GridCrypt: High Performance Symmetric Key Cryptography using Enterprise Grids

Agus Setiawan, David Adiutama, Julius Liman, Akshay Luther and Rajkumar Buyya

Grid Computing and Distributed Systems Laboratory
Dept. of Computer Science and Software Engineering
The University of Melbourne, Australia

Email: {a.setiawan, d.adiutama, j.liman}@students.cs.mu.oz.au
akshayl@cs.mu.oz.au, raj@cs.mu.oz.au

Abstract

Today's cryptanalysis on symmetric key cryptography is encouraging the use of larger key sizes and complex algorithms to achieve an unbreakable state. However, this leads an increase in computational complexity. This has promoted many researchers to develop high-performance symmetric key cryptography schemes using approaches such as the use of high-end computing hardware. Peer-to-peer (P2P) or enterprise grids are proven as one of the approaches for developing cost-effective high-end computing systems. By utilizing them, one can improve the performance of symmetric key cryptography through parallel execution. This approach makes it attractive for adoption by businesses to secure their documents. In this paper we propose and develop an application for symmetric key cryptography using enterprise grid middleware called Alchemi. An analysis and comparison of its performance is presented along with pointers to future work.

1 Introduction

Symmetric key cryptography, also called private key or secret key cryptography, is a method that uses the same key for encryption of plain text to generate the cipher text and decryption of the cipher text to get the original plain text. This method is used to secure data for transmission over open networks such as the Internet. The use of keys is complemented by the use of advanced algorithms.

There are two methods that are used in symmetric key cryptography [1]: block and stream. The block method divides a large data set into blocks (based on predefined size or the key size), encrypts each block separately and finally combines blocks to produce encrypted data. The stream method encrypts the data as a stream of bits without separating the data into blocks. The stream of bits from the data is encrypted sequentially using some of the results from the previous bit until all the bits in the data are encrypted as a whole.

Although stream ciphers are designed to encrypt data as a whole, we introduce a new modified scheme that divides the data into several blocks prior to processing the data with stream cipher. This allows us to divide the stream cipher process into several processes so that we can apply multiprocessing principles to speed up the stream cipher method. The process of changing stream cipher method into the hybrid scheme method is shown in Figure 1.

In this paper, we consider three popular encryption algorithms. The first two, DES (Data Encryption Standard) and Blowfish, use the block cipher method and the third one, RC4, uses the stream cipher

method. The similarity of these encryption algorithms is that they initialise an S-Box for use in the encryption process derived from the key used for encryption [2]. During decryption, the same key is used to generate an S-Box to decrypt the cipher data.

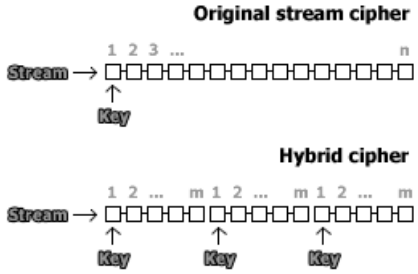

Figure 1. Modified stream cipher

1.1 DES/3DES Encryption

DES was developed at IBM in 1977 and endorsed by the U.S. Department of Defense as an official standard and forms the basis not only for the Automatic Teller Machines (ATM) PIN authentication but a variant is also utilized in UNIX password encryption [5]. DES applies a symmetric 56-bit key to each 64-bit block of data. The process can run in several modes and involves 16 rounds or operations [7].

Due to recent advances in computer technology, some experts no longer consider DES secure against all attacks; since then Triple-DES (3DES) has emerged as a stronger method. Using standard DES encryption, Triple-DES encrypts data three times and uses a different key for at least one of the three passes giving it a cumulative key size of 112-168 bits [5]. If we consider a triple length key to consist of three 56-bit keys K1, K2, K3 then encryption is as follows: encrypt with K1, decrypt with K2, encrypt with K3. Decryption is the reverse process: decrypt with K3, encrypt with K2, decrypt with K1 [6]. Since there is some concern that the encryption algorithm will remain relatively unbreakable, NIST has indicated DES will not be recertified as a standard and submissions for its replacement are being accepted. The next standard will be known as the Advanced Encryption Standard (AES) [7].

1.2 Blowfish Encryption

Blowfish is an encryption algorithm that can be used as a replacement for the DES or IDEA algorithms. It is a symmetric (that is, a secret or private key) 64-bit block cipher that uses a variable-length key, from 32 bits to 448 bits, making it useful for both domestic and exportable use. (The U.S. government forbids the exportation of encryption software using keys larger than 40 bits except in special cases.) Blowfish was designed in 1993 by Bruce Schneier as an alternative to existing encryption algorithms. Designed with 32-bit instruction processors in mind, it is significantly faster than DES. Since its origin, it has been analyzed considerably. Blowfish is unpatented, license-free, and available free for all uses [9].

Blowfish encrypts data in 8-byte blocks. The algorithm consists of two parts: a key-expansion part and a data-encryption part. Key expansion converts a variable-length key of at most 56 bytes (448 bits) into several subkey arrays totaling 4168 bytes. Blowfish has 16 rounds. Each round consists of a key-dependent permutation, and a key and data-dependent substitution. All operations are XORs and additions on 32-bit words. The only additional operations are four indexed array data lookups per round [8].

1.3 RC4 encryption

An alleged implementation of RC4 (Ron's Code #4) was posted September 13, 1994 anonymously on the Internet newsgroup sci.crypt{rc4} without permission or verification from Ron Rivest, co-inventor of the RSA Scheme [10].

RC4 is used in a number of commercial systems like Lotus Notes and Netscape. It is a cipher with a key size of up to 2048 bits (256 bytes), which on the brief examination given it over the past year or so seems to be a relatively fast and strong cipher [5].

RC4 is a shared key stream cipher algorithm, which requires a secure exchange of a shared key that is outside the specification. The algorithm is used identically for encryption and decryption as the data stream is simply XORed with the generated random key sequence. The algorithm is serial as it requires successive exchanges of state entries based on the key sequence. Hence implementations can be very computationally intensive. This encryption algorithm is used by standards such as IEEE 802.11 within WEP (Wireless Encryption Protocol) using a 40 and 128-bit keys [11].

2 Alchemi

Alchemi [12] is a .NET based grid computing framework developed at the University of Melbourne. It is an open source project which provides middleware for creating an enterprise grid computing environment by harnessing Windows machines. Alchemi supports multithreaded parallel operation in a manner similar to threading in Java or C#, but with their execution on distributed resources. The parallelism is realised at thread level and the programmer has to identify functions to be parallelized and implement them in the form of threads. Currently, inter-thread communication is not supported, so threads must be independent.

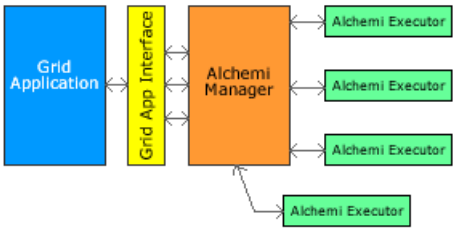

Figure 2. Alchemi's main components

A deployment scheme for Alchemi is shown in Figure 2**.** Its main components are manager and executor that support a master-worker parallel model. Alchemi has a number of features that ease the process of setting up of a grid environment in an enterprise. The executors can be setup in dedicated or non-dedicated mode on employees' desktop computers. In non-dedicated mode, Alchemi has no impact on the workstation as far as the user is concerned. The Alchemi manager also requires a Microsoft SQL Server instance, which is available in most companies.

3 Design and Architecture

Alchemi provides a Software Development Kit (SDK) that can be used by developers to develop grid applications. The SDK includes a Dynamic Link Library (DLL) that supports object oriented programming model for multithreaded applications. Currently, Alchemi only supports completely parallel threads and does not support inter-thread communication. This affects the design of parallelism in our symmetric key cryptography implementation. While this does not affect the parallelization of block cipher method (Section 3.1), we have to use the hybrid scheme discussed above for the stream cipher method.

The architecture of GridCrypt is shown in Figure 3. Our .NET application interacts with Alchemi to enhance the symmetric key cryptography performance. In this application we have developed three main classes. The first class (GridCryptForm) is the interface to control and monitor the progress of the encryption and connection with Alchemi manager. The second class (ConfigurationForm) is the form that can be used to configure the number of threads to be submitted and specify the location of the Alchemi

manager. The last class (GridCryptThread) is the thread class that is run under Alchemi and it uses the algorithm classes.

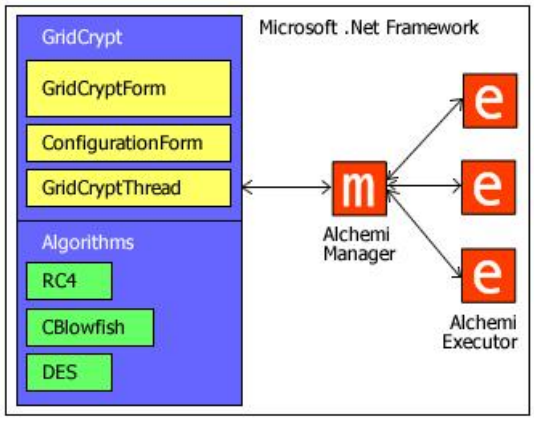

Figure 3. GridCrypt Architecture

3.1 Multithreading Symmetric key Cryptography

In our design of multithreaded symmetric key cryptography, parallelization is being carried to process the data that need to be encrypted. We use the Task-Farming (master-slave) model for execution and principles of SPMD (Single Program Multiple Data) model for application parallelisation. The effect of this parallelization method is that we have to find a way to divide the files into several blocks so that the process can be done in parallel on each block of data.

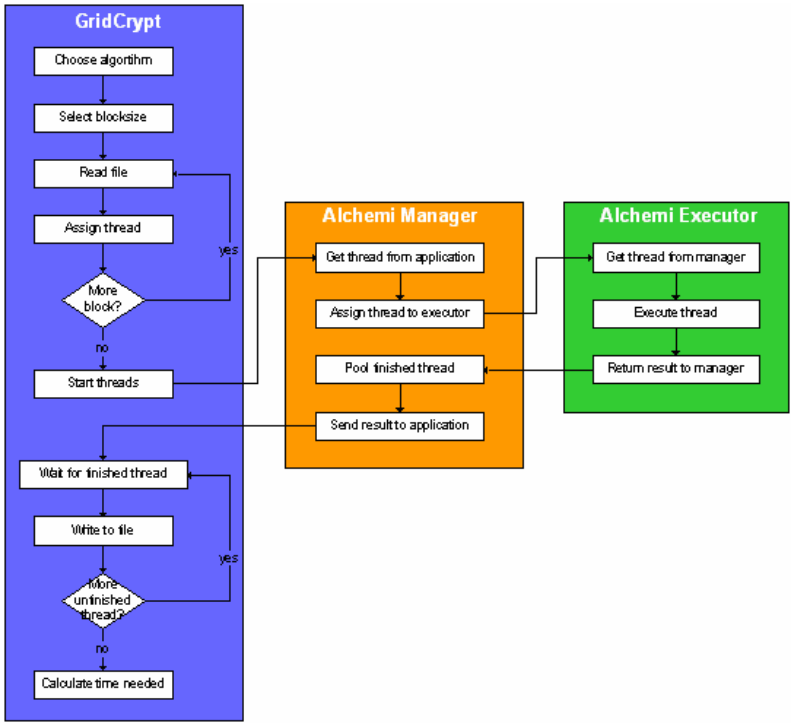

Figure 4. GridCrypt flow of process.

We divide the raw file into several blocks based on the configuration form. An analysis of the three algorithms reveals that the size of the block to be able to run each encryption algorithm should be

divisible by 8. This is because in the DES and Blowfish algorithm the encryption was done 64 bit (8 byte) at a time, so if the block is not divisible by 8 the encryption algorithm that we implement will simply pad the block so that it will be divisible by 8. The padding is carried our on the last block of the file.

The pseudo code of the multithread encryption process is shown in Appendix A. The follow of GridCrypt program is shown in Figure 4. It separates the input file into several parts in order to parallelise the encryption process. After the separation of the input file, each part of the file (block) is assigned to thread including the last block whose size is the remainder of the predefined block size. The file separation process is done by reading the file sequentially according to the block size. After the threads return with the encrypted result, GridCrypt writes the encrypted data to a random access file according to the index that is carried within the thread.

4 Performance Evaluation

The user interface of GridCrypt is shown in Figure 5; it displays an instance of the application during DES encryption on a large file. The GridCrypt program monitors each of the threads it sends to the Alchemi manager. In the figure, we can see the monitoring panel at the bottom portion of the user interface. The blue circle indicates the thread that has been submitted and has finished while the grey circle indicates a thread that has been submitted to Alchemi manager but has not returned. The red circles shows which threads are currently doing write process to the random access file. A custom configuration form is used to specify the block size. Note that larger the block size configuration for the file split process means lesser number of work units.

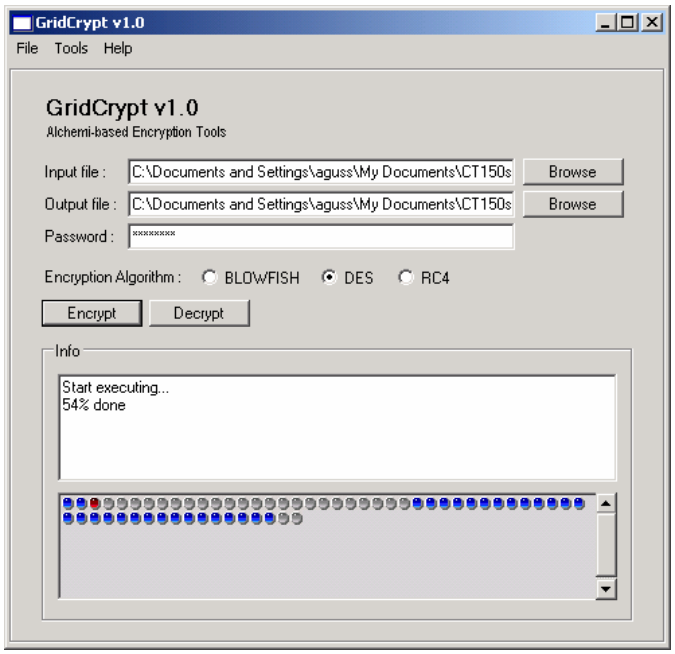

Figure 5. GridCrypt at runtime.

4.1 Runtime Comparison

We have done a runtime comparison for the GridCrypt application using 4 executor nodes each with the same specification of Pentium IV 2400 MHz processor and 512 MB of memory and running Windows 2000 Professional operating system. All these nodes were interconnected over a shared student laboratory LAN network of 100 Mbps. The Alchemi manager was installed on a separate computer together with SQL Server 2000. A separate user machine was used to initiate the execution of the GridCrypt application. We monitored the CPU usage and the threads execution details using the Alchemi console (see Figure 6).

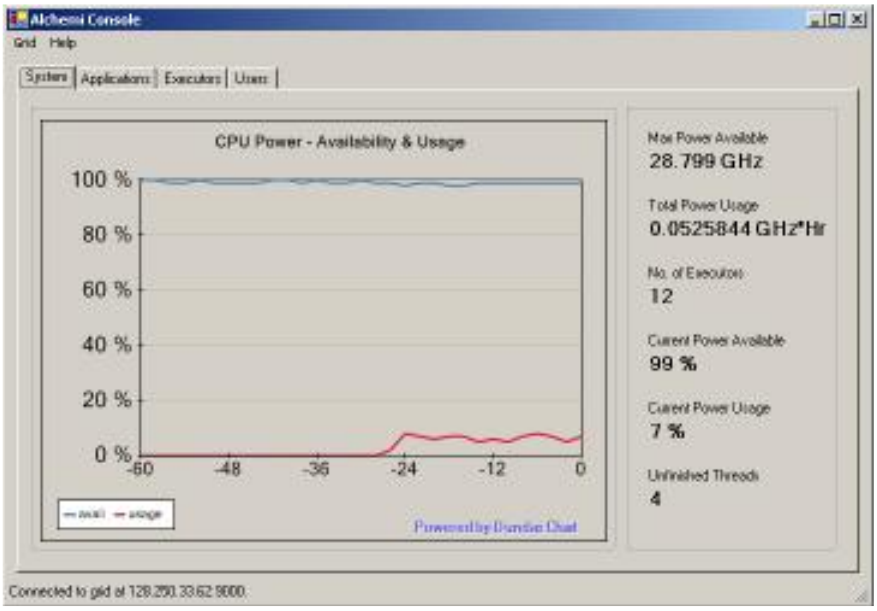

Figure 6. Alchemi performance while running 12 executors.

The encryption experiments were conducted on files of size 55 MB and 110 MB with 10 Mb block size, which lead to the creation of 6 and 11 work units respectively. For each file the encryption was carried on 1, 2, and 4 executor nodes. The performance results of these experiments are shown in Figure 7.

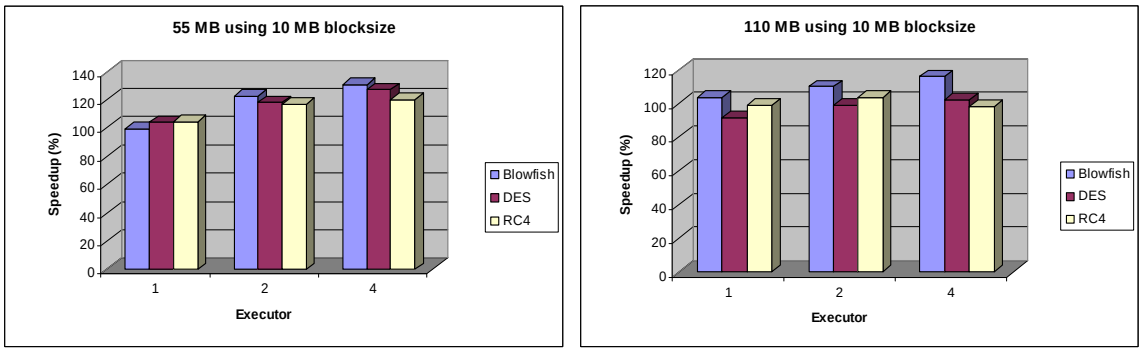

Figure 7. Performance improvement for 55 MB and 110 MB file size with 10MB block size.

We also carried out experiments for a file of size 55MB with block size of 1MB on 1, 2, 4, 8, and 12 executor nodes. This scenario led to the creation of 55 work units. The configuration of executor nodes was similar to the above experiments and performance results are shown in Figure 8.

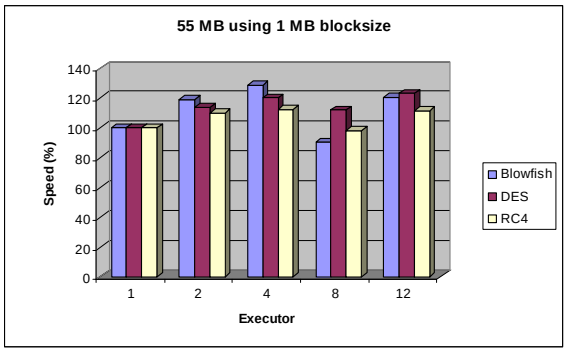

Figure 8. Performance for 55 MB file size with 1MB block size.

In all experiments, there was a reasonable performance improvement when 2 and 4 executors were used -- roughly 120% and 130% respectively. This gain is not linear; a similar phenomenon is also observed in [14]. Although there was a reasonable performance improvement when up to 4 executors were used, there was drop in performance gain when number of executors was increased as show in Figure 8. This is due to various overhead factors including (a) involvement of large datasets with low computation to communication ratio, (b) existence of serial processing component (file splitting and collating results), (c) the use of slow and shared network, and (d) the overhead of the distributed execution environment (e.g., distribution of executable, initiation of execution on a remote node, and management of threads).

A file copy test between two computers in the network for 110 MB file took an average of 20 seconds, which indicates effective usable bandwidth of 5.5 MB/sec. in our shared student laboratory network. Considering the data transfer between the GridCrypt application and executor nodes is via the Alchemi manager, data for each work unit has to travel across the network 4 times. Thus, the file transfer overhead alone contributes about 60-70% of the processing time in this experiment. The use of a faster network such as Gigabit Ethernet and faster storage systems will help minimize the overhead. In addition, although this overhead can be approximately halved by bypassing the manager and transferring date files between the user host and executors directly, it violates the current Alchemi security model.

5 Related Works

One of the approaches to increase the performance of symmetric key cryptography was carried by *Praveen Dongara and T. N. Vijaykumar* [3]. They were implementing Interleaved Cipher Block Chaining method on Symmetric Multiprocessors. This is the case where the effort to improve symmetric key cryptography is focused on the algorithm and hardware. Another related effort carried out by *Jerome Burke, John McDonald* and *Todd Austin* [4] adds instruction set support for fast substitutions, general permutations, rotates, and modular arithmetic. Their experiment has shown overall speedup to the symmetric key cryptography. While these two approaches also enhance the performance of symmetric key cryptography, our approach is scalable and cost-effective due to the use of a commodity-based high-performance computing platform.

6 Conclusion

While the performance of enterprise grid symmetric key cryptography that was implemented using Alchemi shows an increase over the single processor version of the symmetric key cryptography, the performance improvement is limited by the I/O and communication overhead. The use of high performance networks can enhance performance. Another way increase performance to transfer the data directly between the user host and executors. However, it violates the current Alchemi security model and requires enhancement of Alchemi security to support rights delegation.

References

[1] Wikipedia. Symmetric key algorithm. Updated March 10, 2004. http://en.wikipedia.org/wiki/Symmetric_key_algorithm (June 11, 2004)

[2] W. Stallings. *Cryptography and Network Security: Principles and Practice*, 3rd Edition. Prentice Hall, New Jersey, USA, 2003.

[3] Praveen Dongara, T. N. Vijaykumar. *Accelerating Private-key cryptography via Multithreading on Symmetric Multiprocessors*. In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), March 2003.

[4] Jerome Burke, John McDonald, Todd Austin. Architectural Support for Fast Symmetric-Key Cryptography. Advanced Computer Architecture Laboratory University of Michigan.

[5] MyCrypto.net. Encryption Algorithms. http://www.mycrypto.net/encryption/crypto_algorithms.html (June 11, 2004)
[6] CryptographyWorld.com. The Cryptography Guide: Triple DES. http://www.cryptographyworld.com/des.htm (June 11, 2004)
[7] searchSecurity.com. Data Encryption Standard. http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci213893,00.html (June 11, 2004)
[8] B. Schneier. The Blowfish Encryption Algorithm -- One Year Later. http://www.schneier.com/paper-blowfish-oneyear.html (June 11, 2004)
[9] searchSecurity.com. BlowFish. http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci213676,00.html (June 11, 2004)
[10] R. J. Jenkins Jr. ISAAC and RC4. http://burtleburtle.net/bob/rand/isaac.html (June 11, 2004)
[11] VOCALTechnologies, Ltd. RC4 Encryption Algorithm. http://www.vocal.com/RC4.html?glad (June 11, 2004)
[12] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal, Alchemi: A .NET-based Desktop Grid Computing Framework, *High Performance Computing: Paradigm and Infrastructure*, Laurence Yang and Minyi Guo (editors), Wiley Press, New Jersey, USA. Fall 2004. (in print)
[13] Wei-Meng Lee. Creating Stealthy Software with the .NET Cryptography Classes. Hardcore Visual Studio .Net. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnhcvs03/html/vs03l1.asp
[14] Daniel M. Pressel. Scalability vs. Performance. U.S. Army Research Laboratory Aberdeen Proving Ground, MD 21005-5066. http://www.hpcmo.hpc.mil/Htdocs/UGC/UGC00/paper/daniel_pressel_scale_paper.pdf

Appendix A: Pseudo code for multithreaded symmetric key cryptography.

```
/* initialize alchemi grid engine */
InitializeGrid ();
…
/* prepare for reading file */
inFile = new FileStream (inputFileName, FileMode.Open, FileAccess.Read);
binaryReader = new BinaryReader (inFile);
inFile.Seek (0, SeekOrigin.Begin);
…

/* prepare the output file */

outFile =  new System.IO.FileStream (outputFileName, System.IO.FileMode.Truncate,
           System.IO.FileAccess.Write);
binaryWriter = new BinaryWriter (outFile);
…
/* determine number of block */
int blockCount = (int) (fileSizeIn / BLOCKSIZE) + 1;
…
/* determine last block size */
int lastBlockSize = (int) fileSizeIn % BLOCKSIZE;
…
/* initialize crypto engine */
myEngine = new Crypto (password);
…
/* create gridcrypt threads, loop from block 1 to n-1 */
for (int i=0; i < blockCount-1; i++)
{
       /* read n size file */
       buffer = binaryReader.ReadBytes (BLOCKSIZE);
       /* create one block */
       blocks[i] = new FileBlock (i, buffer);
       /* create gridcrypt thread based on algorithm chosen*/
       GridCryptThread myGCT1 = null;
```

```
        /* create new GridCrypt Thread */
        myGCT1 = new GridCryptThread (Crypto, blocks[i], myEngine);
        …
        /* add the thread to alchemi engine */
        ga.Threads.Add (myGCT1);
}
…
/* create the last gridcrypt thread*/
/* read last block file */
buffer = binaryReader.ReadBytes (lastBlockSize);
…
/* create one block */
blocks[blockCount-1] = new FileBlock (blockCount-1, buffer);
…
/* create gridcrypt thread based on algorithm chosen */
GridCryptThread myGCThread = null;
myGCThread = new GridCryptThread (Crypto, blocks[blockCount-1], myEngine);
…
/* add the thread to alchemi engine */
ga.Threads.Add (myGCThread);
…
/* clean up resources */
…
/* assign the finish thread handler to the alchemi */
ga.ThreadFinish += new GThreadFinish (UpdateOutput);
…
/* assign the finish application handler to the alchemi */
ga.ApplicationFinish += new GApplicationFinish (ApplicationFinish);
…
/* start the engine */
ga.Start ();
```