Development of a Distributed Cryptographic System Employing Parallelized Counter Mode Encryption

Dev Dua, Sujala D. Shetty, Lekha R.Nair
Department of Computer Science
BITS Pilani, Dubai U.A.E.
Emails: {devdua@live.com, sujala@dubai.bits-pilani.ac.in, lekharnair@gmail.com}

***Abstract*— In the current era of Cloud Computing, Big Data and always-connected devices and apps, it is critical that the data that reside on networks be secured by robust techniques and protocols. As the data are highly distributed and voluminous, the security measures should employ methods that efficiently and rapidly encrypt data. This calls for scaling the model up to employ a distributed programming paradigm, so that the utilization of resources (computing and storage) on the network is high and channeled for processing in an optimum way. Exploring the idea of distributed cryptography might hold solutions to address this potential problem. We have tried to probe in this direction by building a distributed and flexible cryptosystem that employs a parallelized version of the AES algorithm. The results obtained emphasize the performance gains, agility, flexibility and scalability of the concept of cryptography using distributed systems.**

Keywords-Big Data; Cluster Computing; Distributed Systems; Security; Cryptography; MPI.

I. INTRODUCTION

In the past few years, the concept of Cloud Computing [13] has gained a lot of traction, and is seeing a high adoption rate. This concept, wherein a third party provides networked hardware, computing platforms and application software as a service accessible via the Internet to its customers is more commonly given the generic term of "cloud". In most cases the exact location of data stored is unknown to the customers which raises concerns over the privacy of the data on the external network. Security of data and their integrity is a crucial concern that most organizations have when moving their data to a cloud. Since cloud computing is relatively a recent trend, encryption in the cloud is still in its early stages. Only a few cloud providers one of them being Microsoft Azure provision encryption of data stored in their data centers [12]. The primary objective of encryption in the cloud is to deter unauthorized access, as access to sensitive data (without the knowledge of the owner) by non-permitted entities is a privacy violation. The focus of this paper is on how encryption can be adapted to utilize the virtually infinite amount of resources that the cloud provides, and not Key Management, which is another crucial aspect of a cryptosystem.

The cloud model being considered here is a private/hybrid cloud. The public cloud model provides services and infrastructure over the Internet, from a location that is external to the customers' computing environment. Since resources are shared in this model, this commonality of storage/processing space makes them more vulnerable to data protection issues than private clouds. Private clouds are basically the same in infrastructure and software as public clouds, and they differ in the aspect that the network that powers and interconnects the systems in the private cloud is exclusive to the organization that houses the cloud. Only authorized users belonging to the organization domain can make use of the private cloud. Hybrid clouds provide a balance between the other two models, as one can even combine services provided by different cloud providers to keep each aspect of the organization's operation process efficient and streamlined. However, one of the drawbacks is the hassle to orchestrate security platforms of all providers to play with each other. Hence, it is recommended to keep critical data safe in the private side of the hybrid cloud, and use strong security policies to protect the data in-house.

Thus, this paper assumes that the data to be kept safe in the data storage facilities has already been transmitted there using a Secure Sockets Layer encrypted connection or a connection protected by Transport Layer Security. The cryptosystem aims to operate as soon as the data reaches the cloud, so that the window between the unencrypted and encrypted states of the data is as small as possible.

Parallelized encryption methods have been explored in the past [6][10][11] and researchers have implemented parallel encryption/decryption algorithms on single machines with multi-core CPUs. This leads to an overall speedup on the time spent on encryption/decryption on such machines when compared to single process, sequential execution of the cryptography technique. However, the parallelizable algorithms can be scaled to the next level, by allowing them to run on distributed systems, so that the number of processes employed during the computation can be drastically increased. This would allow for noticeably faster and efficient encryption and decryption of voluminous, stagnant files. Since the data being encrypted will be distributed in segments over a net-work, it has a higher level of security by virtue of the randomness of its location.

The objective of this paper is to meet the following:

- Build and setup a distributed homogenous computing cluster with 3 compute nodes
- Create a native cryptographic platform that makes use of the nodes to encrypt/decrypt files in a parallelized manner.

 ISBN: 978-1-61208-412-1

- Analyse the performance of the parallelized encryption and decryption code on the cluster by varying the number of slave processes operating on varying file sizes.
- Further explore possible extensions to the project aims to efficiently ensure security of the platform by deploying user access control, more control in terms of options and features to encrypt data, and scheduling regular refreshes to the encrypted state.

II. MPI AND PARALLELIZATION OF CODE

A. Message Passing and Message Passing Interface

Message Passing [1][3][4] is one of the types of inter-process communication, which is employed in object oriented programming and parallel programming. Message passing involves sending functions, signals and data packets from one process to either all other processes or a specific process executing the Message Passing Interface (MPI) job. The invoking program sends a "message" to the object. This standard contrasts from routine programming [2] wherein a method, process or a subroutine is specifically called by name. One of code libraries that have been consented by the MPI Forum is the Message Passing Interface Standard (MPI) [3], and is widely supported by users, software library developers, researchers and even vendors. The Message Passing Interface was built to develop a flexible, standard and portable message passing standard to be extensively used in programs that employ message passing.

B. Parallelization of the AES algorithm

To adapt the cryptosystem to make full use of the cluster, the most apt cryptography mode had to be determined so that the cipher can optimally encrypt/decrypt data. The Counter (CTR) mode [9] was chosen, due to the reason that unlike traditional encryption modes, the CTR mode encrypts data by converting the underlying block cipher into a stream cipher. It generates subsequent key stream blocks by encrypting successive values of a counter, which can be any sequence generating function that produces a sequence which is guaranteed not to repeat for a long time. The most commonly used counter, due to its popularity and simplicity is a simple increment-by-one counter.

The CTR mode encrypts blocks of data in a way that encryption of a block is totally independent of the encryption done on the previous block. This enables efficient use of hardware level features like Single Instruction Multiple Data (SIMD) [14] instructions, large number of registers, dispatch of multiple instructions on every clock cycle and aggressive pipelining. This makes CTR mode encryption to perform much better and faster than a CBC (Cipher Block Chaining) mode encryption. Using the CTR mode, encryption can be easily parallelized as each block can be independently encrypted, thus using the entire power of the underlying hardware.

After the plaintext is received by the cipher, encryption can then be carried out by using the pre-computed number sequence (if pre-processing of Initialization Vectors is used). This can lead to massive throughput, of the order Gigabits per second. As a result, random access encryption is possible using this mode, and is particularly useful for encryption of Big Data, where the plaintext are chunky blocks of huge amounts of data. A unique characteristic of the CTR mode is that the decryption algorithm need not be designed, as the decryption has the same procedure as encryption, differing only in terms of initialization of the Initialization Vector (IV). This fact holds most weightage in case of ciphers such as Advanced Encryption Standard (AES), used in this project, because the forward "direction" of encryption is substantially different from traditional inverse mapped "direction" of decryption, which has no effect on the CTR mode whatsoever. An additional benefit of the CTR mode is the lack of need to implement key scheduling. All these factors contribute to a significant yield in hardware throughput, by making the design and implementation simpler.

III. PROPOSED DESIGN FOR THE CRYPTOSYSTEM

The CTR mode of cryptography has similar characteristics to Output Feedback (OFB) [8], but also allows a random access property during decryption, which is quite suitable for this project. CTR mode is well suited to operate on a multi-processor machine where blocks can be encrypted in parallel. However, since the data and processing in a cluster is distributed, the way in which distributed encryption can be achieved needs to be determined. One way in which the CTR mode is easily parallelizable across nodes is by letting each process have its own counter, and by virtue of this, its own Initialization Vector (IV). The approach in our case is different compared to [6] in terms of the extent to which the processing power has been utilized. However the file splitting is somewhat similar in approach.

The IVs and the counters should be initialized at runtime, depending on the number of processes attached to the MPI job, which is accessed using MPI_COMM_SIZE. The file to be encrypted is read in parallel as *n* equal and distinct parts, where n is equal to MPI_COMM_SIZE. Each process, after reading the portion of the file that has been assigned to it, encrypts it by passing substrings of size AES_BLOCK_SIZE to the AES_ctr128 method of the OpenSSL library, and writes the encrypted data to a file that contains the block of data encrypted by that process. Thus, after encryption, *n* encrypted blocks along with *n* IVs are created, which reside in the machine that generated them.

A. Setup of the experimental environment

To implement and test the cryptosystem, a Beowulf cluster [5] built using 3 compute nodes, which are regularly

 ISBN: 978-1-61208-412-1

indistinguishable, was setup and connected in a 100M network over password-less SSH [7] and libraries and frameworks required by the cryptosystem like unison, libcrypto, openssl-dev and OpenMPI 1.6 were installed. Each compute node in the Beowulf Cluster created to collect results had the following specifications:

CPU : Intel Core i7 4770 (4 cores X 3.40 Ghz)
Memory : 8GB
Operating System : Ubuntu 14.04 LTS

BASH scripts were used to run the scripts used to control the cluster. All programs that were run on the MPI ORTE were written in C. OpenMPI was used to provide a parallel environment to the code. Unison was used to synchronize the source code and executables across the compute nodes. The 128bit CTR implementation of the AES algorithm available in the OpenSSL library was used in the cryptosystem.

For encryption, the file path and the number of processes are passed as command-line arguments to the cryptosystem. The script then synchronizes the file among all available nodes, so that a copy of the file under encryption is available locally on the HDD of every node. This is done to minimize I/O overhead, and the network overhead involved in this approach is minimal compared to the performance gains expected. The master process evaluates the file and pads the end of the file with bytes if the file size is not a multiple of the number of processes specified for use. Each node then operates on the file with 4 processes per node, with each process generating an encrypted block and iv as separate files. The display names of the files themselves are generated in a way that the file names of related encrypted blocks (blocks generated from the same unencrypted file) seem totally unrelated to each other.

During decryption, the cryptosystem recognizes the blocks to be decrypted by means of the file name passed via command line arguments and then attempts to decrypt those blocks. Decryption succeeds if the number of processes specified matches the number of processes that were used to encrypt the file. The master process then accumulates the decrypted blocks and creates a decrypted file with the padding removed (if any).

B. Advantages of the cryptosystem and Impact on security

While thinking of the above design, there were concerns about the security and integrity of the approach. The advantages outweigh the issues, which can be improved upon as explained in Section V.

- Counters in the CTR mode are usually limited to the value 2^{64}-1, which then has to be reset once the counter of the code reaches this value to avoid overflow. However, in the proposed design, each process has a counter of its own with the above limit, so an increase in the number of processes that are running the engine simultaneously linearly increases the amount of data that can be encrypted/decrypted. Thus with *n* processes, the amount of data that can be encrypted/decrypted increases *n* fold.
- Data encrypted by *n* processes can be decrypted if and only if *n* processes are used to decrypt it. Any mismatch in the number of processes will lead to an Exception or files with junk characters. This ensures that the data is undisclosed and confidential as only the party that encrypts the data would know the number of processes that were chosen to encrypt the data. This adds an extra security parameter apart from the secret key. Brute-forcing will take longer as well, as the master key is 10 bytes long (8 bytes for the AES key and 2 bytes for the number of processes).
- Since process ranks are arbitrarily decided at runtime, the splitting and transmission of data that occurs during encryption is completely random. Thus, data being encrypted is spread across the compute nodes, which increases entropy of location of the data. One cannot determine with surety the exact location of every block being generated by the cryptosystem.
- As the blocks being generated are much smaller than the original unencrypted file, the network overhead is also minimized.
- The system built is highly portable and extensible, with very less setup involved. This scalability makes the cryptosystem highly compatible with the scaling capabilities offered by the cloud environment it is hosted on.
- Since OpenSSL is being at the very core of the cryptosystem, the security of the entire system in general is enforced. The core crypto library of OpenSSL provides basic cryptographic functions that are highly supported by the network security community and provides an abundantly accepted set of encryption implementations. Any vulnerabilities in the library are updated fairly often, and thus the cryptosystem is free of core security issues.

C. Description of the test files

The files used to measure the total execution time of the algorithm were generated by using the native *dd* command provided by Linux, and contained randomized data with the text file having a fixed sizes. A random stream was generated by Linux, which was subsequently captured by the *dd* command till the stream filled in the file of the specified size. The sizes of the aforementioned files were 100 KB, 200 KB, 500 KB, 1MB, 2.5MB, 5MB, 10MB, 20MB, 40MB, 50MB, 100MB, 500MB, and 1GB.

 ISBN: 978-1-61208-412-1

IV. PERFORMANCE ANALYSIS

As can be seen from Figure 1, the most inefficient use of the cluster is in the first case where the number of processes (nP) is 1. It can be safely assumed that the time taken by the cluster to execute parallel code using just 1 process will be close to executing a serial version of the code on a uniprocessor. Also, as the file size increases, there is a sharp increase in execution time, which is largely due to high process idle times. Hence it is meaningless to use just 1 process to encrypt a big file. On the other end, the most efficient use of the cluster is when it is used to maximum capacity, i.e., 12 processes. A maximum performance increase of 6.7x is obtained when the cluster is optimally used, in the case of encryption of the 1GB file. The throughput obtained here is 4.233 Gbps. It can also be seen from the graph as the number of processes allocated to the encryption sequence is increased, the time vs. size graph tends to become flatter and linear.

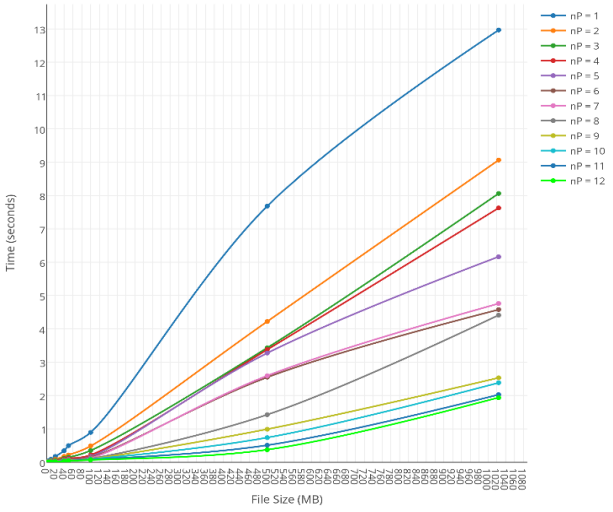

Figure 1. Execution time (s) vs. File size (MB) for encryption

The similarity between Figure 1 and Figure 2 is apparent, courtesy of the nature of CTR mode cryptography. Most of the code in the decryption process is the same as the encryption process, and only slight variations in execution times are observed. The performance speedup obtained upon operation on the 1 GB file by using 12 processes in this case is only 6.378x compared to 6.7x in the case of encryption. This is solely due to the conglomeration operation at the end of the process, which consumes time in stitching the decrypted blocks together. Another pattern observed is that the curves obtained when the number of processes are increased tend to group together, indicating that O(n) times are possible as the number of processes goes up.

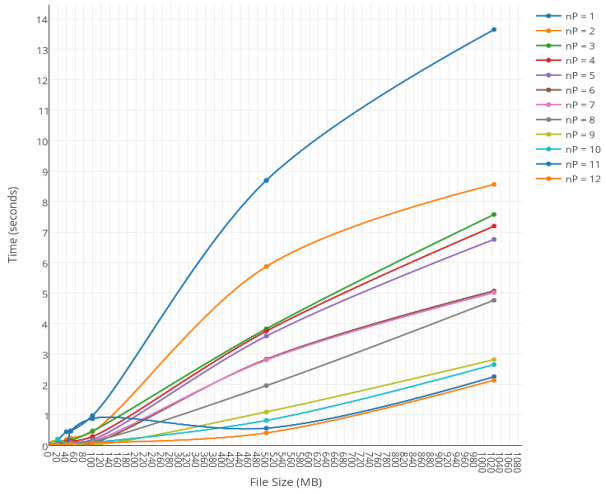

Figure 2. Execution time (s) vs. File size (MB) for decryption

V. POSSIBLE IMPROVEMENTS TO THE CRYPTOSYSTEM

Several hardware level and implementation improvements can be made to the cryptosystem as the one designed here is not production ready, and is a prototype to validate the design concepts. Some improvements include:

- Using an SSD array for storage of the files would lead to drastically improved performance, as the I/O capabilities of SSDs are much better compared to conventional HDDs
- The processors used for testing were consumer machine grade, and not high performance processors typically used by server farms. Hence, better processors could be used
- The network on which the system was tested was a 100M LAN, however the network can be upgraded to a Gigabit connection and a fast Switch can be used exclusively for the compute nodes in the cluster
- The Scatter/Gather pattern of communicative computation provided by Open can be employed to

 ISBN: 978-1-61208-412-1

ensure robustness of the system, and allow for better handling of processes to avoid process idle times

- In-memory storage can be used to store the files and restrict hard drive reads and writes. This can lead to significant improvements in execution times, and reduce I/O overhead
- Reduction in network data transmission and file generation to disclose as little data as possible, as well as avoid chances of snooping and eavesdropping

VI. CONCLUSION

Clusters are easy to setup using OpenMPI. However, the network being used to connect the compute nodes has to be reliable and sufficiently fast to reduce the communication time. The security of the platform can be further increased by scheduling encryption at periodic intervals, in a real world scenario, so that chances of attacks are less, and the safety of data is uncompromised. The computing power of the cloud can also be harnessed to create virtualized clusters, with a fixed number of nodes, but scalable in terms of configuration of the nodes. So, this could result in a low performing cluster with 3 compute nodes with just 3 logical processors and 3GB of RAM, to a high performance 3 node cluster with 48 logical processors and 168 GB of RAM. Furthermore, the cluster can be connected to a REST Web API that can accept files to be encrypted, run encryption on them, and return a link to the encrypted versions.

This shows the flexibility of the platform designed in this project, and also proves the ease by which a cluster can be created in just a few hours. Also, since the tools used are open source and are industry leading solutions with a lot of community support, the maintenance of the cluster is minimal. This makes the cluster fault tolerant in a way, as well as highly extensible.

REFERENCES

[1] Goldberg, Adele, David Robson (1989). *Smalltalk-80 The Language*. Addison Wesley. pp. 5–16. ISBN 0-201-13688-0.

[2] Orfali, Robert (1996). *The Essential Client/Server Survival Guide*. New York: Wiley Computer Publishing. pp. 1–22. ISBN 0-471-15325-7.

[3] Blaise Barney, Lawrence Livermore National Laboratory, *Message Passing Interface*, URL : https://computing.llnl.gov/tutorials/mpi/, Retrieved: October 2014

[4] Extreme Science and Engineering Discovery Environment (XSEDE) Project, National Center for Supercomputing Applications (NCSA), *Introduction to MPI*, University of Illinois at Urbana-Champaign. URL: https://www.xsede.org/high-performance-computing, Retrieved: November 2014

[5] Donald J Becker, Thomas Sterling, Daniel Savarese, Johne E Dorban,; Udaya A Ranawak and Charles V Packer, "*BEOWULF: A parallel workstation for scientific computation"*, in Proceedings, International Conference on Parallel Processing vol. 95, (1995). URL: http://www.phy.duke.edu/~rgb/brahma/Resources/beowulf/papers/ICPP95/icpp95.html, Retrieved: October 2014

[6] Ozgur Pekcagliyan and Nurdan Saran, *Parallelism of AES Algorithm via MPI*, 6th MTS Seminar, Cankaya university, April 2013 URL: http://zgrw.org/files/mpi_AES.pdf, Retrieved: November 2014

[7] Hortonworks, *HOWTO: Generating SSH Keys for Passwordless Login*, URL: http://hortonworks.com/kb/generating-ssh-keys-for-passwordless-login/, Retrieved: October 2014

[8] ISO/IEC 10116:2006 - Information technology -- Security techniques – *"Modes of operation for an n-bit block cipher"*, URL: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38761, Retrieved: December 2014

[9] Helger Lipmaa, Phillip Rogaway, Chiang Mai and David Wagner, Helsinki University of Technology (Finland) and University of California at Davis (USA) and University of Tartu (Estonia) University (Thailand), University of California Berkeley (USA), *"CTR-Mode Encryption"*, URL: http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/workshop1/papers/lipmaa-ctr.pdf, Retrieved: December 2014

[10] Deguang Le, Jinyi Chang, Xingdou Gou, Ankang Zhang and Conglan Lu, *"Parallel AES algorithm for fast Data Encryption on GPU"*, 2nd International Conference on Computer Engineering and Technology (ICCET), 16-18 April 2010 vol.6, no., pp.V6-1,V6-6, doi: 10.1109/ICCET.2010.5486259
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5486259&isnumber=5485932, Retrieved: November 2014

[11] M. Nagendra and M. Chandra Sekhar, Department of Computer Science & Technology, Sri Krishnadevaraya University, Anantapuramu, India, "*Performance Improvement of Advanced Encryption Algorithm using Parallel Computation"*, International Journal of Software Engineering and Its Applications Vol.8, No.2 URL: http://www.sersc.org/journals/IJSEIA/vol8_no2_2014/28.pdf, Retrieved: November 2014

[12] Microsoft Azure Trust Center: Security URL: http://azure.microsoft.com/en-us/support/trust-center/security/, Retrieved: October 2014

[13] Mladen A. Vouk, Department of Computer Science, North Carolina State University, Raleigh, North Carolina, USA, *"Cloud Computing – Issues, Research and Implementations"*, Journal of Computing and Information Technology - CIT 16, 2008, 4, 235–246, doi:10.2498/cit.1001391, URL: http://hrcak.srce.hr/file/69202 Retrieved: October 2014

[14] William Gropp, Mathematics and Computer Science Division Argonne National Laboratory Argonne, IL 60439, USA, *"Tutorial on MPI: The Message-Passing Interface"*, URL: https://www.idi.ntnu.no/~elster/tdt4200-f09/gropp-mpi-tutorial.pdf, Retrieved: October 2014

 ISBN: 978-1-61208-412-1