High Performance Multithreaded Model for Stream Cipher

Khaled M. Suwais and Azman Bin Samsudin

School of Computer Sciences, Universiti Sains Malaysia (USM), Penang, Malaysia

Summary
New multithreaded model for stream cipher algorithms is presented as a step to enhance the performance (encryption rate) of stream ciphers. The architecture of the model relies on the multi-core technology which has become a common nowadays. The model is divided into three (components) phases: Thread Creation Phase, Keystream Generation Phase and Encryption Phase. Multiple threads will be created and synchronized in order to ensure higher performance and stable execution among the model's components. The experimental results show that the encryption rate of the tested ECSC-128 stream cipher has increased greatly. However, the main characteristics of the proposed model are easy to implement and fast on execution in a compact structure.

Key words:
Multithreading, Multi-Core, Stream Cipher and Encryption.

1. Introduction

Improving the performance of any system or application is one of the important issues in our digital life (e.g. Information Exchanging, Security Applications, etc). The performance improvements can be seen from two different perspectives - the hardware and software perspective. Recently, the performance improvement which is based on the hardware development has lead to a fascinating innovation known as multi-core technology, resulting in faster computation and higher throughput. On the other hand, since the 1960s [1], the software-based performance improvement such as multithreading techniques has shown acceptable results as opposed to the limitation of the resources at that time. Anyhow, the improvement of the performance will enhance the systems and the applications from two angles. It will speed up the execution and calculations of currently available systems, and from the other angle, it will give the chance for new systems to be implemented after it was restricted due to the intensive calculations and resources limitation. In this paper, we are presenting multithreading model in cooperation with multi-core technology to improve the performance of one of the symmetric cryptographic algorithms known as Stream Cipher in order to facilitate the chances for such ciphers to increase their security level even with intensive computations.

Manuscript received March 5, 2008
Manuscript revised March 20, 2008

2. Preliminaries

In this Section we highlight the two components of our proposed model. The two components are the multithreading techniques and the stream cipher algorithm.

2.1 Multithreading System Architecture

As multi-core computers become more common nowadays, executing sequential applications has become obstructive to performance. In order to make use of the extra cores, applications must be divided into smaller segments (tasks) and those segments need to be run at the same time on separate cores in a process known as concurrent programming.

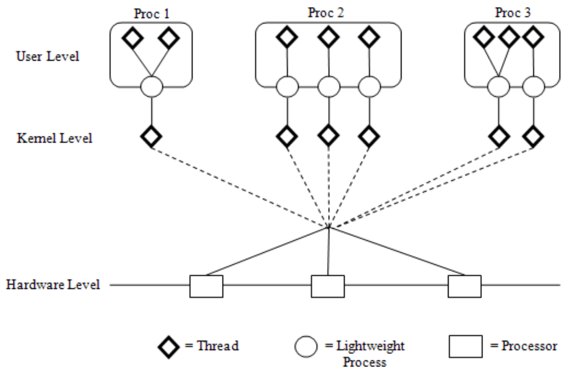


Fig.1. Multithreaded System Architecture [modified from 1]

One way of expressing concurrency is by dealing with low-level technology known as Threads. It can be viewed as mapping independent tasks to threads to give the operating system greater flexibility in processes scheduling, which in turn helps to hide program latency [2]. The process of mapping independent tasks to threads, and the connection between the three systems' level (user, kernel and hardware) of the multithreaded system architecture is portrayed in **Fig.1**.

Lightweight Processes (LWP) are the underlying threads of control supported by the kernel. It can be viewed as a virtual CPU which links the user level to the kernel level. **Fig.1** shows that each process (Proc 1 to Proc 3) can be executed by different numbers of threads, resulting in faster execution in shorter time. In fact, multi-core technology provides systems with greater performance compared to single-core technology by providing extra cores (Macro-Improvement). Furthermore, multithreading techniques work in lower levels to improve the performance by allowing many threads associated with each core to accomplish one task (Micro-Improvement). **Fig.2** illustrates the performance gained by applying multithreading techniques to multi-core machine.

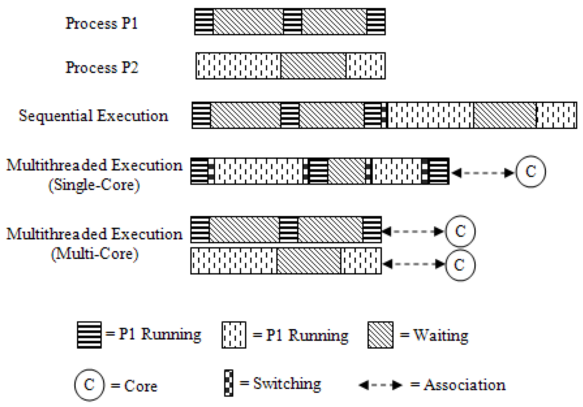


Fig.2. Sequential and Concurrent Execution Performance Comparison

We aim to use multithreading technique in our model for several reasons. Firstly, to improve the stream cipher responsiveness by dividing the structure of the cipher into independent or semi-independent tasks. Secondly, to make use of multi-core efficiency by giving the running threads more resources (cores). Thirdly, to improve the performance of stream cipher algorithms by executing the cipher's tasks concurrently.

2.2 Stream Cipher Algorithm

One of the cryptographic primitives used to ensure secure communication over public and unsecured channels is the stream cipher. Stream cipher algorithm is based on generating a pseudorandom keystream to encrypt a stream of plaintext to generate a stream of incomprehensible text known as ciphertext as shown in **Fig.3** [3].

In fact, this type of encryption algorithms is recommended since it is efficient in software and hardware implementation [3, 4]. Examples on stream ciphers are: RC4 [5], SNOW [6], ECSC-128 [7] and many other ciphers. The core of those ciphers are basically based on simple logical operations and bit manipulation as in RC4, Linear Feed-Back Shift Registers as in SNOW or complex mathematical problems as in ECSC-128.

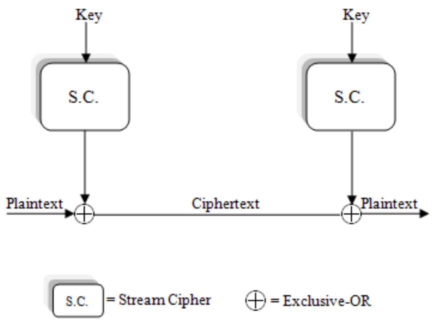


Fig.3. Stream Cipher Algorithm

In this research, we will choose ECSC-128 stream cipher as a targeted algorithm to apply our multithreaded model in order to show the mechanism and the effects of our proposed model as described in Section 4.

3. ECSC-128

ECSC-128 is a stream cipher based on the intractability of the Elliptic Curve Discrete Logarithm Problem (ECDLP) [7]. The design of this cipher includes three stages: Initialization Stage (IS), Keystream Generation Stage (KGS) and the Encryption Stage (ES) as portrayed in **Fig.4**.

The algorithm starts by IS stage with two parameters (Pt, ℓ), where Pt is the plaintext and ℓ is the input key of 128 bit. The output of this stage is a point P_1 on the curve E and initial value of the key k. The value of P_1 is calculated by embedding ℓ on E with the correct (x, y) coordinates.

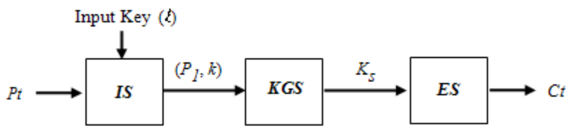


Fig.4. Overall Design of ECSC-128

The output of IS stage will be the input of the following KGS stage. This stage forms the core of ECSC-128 and it is based on multiplying the point P_1 by a big integer k in a process known as Point Multiplication. The idea of performing this multiplication is to calculate new point on the curve P_2 such that $P_2 = k\,P_1$. The resulted point will be transformed to its binary representation used in the next stage ES.
As ECSC-128 reaches its last stage, ES, and as other stream ciphers, it will perform exclusive-OR operation between the plaintext bits and the generated keystream bits from the previous stage, KGS. The size of the keystream is

320-bit. It will be divided into ten sub-keystreams as shown in **Fig.5**.

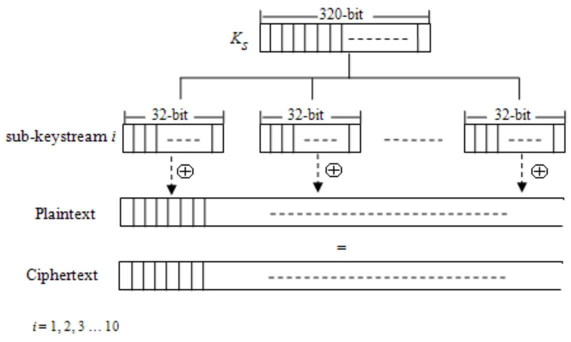


Fig.5. Encryption Stage of ECSC-128

There are two important reasons behind applying our proposed model on the ECSC-128 stream cipher. Firstly, ECSC-128 shows its resistance against cryptanalysis attacks since it is based on the well known complex problem ECDLP. Secondly and most importantly is the simplicity of its structure which divides the cipher into stages. In fact, the simplicity of this structure makes applying our model easier and more efficient. In the next section we will discuss our proposed model in detail and show how it is applied on the chosen cipher.

4. The Proposed Multithreaded Model

Our proposed model is divided into three main phases (components). The first phase includes the threads creation and initialization processes where we create threads to perform tasks related to generating keystreams and encrypting plaintext. The second phase is the keystream generation phase, where we generate more than one thread at the same time (concurrent keystream generation). The last phase of the multithreaded model is designed to encrypt the stream of plaintext with the incoming keystream from the previous phase. The encryption process will be performed concurrently more than once, as we will describe later.

4.1 Thread Creation and Initialization Phase

In terms of multithreading techniques proposed in our model, we will start by creating threads to be associated with specific tasks in a phase called the Thread Creation Phase (TCPr). We aim to create two threads to be bind with each task (total of 4 threads) on a multi-core machine. The two tasks associated with each of the two threads among the four threads are the keystream generation and encryption processes. **Fig.6** illustrates the TCPr and its associated tasks.

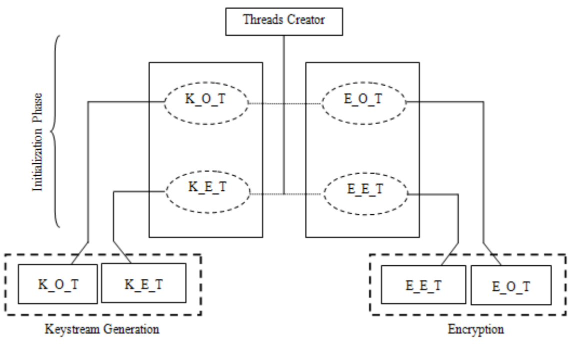


Fig.6. Thread Creation Process (TCPr)

In general, applying the first phase of our model to ECSC-128 is straightforward and thread creation can be simply added to the initialization stage (IS). Therefore, the IS will be responsible for generating the initial values of key *k*, point P_1 on curve *E* and executing the embedded TCPr component as shown in **Fig.7**.

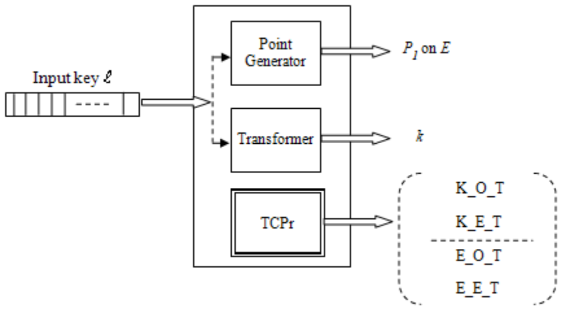


Fig.7. Applying TCPr component on IS stage of ECSC-128

4.2 Keystream Generation Phase

Now we move on to the second phase of our model which deals with the keystream generation process. The K_O_T/E_O_T and K_E_T/E_E_T threads in the above figure refer to the processing fashion while the K_O_T thread generates a sequence of keystream bits based on the odd increment of the key *k*, and E_O_T threads use those keystream bits to encrypt bits at the odd position of the plaintext. Meanwhile, K_E_T and E_E_T threads of the keystream generation and encryption processes work in the same processing fashion but with even increment of *k* for K_E_T and even bit position of the plaintext for E_E_T as shown in **Fig.8**.

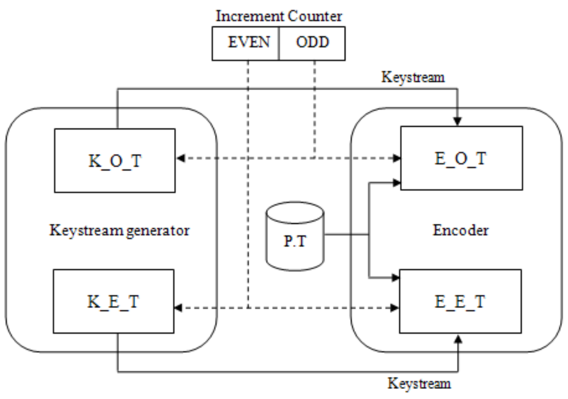


Fig.8. Threads cooperation for keystream generation and plaintext encryption

The second phase of our model presents high cooperation between two stream cipher tasks - keystream generation and Encryption tasks (equivalent to KGS and IS stages in ECSC-128). The cooperation is based on the Producer-Consumer fashion that provides the system with some synchronization between the running threads in order to ensure successful feeding of bits from the keystream generator to the encoder. The synchronization between the KGS and ES stages of ECSC-128 is presented in **Fig. 9**.

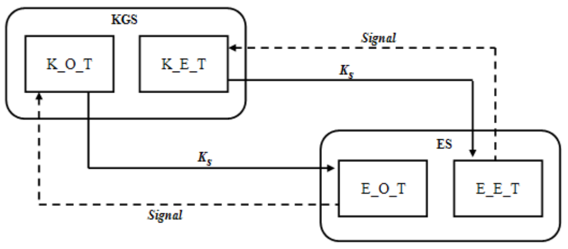


Fig.9 Threads Synchronization

The goal of designing this phase is to generate multiple keystreams concurrently. Those generated keys are used as secret keys in the encryption phase. The concurrent multiple keystream generation process is visualized in **Fig.10**.

4.3 Encryption Phase

The encryption phase of our model is designed to encrypt the given plaintext by applying XOR operation between the keystream bits and the plaintext bits. The encryption process is accomplished by dividing the activities into two parts handled by two threads and synchronized with their analogue threads in the keystream generator component. The two threads which control the encryption phase have an extra job of monitoring keystream bits availability. The two threads, before performing the encryption process, will check the available bits of the pre-generated keystream. If the whole keystream generated in round *i* (R$_i$) is used, the thread is responsible for signaling its corresponding thread in the keystream generator to generate a new keystream. The encryption process is illustrated in **Fig.11**.

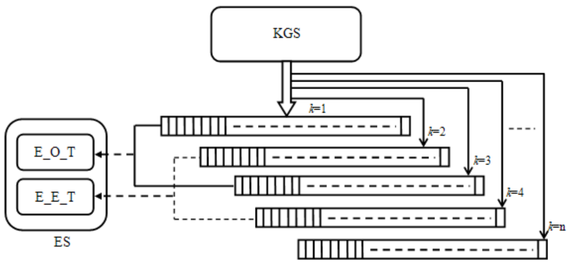


Fig.10 Concurrent Keystreams Generation

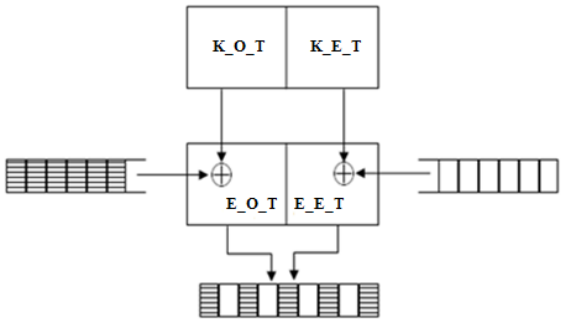


Fig.11 Synchronized Encryption Phase

5. Implementation and Performance Evaluation

As mentioned earlier in this document, the aim of designing a multithreaded model for stream cipher is to increase the performance since the current stream ciphers tend to use intensive calculations for keystream generation in order to increase the provided security level. Our model with ECSC-128 was coded in C using MinGW-2.05. POSIX-2.8.0 (Pthread) library is used to handle thread-related functions found in the model. For testing purposes we ran our model (applied on ECSC-128) on a workstation with Core 2 Duo® 2.13GHz processor and 2GB of RAM.

Single thread execution (sequential) is a one-path execution whereby the work flow of threads associated with each core will start the execution at time t_0 and finish at t_m. For instance, the time taken to execute KGS and IS stages of ECSC-128 is 0.09ms and 0.002ms (in one round) respectively, and the time required during switching is σ = 0.001. Therefore, the work flow of two rounds can be visualized as appear in **Fig.12-A**. The total time required to execute the KGS and ES is computed as the following (Eq. 1):

$$t_m = R\sum_{i=1}^{m} time(C_i) + \sigma \qquad (1)$$

where R is the number of rounds, i is the number of components to execute in each round, m is the total number of components in the overall rounds, σ is the wasted time during switching, and $time(C_i)$ is the time required to execute each component. In our case, the total time t_m = 2(0.09 + 0.002) + 3(0.001), hence t_m = 0.187ms. In contrast to the single-thread execution, multi-thread execution provides multiple-paths to accomplish the same task as shown in **Fig.12-B**. The total time required to accomplish the same work flow as discussed above is divided among multiple cores, resulting in higher throughput and performance. In this case, t_m = 0.09 + 0.002 + 0.001, hence t_m = 0.093ms.

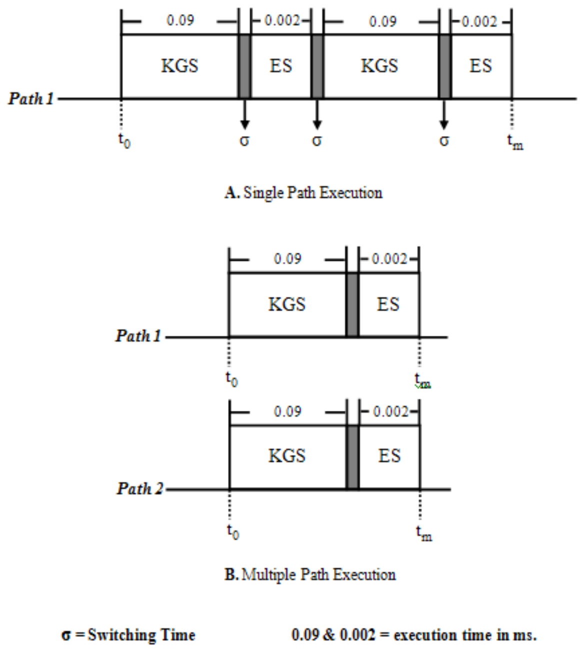


Fig.12 Single and Multiple Path Execution comparison

The performance of our model can be evaluated by comparing the encryption rate gained by the traditional execution of ECSC-128 and the encryption rate gained after applying the multithreaded model. The encryption rate of the ECSC-128 stream cipher (single thread) is 27108byte/second (executed on Single-Core machine). Executing the same cipher on Duo Core 2.13GHz machine with 2GB RAM has increased the encryption rate relatively as shown in **Fig.13**.

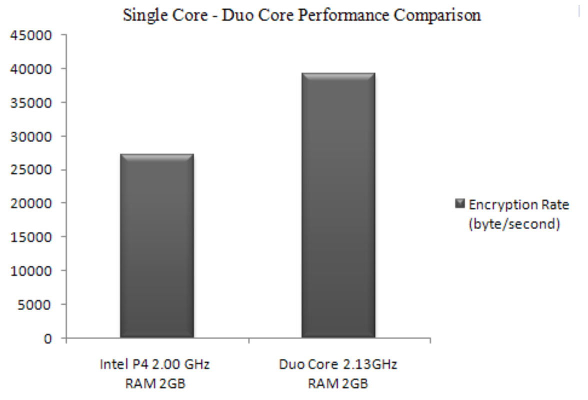
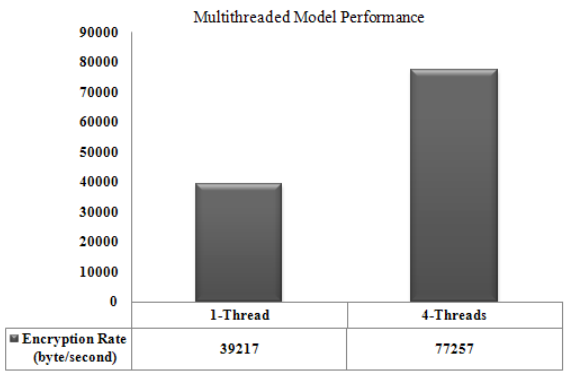


Fig.13 ECSC-128 Performance Comparison

The multithreaded model proposed in this paper shows good results compared to the sequential execution of the stream cipher. The multi-core architecture provides us with better performance due to the extra resources (core) added to machines. **Fig.14** evaluates the results gained from applying our model (on ECSC-128) based on executing it on two-core machine (machine specification as given above) compared to the sequential execution.

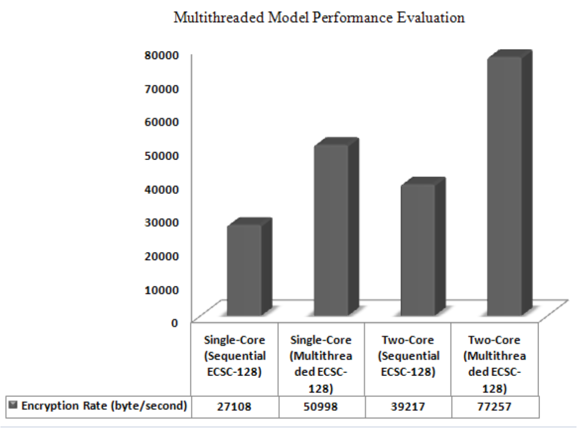


	1-Thread	4-Threads
▨ Encryption Rate (byte/second)	39217	77257

Fig.14 Performance Evaluation of the Multithreaded Model

From another perspective applying the multithreaded model on a single-core machine has shown some relatively good results compared to the sequential execution. **Fig.15** illustrates different results obtained by executing our model in different architectures.

After evaluating the performance of the model from different perspectives, we can conclude that increasing the number of cores of the machine and multithreading will enhance the performance of the model. This conclusion comes from the technical aspect, whereby adding extra cores to the machine will provide the running threads with more resources needed for faster execution.



	Single-Core (Sequential ECSC-128)	Single-Core (Multithreaded ECSC-128)	Two-Core (Sequential ECSC-128)	Two-Core (Multithreaded ECSC-128)
▨ Encryption Rate (byte/second)	27108	50998	39217	77257

Fig.15 Performance Evaluation of the Multithreaded Model Applied on Different Architectures

6. Conclusion

In this paper we present a new multithreaded model for enhancing the performance of the stream cipher algorithms (ECSC-128 as a case study). The multithreaded model is based on generating multiple threads that able to accomplish the overall works handled by some synchronization techniques. The model divides the execution of the stream cipher into three stages: Thread Creation Phase (TCPr), Keystream Generation Phase and the Encryption (Encoder) Phase. The implementation and performance evaluation and analysis show that the model is reliable and easy to implement. The strength of our architecture is in the utilization of currently available technology, whereby the model is implemented on multi-core technology. The multithreaded model showed great improvement in performance compared to the single thread (sequential) model. Lastly, complete descriptions of the proposed model phases and their implementation have been discussed in details in the context of this document.

Acknowledgments

The authors would like to express their thanks to Universiti Sains Malaysia for supporting this study.

References

[1] SunSoft, Multithreaded Programming Guide, CA: Sun Microsystems, 2002.
[2] Gabb, H., Common Concurrent Programming Errors, 2002, www.linux-mag.com/content/view/983/2038/1/0/ (accessed March 2, 2008).
[3] Mollin, R., An Introduction to Cryptography, 2nd Edition, Edited by Kenneth H. Rosen. Boca Raton: Chapman & Hall/CRC, 2007.
[4] Stalling, W., Cryptography and network security: principles and practice. 3rd, New Jersey: Prentice Hall, 2003.
[5] Rivest, R.., The RC4 Encryption Algorithm, RSA Data Security Inc., Document No, 003-013005-100-000000, 1992.
[6] Ekdahl, P. and Johansson, T., "A New Version of the Stream Cipher SNOW", In Selected Areas in Cryptography, Berlin / Heidelberg: Springer, 2003, pp. 47-61.
[7] Suwais, K. and Samsudin, A., "ECSC-128: New Stream Cipher Based on Elliptic Curve Discrete Logarithm Problem", First International Conference on Security of Information and Networks (SIN 2007), Famagusta: Trafford, 2007, pp.13-23.

Khaled M. Suwais received a B.Sc. degree in Computer Science from Al al-Bayt University in 2004. He obtained his M.Sc in Computer Science from Universiti Sains Malaysia (USM) in 2005. Since 2006, he has been working as a research and teaching assistant at the School of Computer Sciences, USM. Currently he is a PhD student at the School of Computer Sciences, USM. His research interests are in Cryptography and Parallel Computing.

Azman Samsudin is a lecturer at the School of Computer Sciences, Universiti Sains Malaysia. He received the B.Sc. degree in Computer Science from the University of Rochester, USA, in 1989. He obtained his M.Sc. and Ph.D. degrees in Computer Science from University of Denver, USA, in 1993 and 1998, respectively. His research interests are in the field of Cryptography, Interconnection Switching Networks, and Parallel Computing.