Implementing Cryptographic Techniques in Message Passing Interface Systems

A.Brillia, D. Jagadiswary, R. Muthu Venkata Krishnan

Abstract*— In the concept of Message Passing Interface (MPI) chatting and file transmission the decryption part will be done automatically. Here three types of keys are used; they are public, private and secret key. Keys are displayed to the destination only if they accept the request or else displaying of key is not possible in the destination side and also it won't give or establish the Connection. In largely spread clusters, computing nodes are naturally deployed in a variety of computing sites. The Information processed in a spread cluster is communal among a group of distributed processes or client by high-quality of messages passing protocols (e.g. message passing interface - MPI) running on the Internet. Because of the open available nature of the Internet, data encryption for these large-scale distributed clusters becomes a non-trivial and challenging problem. We improved the security of the MPI protocol by encrypting and decrypting messages sent and received among computing nodes. We are listening carefully on MPI rather than more protocols because MPI is one of the most accepted communication protocols for cluster computing environments. From among a multiple of MPI implementations, we selected MPICH2 developed by the Argonne National Laboratory. Design goal of MPICH2 - a commonly use MPI implementation - is to join portability with high presentation. we gives a security enhanced MPI-library with the standard MPI interface, data communications of a conservative MPI program can be secured without converting the program into the corresponding secure report. We included encryption algorithms into the MPICH2 library so that data in secret of MPI applications could be readily preserved without require modifying the source codes of the MPI applications. This system use Sandia Micro Benchmark and Intel MPI Benchmarks to evaluate and compared the performance of original MPICH2 and Enhanced Security MPICH2. According to the performance estimation, ES-MPICH2 provides protected Message Passing Interface by give up sensible system performance.

***Index Terms*— Secret key, Encryption, MPI, Parallel Computing, Cryptosystem**

I. INTRODUCTION

In unclustered networks, the data encryption for large scale distributed clusters becomes a non trivial and challenging problem, due to the open accessible nature of the internet. Information processed in a distributed cluster is shared among a group of distributed processes or users by virtue of Message Passing protocols (e.g. Message Passing Interface -MPI) running on the internet. To combine the portability with high performance the ES-MPICH2 with the original MPICH2 version is used for incurring the overhead by the confidentiality services. Due to high performance clusters, the security overhead can be reduced in ES-MPICH2. To preserve the data confidentiality, the encryption algorithm can be integrated into the MPICH2 library. To encode messages using Advanced Encryption Standard (AES), Triple Data Encryption Standard (3DES) and Elliptic Curve Cryptography (ECC) are the three cryptographic techniques used in Message Passing Interface system to provide a data confidentiality for several computing nodes.

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. Numerous scientific and commercial applications running on clusters were developed using the MPI protocol. 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. Thus, data confidentiality of MPI applications can be readily preserved without a need to change the source codes of the MPI applications

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 .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. Our channel-level solution aims at providing message confidentiality in a communication channel that implements the Channel Interface 3 (CH3) in MPICH2

Manuscript received on March 2013.

A.Brillia, with the department of computer science, Dr. Paul's Engineering College, Chennai, India.

D.Jagadiswary, with the Department of computer science, Dr. Paul's Engineering College, Chennai, India.

R.Muthu Venkata Krishnan, with the Department of computer science, Jaya Engineering College, Chennai, India.

B. Contributions

The three major contributions of this study includes

- 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 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 MPICHI2 library. To demonstrate this feature, we implemented the AES, 3DES and ECC algorithms in ESMPICH2. We also show in this paper how to add other cryptographic algorithms into the ES-MPICH2 framework.

II. RELATED WORK

Due to an increasing number of commodity clusters connected to each other by public networks, the encrypting and decrypting messages sent and received among computing nodes are not efficient and the data confidentiality is not readily preserved. So to implement secure applications, the programmers may rely on external libraries (e.g. SEAL [26] and NEXUX [11]).There is a minimal changes to the interface to write the secure MPI applications and the calculation time and memory needs for larger key sizes are more in the popular asymmetric cryptosystems like RSA.

III. ENHANCED SECURITY – MPICH2

To offer data confidentiality for secure network communications in message passing environments, a standard MPI mechanism called ES-MPICH2 was introduced. This 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 any additional code for data-confidentiality protection. We seek a channel-level solution in which encryption and decryption functions are included into the MPICH2 library. Thus the ES-MPICH2 maintains a standard MPI interface to exchange messages while preserving data confidentiality. ES-MPICH2 framework provides a secured configuration file that enables application programmers to selectively choose any cryptographic algorithm. It provides easy and full control of security services. AES, 3DES and ECC algorithms are used in ESMPICH2.

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 ciphertext 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.

B. Design structure of ES- MPICH2

One of the objectives in MPICH2 design is portability. To facilitate porting MPICH2 from one platform to another, MPICH2 uses ADI3 (the third generation of the Abstract Device Interface) to provide a portability layer. ADI3 is a full-featured abstract device interface and has many functions, so it is not a trivial task to implement all of them. To reduce the porting effort, MPICH2 introduces the CH3 interface. CH3 is a layer that implements the ADI3 functions, and provides an interface consisting of only a dozen functions. A "channel" implements the CH3 interface. Channels exist for different communication architectures such as TCP sockets, SHMEM, etc. Because there are only a dozen functions associated with each channel interface, it is easier to implement a channel than the ADI3 device.

The hierarchical structure of MPICH2, as shown in Figure1, gives much flexibility to implementers. The three interfaces (ADI3, CH3, and RDMA Channel Interface) provide different trade-offs between communication performance and ease of porting. As a successor of MPICH, MPICH2 [1] aims to support not only the MPI-1 standard, but also functionalities such as dynamic process management, one-sided communication and MPI I/O, which are specified in the MPI-2 standard.

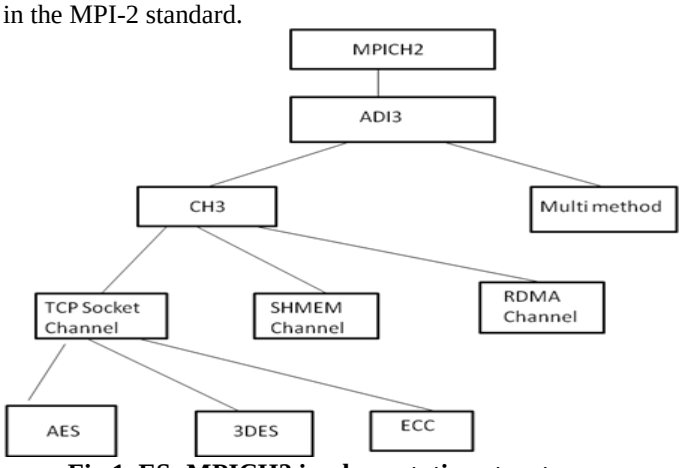

Fig 1. ES- MPICH2 implementation structure

However, MPICH2 is not merely MPICH with MPI-2 extensions. It is based on a completely new design, aiming to provide more performance, flexibility and portability than the original MPICH2. The future development for MPICH, including those necessary to accommodate extensions to the MPI Standard now being contemplated by the MPI Forum.

The process of creating a standard to enable portability of message-passing applications codes began at a workshop on Message Passing Standardization and the Message Passing Interface (MPI). 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.

IV. IMPLEMENTATION DETAILS

During the implementation, the system involves Key Agreement, Advanced Encryption Standard, Triple Data Encryption Standard, Elliptic Curve Cryptography, File Chatting and File Sharing. Figure 2 depicts the implementation structure of ES-MPICH2, where a cryptosystem is implemented in the TCP socket layer. Thus the messages are encrypted and decrypted in the TCP socket channel rather than the ADI3 and CH3 layers. We addressed

1. In which layer should we implement cryptographic algorithms?
2. Which cryptosystem should we choose to implement?
3. How to implement secure key management?

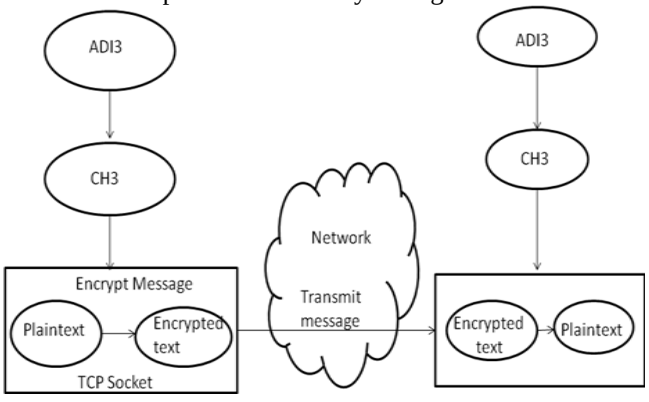

Fig 2. Implementation structure of ES-MPICH2 in which the cryptosystem is implemented in the TCP socket layer for complete transparency.

A. Key Agreement

In Key Agreement, the Diffie–Hellman key exchange algorithm is used that establishes a shared secret that can be used for secret communications by exchanging data over a public network. Figure 3 presents the key agreement infrastructure using Arithmetic XOR operations (e.g. 1 0→ 1). The terminal A and terminal B can share their secret keys by using hash functions. The positive response (e.g. 1) can be saved in the source side. The positive acknowledgement can be stored in the destination side. Thus the secret communications is done by accepting the secret key in the destination side and if it is matched send the acknowledgement to the destination side.

B. Advanced Encryption Standard

AES is based on a design principle known as a substitution-permutation network, and is fast in both software and hardware. Unlike its predecessor DES, AES does not use a Feistel network. AES is a variant of Rijndael which has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. By contrast, the Rijndael specification *per se* is specified with block and key sizes that may be any multiple of 32 bits, both with a minimum of 128 and a maximum of 256 bits.

AES operates on a 4×4 column-major order matrix of bytes, termed the *state*, although some versions of Rijndael have a larger block size and have additional columns in the state. Most AES calculations are done in a special finite field.

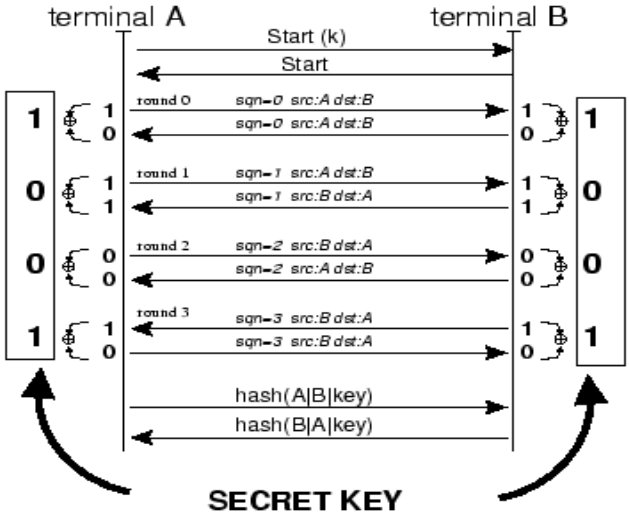

Fig 3 Key Agreement using XOR operations for sharing secret keys.

The key size used for an AES cipher specifies the number of repetitions of transformation rounds that convert the input, called the plaintext, into the final output, called the cipher text. The numbers of cycles of repetition are as follows:

- 10 cycles of repetition for 128 bit keys.
- 12 cycles of repetition for 192 bit keys.
- 14 cycles of repetition for 256 bit keys.

Each round consists of several processing steps, including one that depends on the encryption key itself. A set of reverse rounds are applied to transform cipher text back into the original plaintext using the same encryption key.

Figure 4 depicts the internal operations of Advanced Encryption Standard in which each round consists of four special functions. They are

- Byte Substitution
- Permutation
- Arithmetic operations over a finite field
- XOR with a key

These transformations are applied to a 128-bit input block in a certain sequence to perform an AES encryption or decryption. In both cases, the transformations are grouped to so-called rounds. There are three different types of rounds, namely, the initial round, the normal round, and the final round. AES is a symmetric block cipher. It acts as a resistance against all known attacks.

In a wide range of platforms, the AES has speed and code compactness. The key unit is used to store keys and to calculate the key expansion function. Due to the fact that the AES is standardized for 128, 192, and 256-bit keys, the interface between the key unit and the data unit is designed for the key expansion for several different key sizes can be implemented on the same chip. It is the preferred algorithm for implementations of cryptographic protocols.

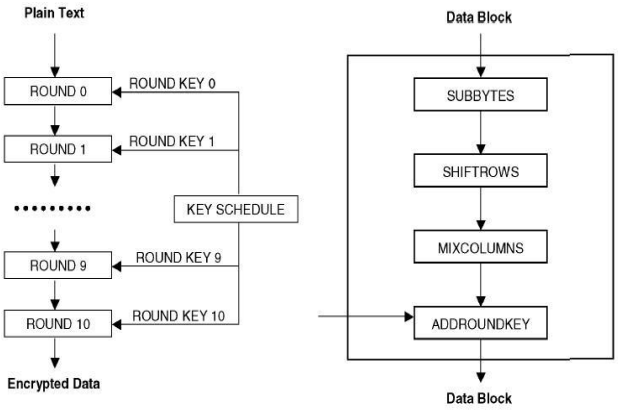

Fig 4 Internal operations of Advanced Encryption Standard.

C. *Triple Data Encryption Standard*

The original DES cipher's key size of 56 bits was generally sufficient when that algorithm was designed, but the availability of increasing computational power made brute-force attacks feasible. Triple DES provides a relatively simple method of increasing the key size of DES to protect against such attacks, without the need to design a completely new block cipher algorithm. Triple DES uses a "key bundle" which comprises three DES keys, K_1, K_2 and K_3, each of 56 bits (excluding parity bits). Figure 5 depicts the internal operation of Triple Data Encryption Standard in which it has three keys (56 x 3 = 168 bits) for encryption and decryption.

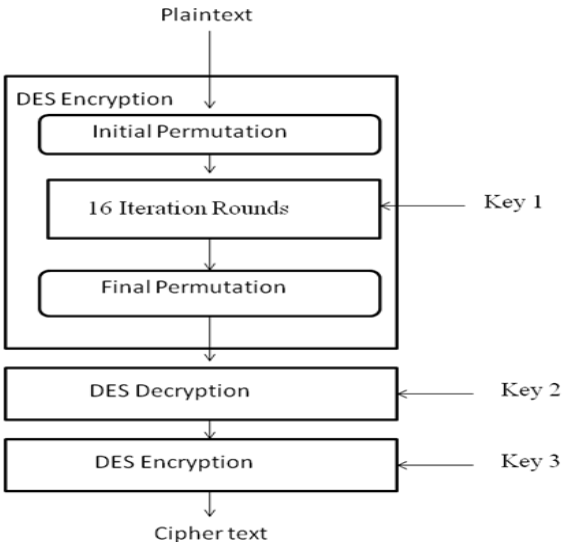

Fig 5 Internal operation of Triple Data Encryption Standard.

The encryption algorithm is:
Cipher text = $E_{K3}(D_{K2}(E_{K1}(\text{plaintext})))$
i.e., DES encrypts with K_1, DES *decrypt* with K_2, then DES encrypt with K_3.Decryption is the reverse:
Plaintext = $D_{K1}(E_{K2}(D_{K3}(\text{cipher text})))$
i.e., decrypt with K_3, *encrypt* with K_2, and then decrypt with K_1.

D. *Elliptic Curve Cryptography*

Elliptic curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. Popular asymmetric cryptosystems like RSA are known to be very costly concerning calculation time and memory needs for larger key sizes. To solve this problem, elliptic curve cryptosystems based can be used.

An elliptic curve cryptosystem is an asymmetric cryptosystem relying on the hardness of the discrete logarithm problem in elliptic curve groups With ECIES encryption, it is possible to encrypt data with a 160-bit key as secure as with RSA using a 1024-bit key. So ECIES encryption is the better choice compared to RSA when calculation time and available memory are restricted, e.g. when using cryptosystems on smartcards

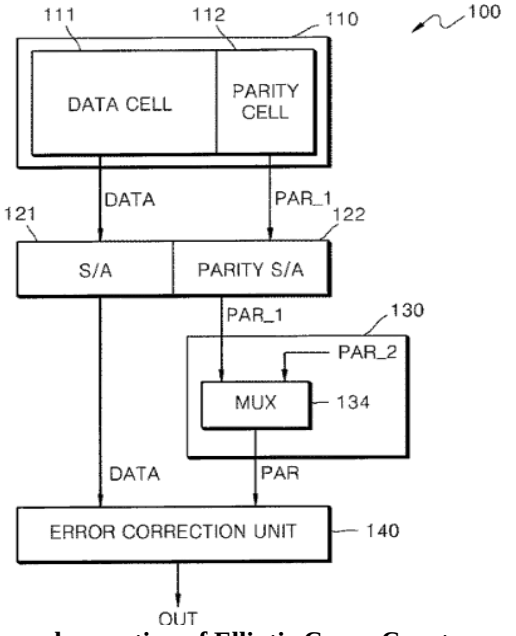

Fig 6 Internal operation of Elliptic Curve Cryptography

The Elliptic Curve Integrated Encryption Scheme (ECIES), also known as Elliptic Curve Augmented Encryption Scheme or simply the Elliptic Curve Encryption Scheme.

Figure 6 depicts the internal operations of Elliptic Curve Cryptography in which two parity cells are added and compared and if it is accepted then no error is found if it is not accepted then the error is detected by the Error Correction Unit.

V. BENCHMARKS FOR POINT TO POINT OPERATIONS

In this section we present some of the simplest benchmarks for performance of MPICH on various platforms. The performance test programs mpptest and goptest can produce a wealth of information; the script basetest, provided with the MPICH implementation, can be used to get a more complete picture of the behavior of a particular system. Here, we present only the most basic data: short- and long-message performance.

For the short-message graphs, the only options used with mpptest are -auto and –size 0 1000 40. The option -auto tells mpptest to choose the sizes of the messages so as to reveal the exact message size where there is any sudden change in behavior (for example, at an internal packet-size boundary). The -size option selects messages with sizes from 0 to 1000 bytes in increments of 40 bytes. The short-message graphs give a good picture of the latency of message passing.

For the long-message graphs, a few more options are used. Some make the test runs more efficient. The size range of message is set with -size 1000 77000 4000, which selects messages of sizes between about 1K and 80K, sampled every 4000 bytes. These tests provide a picture of the best achievable bandwidth performance. More realistic tests can be performed by using -cachesize (to force the use of different data areas), -overlap (for communication and computation overlap), -async (for nonblocking communications), and -vector (for noncontiguous communication). Using -givedy gives information on the range of performance, displaying both the mean and worst-case performance.

VI. PERFORMANCE PROBLEMS

One common problem with simple performance measurement programs is that the results are different each time the program is run, even on the same system. A number of factors are responsible, ranging from assuming that the clock calls have no cost and infinite resolution to the effects of other jobs running on the same machine. A good performance test will give the same (to the clock's precision) answer each time. The mpptest and goptest programs distributed with MPICH compute the average time for a number of iterations of an operation (thus handling the cost and granularity of the clock) and then run the same test over several times and take the minimum of those times (thus reducing the effects of other jobs). The programs can also provide information about the mean and worst-case performance. More subtle are issues of which test to run. The simplest "ping-pong" test, which sends the same data (using the same data buffer) between two processes, allows data to reside.

VIII. CONCLUSIONS AND FUTURE WORKS

For efficient implementation of ES-MPICH2, it is important for the point multiplication algorithm and the underlying field arithmetic to be efficient. There are different methods for efficient implementations like AES, 3DES and ECC suited for different software configurations. Implementation of ECC using projective coordinates has shown considerable improvement in efficiency compared to the affine coordinate implementation. This improvement in efficiency is due to the key size and resistance against attacks and run much faster than the modular reduction in prime field.

The current version of ES-MPICH2 is focused on securing the transmission control protocol (TCP) connections on the internet, because we addressed the data confidentiality issues on geographically distributed cluster computing systems. In addition to the MPI library, other parallel programming libraries will be investigated. Candidate libraries include the shared memory access library and the remote direct memory access library. We plan to provide confidentiality services in the SHMEM and RDMA libraries. A third promising direction for further work is to integrate encryption and decryption algorithms in other communication channels like SHMEM and InfiniBand in MPICH2 because an increasing number of commodity clusters are built using standalone and advanced networks such as Infiniband and Myrinet. So far, our study has been restricted to a fairly small platform which consists of 8 nodes. In the future, we plan to use larger clusters to study various aspects of our designs regarding scalability. Another direction we are currently pursuing is to provide support for MPI-2 functionalities such as one-sided communication using RDMA and atomic operations in InfiniBand. We are also working on how to support efficient collective communication on top of InfiniBand.

REFERENCES

1. Darrel Hankerson, Julio Lopez Hernandez, Alfred Menezes, Software Implementation of Elliptic Curve Cryptography over Binary Fields, 2000.
2. M. Brown, D. Hankerson, J. Lopez, A. Menezes, Software Implementation of the NIST Elliptic Curves Over Prime Fields, 2001.
3. Certicom, Standards for Efficient Cryptography, SEC 1: Elliptic Curve Cryptography, Version 1.0, September 2000
4. Certicom, Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 1.0, September 2000.
5. Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996
6. D.E. Denning, "Secure Personal Computing in an Insecure Network," Comm. ACM, vol. 22, no. 8, pp. 476-482, 1979.
7. J.J. Dongarra, S.W. Otto, M. Snir, and D. Walker, "An
8. Introduction to the Mpi Standard," technical report, Knoxville, TN, 1995.
9. W. Ehrsam, S. Matyas, C. Meyer, and W. Tuchman, "A Cryptographic Key Management Scheme for Implementing the Data Encryption Standard," IBM Systems J., vol. 17, no. 2, pp. 106- 125, 1978.
10. I.F. Blake, G. Seroussi, and N.P. Smart, Elliptic Curves in Cryptography. Cambridge Univ. Press, 1999.
11.] I. Foster, N.T. Karonis, C. Kesselman, G. Koenig, and S. Tuecke, "A Secure Communications Infrastructure for High-Performance Distributed Computing," Proc. IEEE Sixth Symp. High Performance Distributed Computing, pp. 125-136, 1996
12. A. Geist, W. Gropp, S. Huss-Lederman, A. Lumsdaine, E.L. Lusk, W. Saphir, T. Skjellum, and M. Snir, "Mpi-2: Extending the Message-Passing Interface," Proc. Second Int'l Euro-Par Conf. Parallel Processing (Euro-Par '96), pp. 128-135, 1996.
13. R. Grabner, F. Mietke, and W. Rehm, "Implementing an mpich-2 Channel Device over Vapi on Infiniband," Proc. 18th Int'l Parallel and Distributed Processing Symp., p. 184, Apr. 2004.
14. W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A High- Performance, Portable Implementation of the Mpi Message Passing Interface Standard," Parallel Computing, vol. 22, no. 6, pp. 789-828, 1996.
15. P. Hamalainen, M. Hannikainen, T. Hamalainen, and J. Saarinen, "Configurable Hardware Implementation of Triple-des Encryption Algorithm for Wireless Local Area Network," Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing (ICASSP '01), pp. 1221-1224, 2001.
16. G.A. Koenig, X. Meng, A.J. Lee, M. Treaster, N. Kiyanclar, and W. Yurcik, "Cluster Security with Nvisioncc: Process Monitoring by Leveraging Emergent Properties," Proc. IEEE Int'l Symp. Cluster Computing and Grid (CCGrid '05), 2005.
17. M. Lee and E.J. Kim, "A Comprehensive Framework for Enhancing Security in Infiniband Architecture," IEEE Trans. Parallel Distributed Systems, vol. 18, no. 10, pp. 1393-1406, Oct. 2007.
18. 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," Proc. 18th Int
19. Greg Burns, Raja Daoud, and James Vaigl. LAM: An open cluster environment for MPI. In John W. Ross, editor, Proceedings of Supercomputing Symposium '94, pages 379–386. University of Toronto, 1994.