), where HASH is a cryptographic hash function, such as SHA-1
- Select a random integer k from \(1, n-1\)
- Calculate r = x1 (mod n), where \((x1, y1) = k \times G\). If \(r = 0\), go to step 2
- Calculate s = k − 1(e + dA r) (mod n). If \(s = 0\), go to step 2
- The signature is the pair \((r, s)\)

2. Signature Verification
For B to authenticate A’s signature, B must have A’s public key QA
- Verify that r and s are integers in \([1, n-1]\). If not, the signature is invalid
- Calculate e = HASH (m), where HASH is the same function used in the signature generation
- Calculate w = s − 1 (mod n)
- Calculate u1 = ew (mod n) and u2 = rw (mod n)
- Calculate \((x1, y1) = u1G + u2QA\)
- The signature is valid if \(x1 = r (mod n)\), invalid otherwise

VI. Related Work
Message passing interface. The Message Passing Interface Standard (MPI) is a message passing library standard used for the development of message-passing parallel programs [14]. The goal of MPI is to facilitate an efficient, portable, and flexible standard for parallel programs using message passing. MPICH2—developed by the Argonne National Laboratory—is one of the most popular and widely deployed MPI implementations in cluster computing environments. MPICH2 provides an implementation of the MPI standard while supporting a large variety of computation and communication platforms like commodity clusters, high-performance computing systems, and high-speed networks [13].

VII. Conclusion and Future Work
To address the issue of providing confidentiality services for large-scale clusters connected by an open unsecured network, we aim at improving the security of the message passing interface protocol by encrypting and decrypting messages communicated among computing nodes. In this study, we implemented the ES-MPICH2 framework, which is based on MPICH2. ES-MPICH2 is a secure, compatible, and portable implementation of the message passing interface standard. Compared with the original version of MPICH2, ES-MPICH2 preserves message confidentiality in MPI applications by integrating encryption techniques like AES.
and 3DES into the MPICH2 library. In light of ES-MPICH2, programmers can easily write secure MPI applications without an additional source code for data-confidentiality protection in open public networks.

References

Uday Kishore Pusuluri student of M.Tech. in Computer Science from C.R.Reddy College of Engineering, Eluru, West Godavari Dt, Andhra Pradesh.

Gunupusala Satyanarayana has completed his M.Tech in Computer Science and Engineering from JNTU Kakinada. His research in Data Warehousing & Data Mining, Network Security and Software Engineering etc. He is currently associated with Sir C R Reddy College of Engineering, ELURU, West Godavari District A.P. India. Affiliated to Andhra University.