International Journal of Security and Its Applications
Vol.7, No.5 (2013), pp.1-16
http://dx.doi.org/10.14257/ijsia.2013.7.5.01

An Empirical Study to Compare the Performance of some Symmetric and Asymmetric Ciphers

Dr. Najib A. kofahi

Department of Computer Science
Faculty of Information Technology and Computer Sciences
Yarmouk University
Irbid, Jordan
nkofahi@yu.edu.jo

Abstract

In this paper we present empirical results obtained from Java implementation of Elliptic curve Cryptosystem (ECC) as an asymmetric block cipher algorithm and a set of symmetric block cipher algorithms namely Triple-Data Encryption Standard (T-DES), Advanced Encryption Standard (AES), and Blowfish. Performance evaluation based on CPU execution time is presented under WinXP and Linux.

We used in our implementation Java programming language, Java Cryptography Architecture (JCA) and Java Cryptography Extension (JCE). The evaluation of the performance of these algorithms is done for secret key generation and encryption and decryption operations. Results indicated that ECC outperforms the other encryption/decryption algorithms considered in this study regarding the security strength, speed, and key size of ECC. Also, ECC's performance advantage increases as security needs increases for newly emerging applications.

Keywords: *ECC, AES, Triple-DES, Blowfish, Asymmetric Ciphers, Symmetric Ciphers, Performance Evaluation, Java, WinXP, Linux*

1. Introduction

The growth of internet services in general and e-commerce in particular such as e-banking, enterprise collaboration, etc… can be an important factor in increasing national productivity. This growth plays an important role in the global development of economy and can be a key driver of increasing sales while using fewer production resources. A prosperous e-commerce market is also a key pillar of the digital economy. The digital economy is increasingly becoming a priority policy area for governments as well [1].

The need to protect and secure this sensitive information including e-commerce, mobile-commerce, secure messaging, security of ATM cards, computer passwords and other electronic transactions have placed a great demand on a strong and efficient internet security system. It is found that at least 92% of web applications are vulnerable to some form of attack [2].

Hackers and cryptanalysts are actively targeting all Internet services, which necessitates the need for all kinds of security measures including encryption/decryption algorithms to prevent intruders' attacks and thus decrease security vulnerabilities. Cryptanalysis techniques may be classified as brute force attack, linear and non linear cryptanalysis, n-gram analysis, meet in the middle attack, man in the middle attack, *etc*.

These days, developments in security over the Internet is the key to confidence for conducting Internet services and for people wanting to protect their sensitive information, or doing business and all kind of communications over the Internet.

ISSN: 1738-9976 IJSIA

Several algorithms with varying properties were developed and employed for this purpose. Encryption is the process of converting ordinary information (plaintext) into unintelligible text called cipher text (encrypted text) by applying an encryption algorithm to the plaintext message. Decryption on the other hand is the reversing process that recovers the original text from unintelligible cipher text by applying a decryption algorithm to the encrypted message. A cipher is a pair of algorithms which creates the encryption and the decryption processes.

The two general types of key-based encryption/decryption algorithms are symmetric (also called conventional) algorithms [3-5] and asymmetric (or public-key) algorithms [3-8].

In symmetric algorithms [9], any two parties interested in encrypting/decrypting data have to use the same (secret) key generated for both encryption and decryption. Symmetric algorithms are divided into two families: **block ciphers** which operate on blocks of data where message is broken into blocks of bits, and **stream ciphers**, which operate on single bit at a time. Most popular symmetric block ciphers are Triple-DES, AES, and Blowfish. A disadvantage of symmetric ciphers is the key management necessary to use them securely. Each distinct pair of communicating parties must, ideally, share a different key. Stream ciphers, on the other hand, create an arbitrarily long stream of key material, which is combined with the plaintext bit-by-bit or character-by-character, somewhat like the one-time pad. In a stream cipher, the output stream is created based on a hidden internal state which changes as the cipher operates. That internal state is initially set up using the secret key material. RC4 [9] is considered one of the widely used stream ciphers.

Asymmetric algorithms use public key for encryption and private key for decryption [10]. The major drawbacks of asymmetric ciphers are their speed and security strength; they are much slower than the symmetric algorithms and more vulnerable to intruder attacks but they make key exchange easier. Most popular asymmetric ciphers are Rivest, Shamir, and Adelman (RSA), Diffie-Hellman, and ECC. There are three families of asymmetric algorithms, namely: algorithms which are based on Integer Factorization Problem (IFP) like RSA, algorithms which are based on Discrete Logarithm Problem (DLP) like Diffie-Hellman and DSA, and algorithms based on finite field arithmetic like ECC. Most public-key algorithms involve operations such as modular multiplication and exponentiation, which are much more computationally expensive than the techniques used in most symmetric block ciphers, especially with typical key sizes [3].

In this work the implementation of ECC along with three symmetric algorithms (Triple-DES, AES, Blowfish) using Java Cryptography Architecture (JCA) and Java Cryptography Extension (JCE) will be carried out. Performance evaluation based on their execution time will be presented under WinXP and Linux platforms.

Section 2 introduces the encryption/decryption algorithms under consideration. The implementation of the algorithms using Java is discussed in Section 3. Section 4 introduces the experimental results. Finally, Section 5 gives the conclusions and future work.

2. The Encryption/Decryption Algorithms

Recently a performance evaluation based on execution time of encryption/decryption algorithms was reported [6, 8, 11-13]. We present here an implement of the ECC and three other algorithms in Java then compare their performance evaluation based on their execution time under WinXP and Linux platforms.

In the next subsections the block diagrams of the above mentioned algorithms under consideration are presented with a brief explanation of how each of them works.

2.1. The ECC Algorithm

The ECC uses curves whose variables coefficients are finite, there are two families commonly used on this cryptography, the first family uses elliptic curves $E(F_p)$ over prime finite field F_p (also called odd characteristic or modulo p and it is the field of integers modulo an odd prime number p) where p is large prime number. This family is the best for software implementations of ECC. The second family uses elliptic curves $E(F_{2^m})$ over binary field F_{2^m} (also called even characteristic or finite field with 2^m elements), where m is large integer number; this family is more suitable for hardware implementation of ECC [14].

The mathematical operations of ECC are defined over elliptic curve as given in [3, 14-18]. For the first family, the elliptic curve $E(F_p)$ is the set of points (x, y) that satisfies the following equation: $y^2 \bmod p = (x^3 + ax + b) \bmod p$, where $4a^3 + 27b^2 \neq 0$, such that x, y, a, b $\in F_p$.

For the second family, elliptic curve $E(F_{2^m})$ is the set of points (x, y) that satisfies the following equation:

$y^2 + xy = x^3 + ax^2 + b, \ b \neq 0,$ where a,b,x,y $\in F_{2^m}$.

The public key is a point on the elliptic curve and the private key is an arbitrary random number. The steps that underline how ECC algorithm works are described in [14]. Figure 1 depicts the key generation phase.

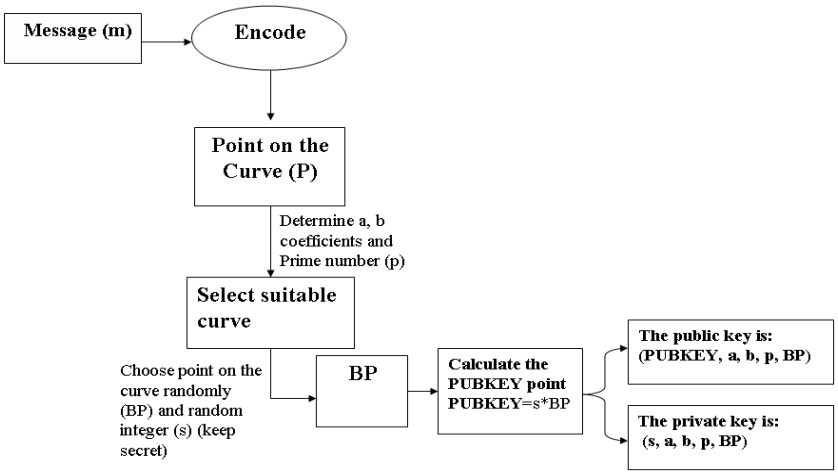

Figure 1. ECC Key Generation Block Diagram

To encrypt P, a user picks an integer, k, at random and sends the point (k* BP, P + k* PUBKEY). Figure 2 depicts the encryption operation.

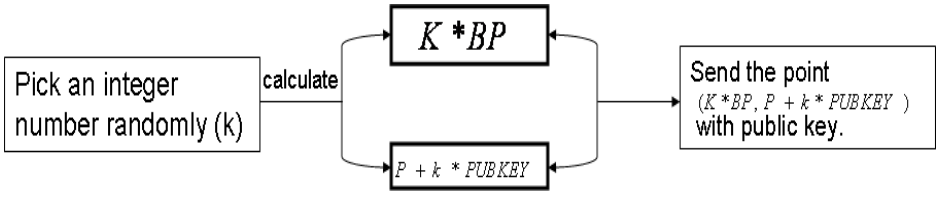

Figure 2. ECC Encryption Process Block Diagram

Decrypting this message is done by multiplying the first component of the received point by the secret key, s, and subtract it from the second component, *i.e.*,
(P + k * PUBKEY) − s * (k * BP) = P + k *(s* (BP)) − s * (k* BP) = P. This operation is shown in Figure 3.

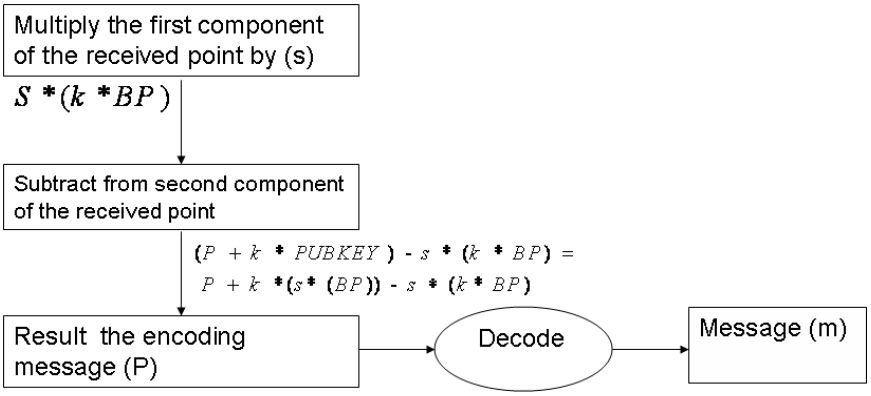

Figure 3. ECC Decryption Process Block Diagram

2.2. Triple-DES Algorithm

T-DES [6, 19, 20] is a modified version of the DES algorithm that improves the security strength of the DES by applying the algorithm three times in succession with three different keys of 56-bits each. The block diagram of T-DES is given in Figure 4 below.

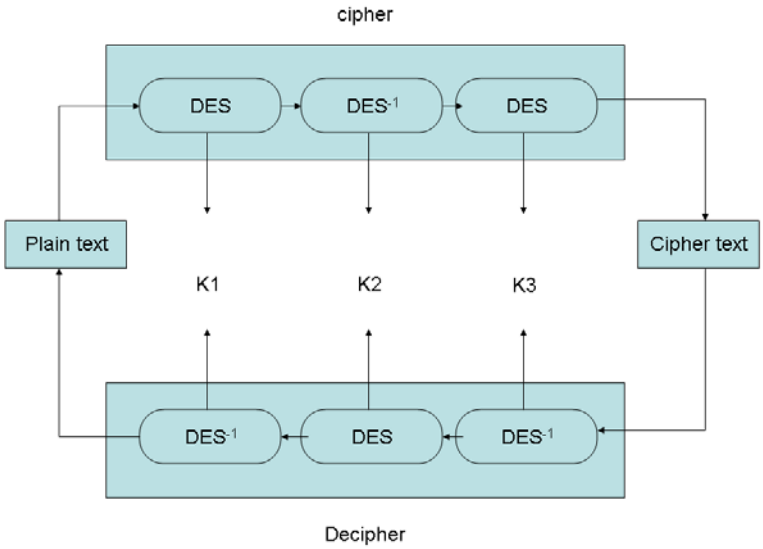

Figure 4. T-DES Block Diagram

2.3. AES Algorithm

AES [5, 6, 20, 21] was approved by the NIST in September 2000, as a replacement of the absolute DES. NIST required that AES should have security strength equal to or better than the TDES and significantly improved efficiency. AES consists of four invertible different stages that make up a standard round [4]. The stages are iterated 10 times for 128-bit key, 12 times for 192-bit key, and 14 times for 256-bit key. The four stages that compose the standard round are as follows:

- Substitute bytes: nonlinear procedure that uses the S-box to perform byte by byte of the data block.
- Shift rows: a simple transformation that uses permutation to shift the bytes within the data block in cyclic fashion.
- Mix columns: a simple transformation that uses arithmetic over $GF(2^8)$ to group 4-bytes together forming 4-term polynomial, then multiplies the polynomials with a fixed polynomial mod (x^4+1).
- Add round key: bitwise XOR of the current block with a portion of the expanded key.

The encryption/decryption process stages are depicted in Figure 5.

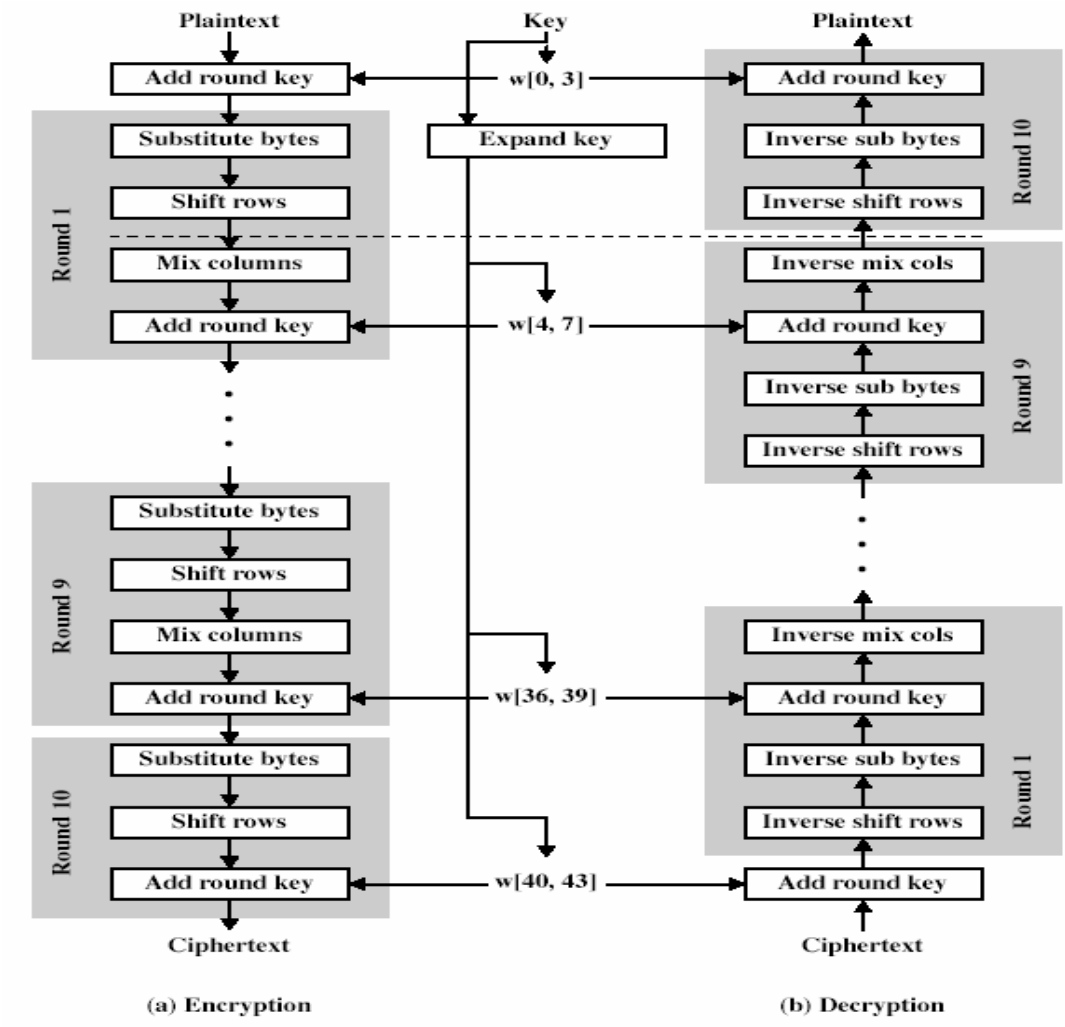

Figure 5. Block Diagram of Main Steps of AES [5]

2.4. Blowfish Algorithm

Blowfish uses 64-bits block size, and a variable key size ranges from 32-bits to 448-bits. Blowfish algorithm consists of two parts: key expansion part and data encryption part. Key expansion converts the key into several sub key arrays of total of 4168 bytes. Data encryption part is done via 16 round Feistel network. Each round consists of the key permutation, and the key- and data- dependent substitution.

All operations are XORs and addition on 32-bit words. The detailed descriptions can be found in [1, 6, 22]. Figure 6 shows the block diagram.

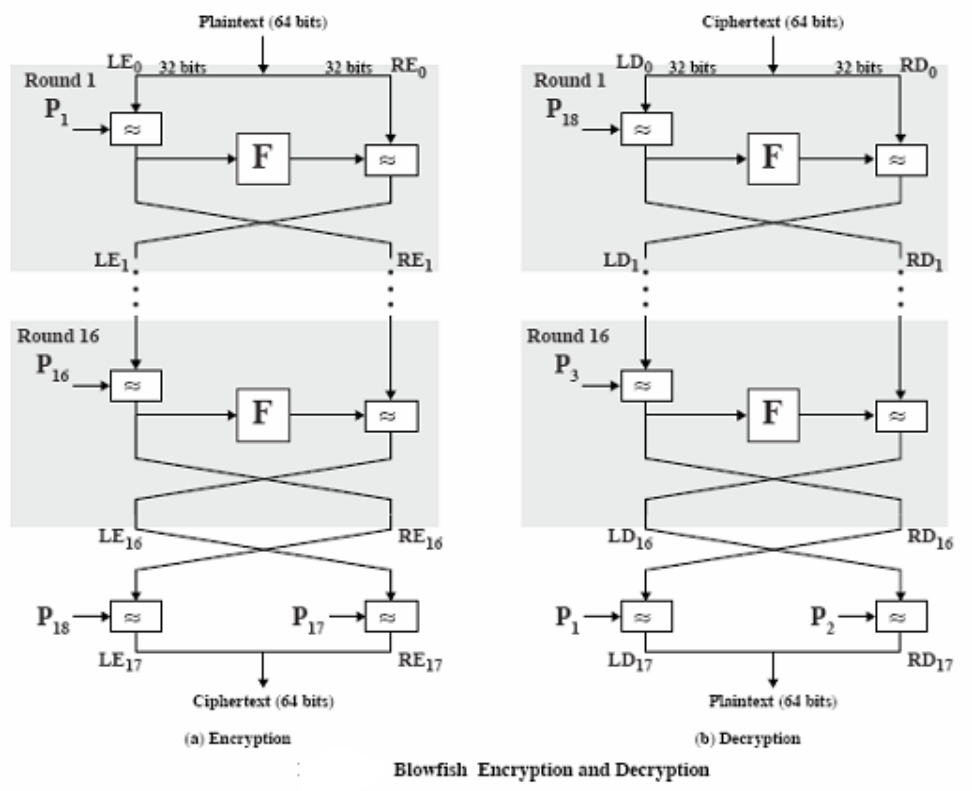

Figure 6. Block Diagram of Blowfish [3]

3. Implementation Details

In this section the steps involved in encryption/decryption processes are presented. Also, we briefly introduce the implementation programming language and packages associated with it. Java (JCA, JCE) tools are used to implement the above mentioned algorithms under both WinXP and Linux. The four above mentioned algorithms are tested with the same data sets, and then the execution time is measured for each of the three phases: key generation, encryption process, and decryption process.

3.1. Encryption/Decryption Processes

The following steps are used to generate the key pair for ECC:

(1) Initialize the required pair of keys and their sizes, and then specify the algorithm and provider that support them.
(2) Specify the suitable curve by determining the parameters of elliptic curve.
(3) Generate pair of Keys.
(4) Determine public and private keys.

The block diagram of the encryption phase, for the four algorithms under consideration, is shown in Figures 7.

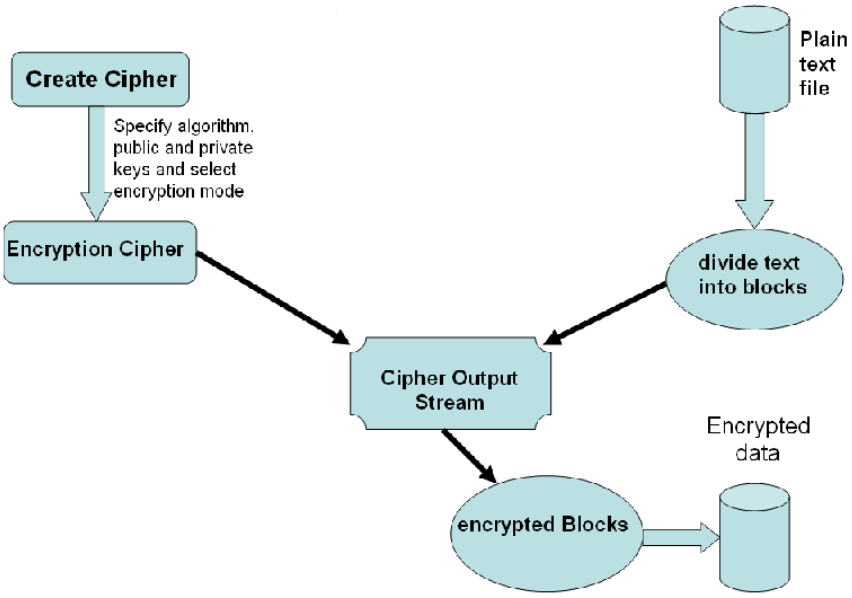

Figure 7. Block Diagram of Encryption Stage [6]

The following steps are used to encrypt the plaintext file:

(1) Initialize the cipher Cipher.getInstance (algorithm name, provider name);
(2) Initialize Integrated Encryption Parameters
(3) Determine the EC domain parameters, and specify the symmetric encryption algorithm and the MAC algorithm.
(4) Identify the cipher with public key, and specify the mode and parameters for encryption.
(5) Divide the file into blocks before reading.
(6) Create the encrypted cipher stream.
(7) Encrypt the data blocks.
(8) Write the encrypted data into output file

The block diagram of the decryption phase for the algorithms under consideration is shown in Figures 8.

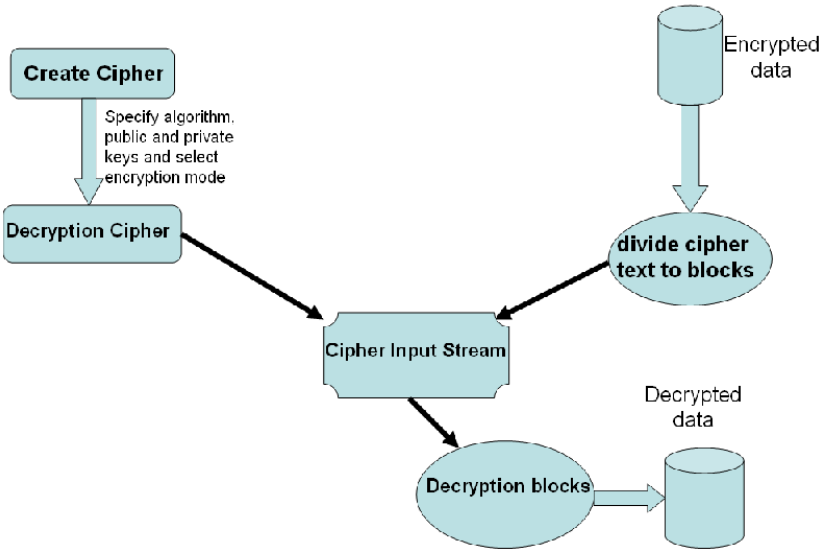

Figure 8. Block Diagram of Decryption Stage [6]

The following steps are used to decrypt the ciphertext file:

(1) Identify the cipher with public key, and specify the mode and parameters for encryption.
(2) Divide the file into blocks before reading.
(3) Create the decrypted cipher stream.
(4) Decrypt the data blocks.
(5) Write the encrypted data into output file

3.2. Java and Java Cryptography Architecture

In our implementation we have used Java programming language, Java Cryptography Architecture (JCA) package, and Java Cryptography Extension (JCE). JCA is a core Application Programming Interface (API) of the Java programming language and is designed to allow developers to incorporate both low-level and high-level security functionalities into their programs [23-28]. The JCA was designed around two principles, namely: Implementation independence & interoperability; and algorithm independence & extensibility [26, 27].

Java programmer deals mostly with the APIs, configuration options, proper handling of cryptographic entities such as certificates and key stores, and interfacing with other security products to satisfy the application's security needs.

The need to support flexible and fine-grained access control security policies, with extensibility and scalability, called for improved security architecture. The architecture introduced with the Java 2 Platform, Standard Edition (J2SE) 1.2, fulfills this goal. It was critical that the original release of JDK 1.0 consider security seriously and provide the sandbox security model. Later an enhancement on JDK 1.1 leads to the release of JDK 1.6 that makes the security solutions on the Java platform easier to use and more robust. JDK 1.6 was used in our implementation.

JCA and JCE are two Java APIs. They both are parts of J2SE SDK v1.6 [29, 30]. They define the general architecture and specific services for cryptographic operations. J2SE v1.6 comes with nine bundled providers: *SUN1.6*, *SunJSSE1.6*, *SunRsaSign1.5*, *SunJCE1.6*, *SunSASL1.5*, *XMLDSig1.0*, *SunPCSC1.6*, *SunMSCAPI1.6* and *SunJGSS1.0*.

Other third party providers like "BC1.6" [31], "Flexi1.6" [32] were also used in this implementation.

3.3. Packages and Classes used in Implementation

The following packages and classes are used in the program:

```
import java.io.*;                          // To initialize files using (File, FileInputStream,
                                           // FileOutputStram)   classes.
import java.security.KeyPairGenerator;   // To initialize pair of Keys and which algorithm
                                                    and
                                           //  provider support these keys and to initialize
                                           // key sizes.
import java.security.KeyPair;          // To generate pair of Keys
import java.security.PrivateKey;  // To determine the private key
import java.security.PublicKey;  // and the public key.

import java.security.SecureRandom;       // This class is used when the pair of keys
                                           // are generated to choose parameter randomly.
import java.security.Security;               // Used to add the two items of this provider and
                                                    to
                                         // use there code in program.

import javax.crypto.Cipher;        // Used to initialize cipher for the algorithm and  specify
```



```
                                    // the mode which we want ENCRYPT-MODE or
                                   // DECRYPT-MODE
import javax.crypto.CipherInputStream;          // to decrypt the file.
import javax.crypto.CipherOutputStream;         // to encrypt the file.

import de.flexiprovider.common.ies.IESParameterSpec; // To determine the EC domain
                                                     // parameters, and to specify the symmetric
                                                       // encryption algorithm and the MAC
algorithm.
import de.flexiprovider.core.FlexiCoreProvider; // Contain all the algorithms.
import de.flexiprovider.ec.FlexiECProvider;     // Contains all the EC features.
                                                       // to identify the curve parameters.
import de.flexiprovider.ec.parameters.CurveParams;
import de.flexiprovider.ec.parameters.CurveRegistry.BrainpoolP160r1;
```

4. Experimental Results and Discussion

In this section we present the experiments that we did for the four algorithms under consideration. We also analyze and interpret the results obtained from these experiments, and graphically present the execution time of each algorithm for the sake of performance evaluation. We experiment with different plaintext file sizes and different file contents (English language, Arabic-English language) under WinXp and Linux platforms.

4.1. Experiment Environment

The four algorithms were run on three different tests; the first test executed under windows XP professional version 2002 service pack 2 on Intel®, Pentium® with Dual-Core CPU speed of 1.60 GHz and a total of 1GB of RAM.

The second test was conducted in the same environment and was executed under Ubuntu /Linux version 8.10 released in October 2008. The third test also was conducted in the same environment of the first test, but the files contain Arabic and English texts. The goal of this test is to investigate the effect of file content on the execution time. The four ciphers are performed on the same files of sizes (100KB, 1MB and 10 MB). The algorithms were tested using key sizes shown in Table 1.

Table 1. Algorithms Key Sizes

Algorithm name	Key size
AES	128
Triple-DES	112
Blowfish	56
ECC	128

The experiment was repeated 100 times for each algorithm and for each operation, key generation, the encryption operation and the decryption operation.

The average of the 100 runs for each operation was computed for each algorithm.

4.2. Experimental Results and Analysis

In the experiment, the CPU execution time is computed for the four algorithms under WinXP and Linux. CPU time includes: system (kernel) time and user time. The system time is the execution time in kernel mode, and the user time is execution time in user mode.

4.2.1. First Test Results: The CPU execution time in seconds for the four algorithms under WinXP with file size of 100 KB is shown in Figure 9

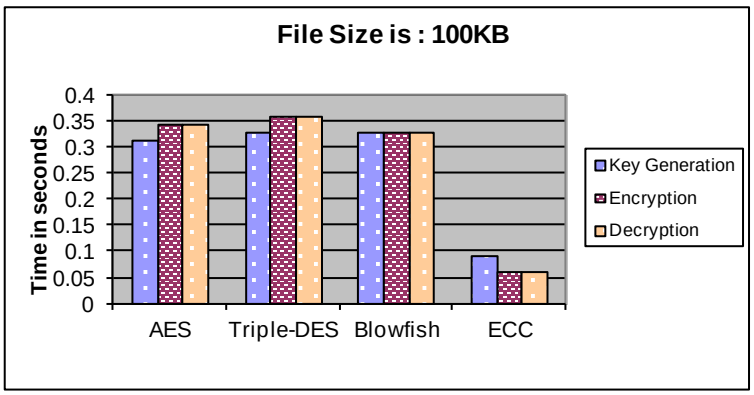

Figure 9. CPU Execution Time in Seconds (WinXP, 100KB)

The CPU execution time in seconds for the four algorithms under WinXP with file size of 1MB is shown in Figure 10.

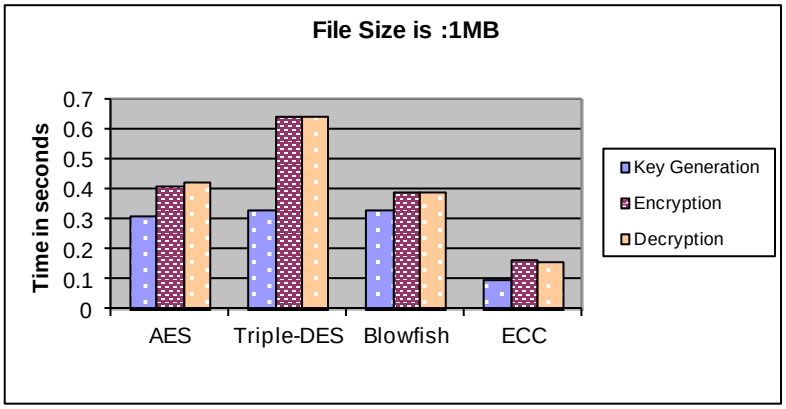

Figure 10. CPU Execution Time in Seconds (WinXP, 1MB)

Also, the CPU execution time in seconds for the four algorithms under WinXP with file size of 10MB is shown in Figure 11.

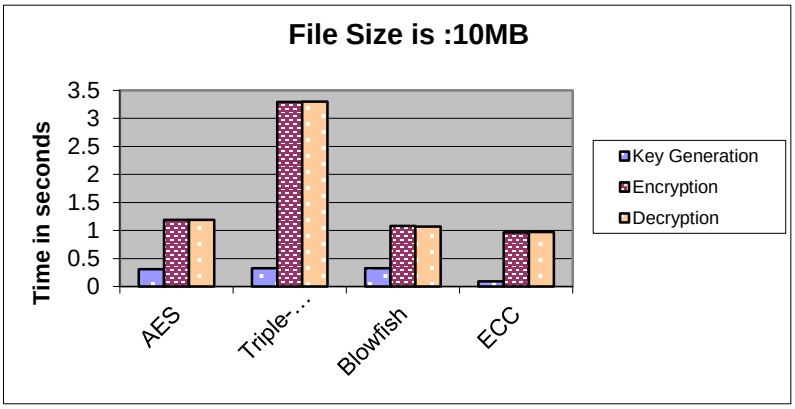

Figure 11. CPU Execution Time in Seconds (WinXP, 10MB)

4.2.2. Second Test Results: The CPU execution time in seconds, for the four algorithms under Linux with file size of 100KB, is shown in Figure 12.

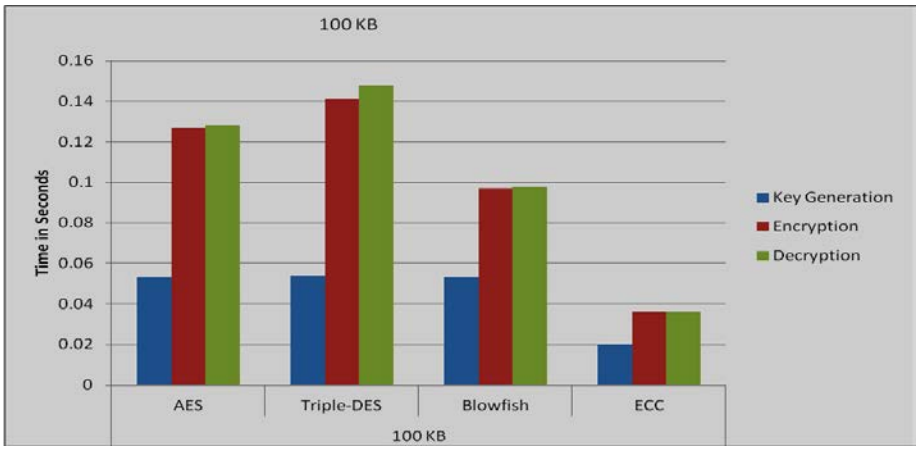

Figure 12. CPU Execution Time in Seconds (Linux, 100KB)

The CPU execution time in seconds, for the four algorithms under Linux with file size of 1MB, is shown in Figure 13.

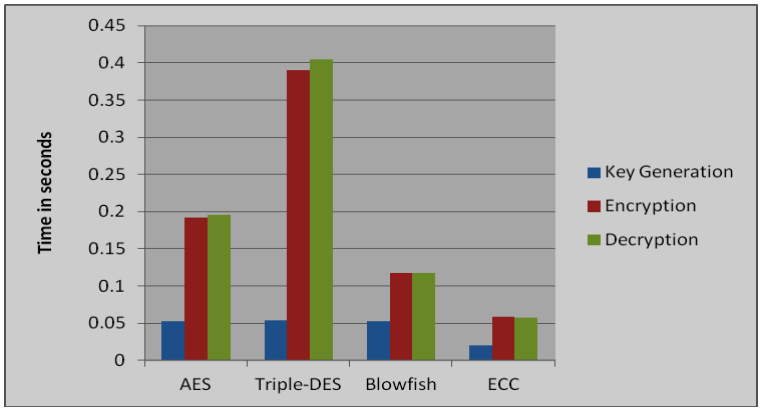

Figure 13. CPU Execution Time in Seconds (Linux, 1MB)

The CPU execution time in seconds, for the four algorithms under Linux with file size of 10MB, is shown in Figure 14.

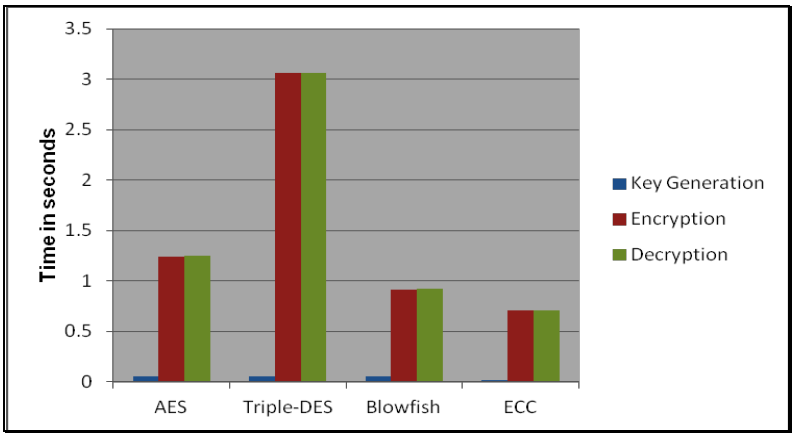

Figure 14. CPU Execution Time in Seconds (Linux, 10MB)

4.2.3. Third Test Results: The CPU execution time in seconds for the four algorithms under WinXP with English language file content of sizes of (100 KB, 1MB, 10MB), and a hybrid file content of same sizes are illustrated in Figures 15, 16, and 17 respectively.

The first column in the three figures depicts the execution time of the English file which includes only English texts, and the second column depicts the execution time for the hybrid file.

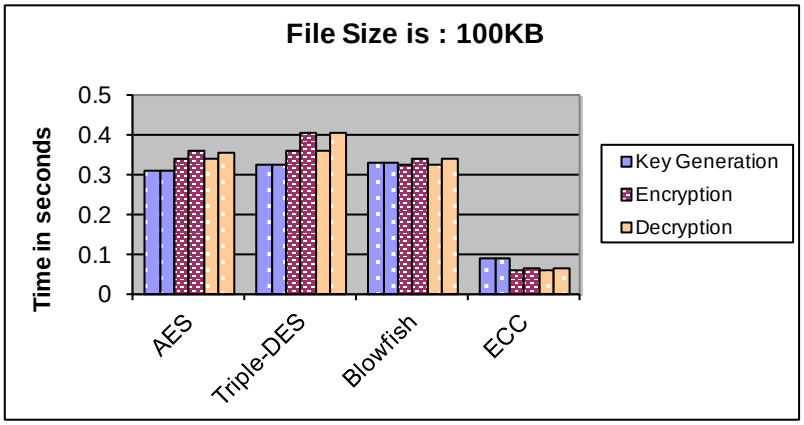

Figure 15. CPU Execution Time in Seconds (WinXP, 100KB)

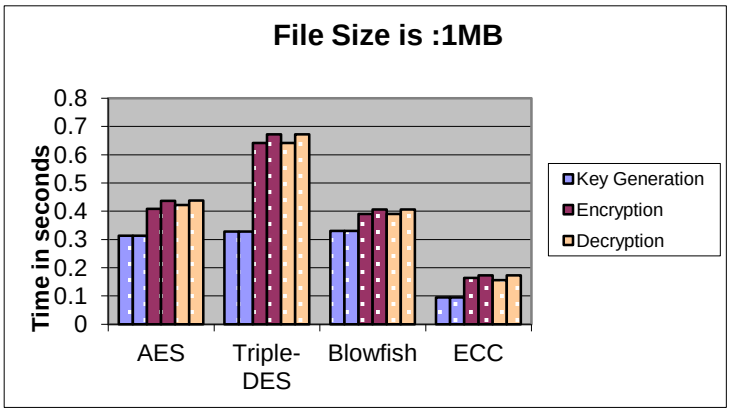

Figure 16. CPU Execution Time in Seconds (WinXP, 1MB)

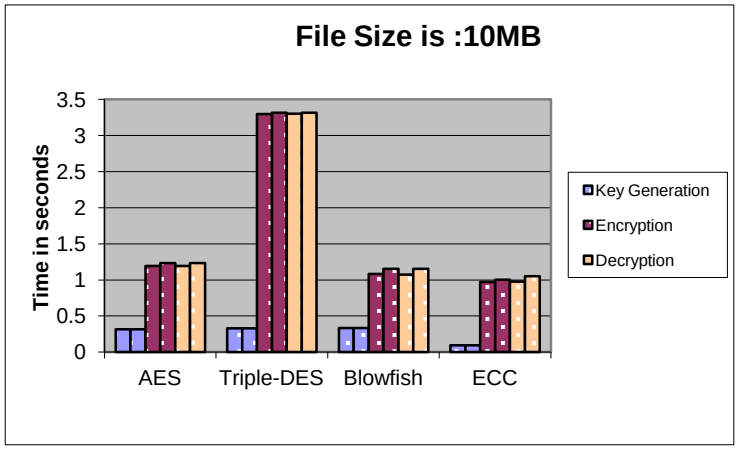

Figure 17. CPU Execution Time in Seconds (WinXP, 10MB)

Figures show that the ECC algorithm outperforms the other algorithms in all tests. Also, the key generation time is almost negligible compared to encryption/decryption time except when the file size is small in the ECC case. From the Figures of Linux, it is clear that the execution time is shorter for all algorithms than that for WinXP. Figures of the third test (Figures 15, 16, 17) of two different file contents (English only and hybrid) shows no significant difference in execution time but little shorter for English only file. For the ECC algorithm, one can notice that the encryption time is a little bit longer than decryption time regardless of file size or operating system; this can be explained since the public key is computed from the private key.

The CPU execution time for ECC is shorter than the CPU execution time of the other algorithms by the factors shown in Table 2 in WinXP and in Table 3 in Linux.

Table 2. Algorithms CPU Execution Time versus ECC Execution Time (WinXP)

	AES	***Triple-DES***	***Blowfish***
Encryption	25 %	240 %	11%
Decryption	22 %	238 %	8%

Table 3. Algorithms CPU Execution Time versus ECC Execution Time (Linux)

	AES	***Triple-DES***	***Blowfish***
Encryption	75%	331%	18%
Decryption	77%	335%	30%

For instance, under Linux, the execution time for ECC in encryption phase is shorter than AES by 75%.

5. Conclusions and Future Work

The performance evaluation of Elliptic Curve Cryptosystem (ECC) as an asymmetric block cipher algorithm and three symmetric block ciphers: Triple-DES, AES, and Blowfish were presented.

We used Java programming language, Java Cryptography Architecture (JCA) and Java Cryptography Extension (JCE) as an implementation tools. Performance evaluation of the above mentioned four algorithms based on CPU execution time is presented under WinXP and Linux operating system platforms.

The analysis of the experimental results shown in figures of section 4 indicates the following conclusions:

- The ECC outperform the other three algorithms in all tests and under WinXP and Linux.
- The key generation time in all algorithms is almost negligible compared to encryption/decryption time except when the file size is small in the ECC case.
- The CPU execution time under Linux platform is shorter for the four algorithms than that under WinXP.
- For the ECC algorithm, the encryption time is a little bit longer than decryption time regardless of file size or operating system used.
- Blowfish algorithm is the fastest among the three symmetric algorithms.
- ECC algorithm is the best in terms of speed, key size, and security strength.

In future, we would like to experiment with these and new algorithms. Areas of research include using different key sizes, block sizes, platforms as embedded system, and

using other factors for performance evaluation as security strength, ease of hardware implementation, and implementing the ECC algorithm using different elliptic curves.

Acknowledgements

I would like to express my gratitude to Yarmouk University, Irbid-Jordan for supporting the publication of this research work. This research was completed while I was on sabbatical leave from Yarmouk University. Also, I would like to express my gratitude to the Hashemite University – Jordan, where I spent my sabbatical leave.

References

[1] D. Sweet and M. P. Chair, "E-Commerce in Canada: Pursuing the Promise", Report of the Standing Committee on Industry, Science and Technology, **(2012)** May.
[2] M. S. Lan, M. Martin, B. Livshits and J. Whaley "Securing web applications with static and dynamic information flow tracking", PERM'08, San Francisco, California, USA, http://suif.stanford.edu/papers/pepm08.pdf, (Referenced 30/12/2012), **(2008)** January 7-8.
[3] R. S. Douglas, "Cryptography Theory and Practice", 3rd Edition, Chapman & Hall/CRC, **(2005)**.
[4] W. Stallings, "Cryptography and Network Security Principles and Practice", 8th Edition, Prentice Hall, 8th Edition, **(2009)**.
[5] E. Thigarajan and M. Gourishetty, "Study of AES and its Efficient Software Implementation Study of AES and its Efficient Software Implementation", http://www.kemt.fei.tuke.sk/predmety/KEMT411_ESM/_materialy/Diplomanti/AES/thiagarajan03study.pdf (Referenced 25/2/2013).
[6] N. A. Kofahi, "Performance Evaluation of AES/Triple-DES/Blowfish Ciphers under W2K and Linux operating system platforms", ABHATH ALYARMOK, Basic Sci. & Eng, vol. 15, no. 2, **(2006)**, pp. 265-285.
[7] CGI Group Inc, White paper, Public Key Encryption and Digital Signature: How do they work? http://www.cgi.com/files/white-papers/cgi_whpr_35_pki_e.pdf (Referenced 12/4/2013).
[8] J. Thakur and N. Kumar, "DES, AES and Blowfish: Symmetric Key Cryptography Algorithms Simulation Based Performance Analysis", International Journal of Emerging Technology and Advanced Engineering, vol. 1, no. 2, **(2011)** December.
[9] http://en.wikipedia.org/wiki/Cryptography. (Referenced 20/4/ 2013).
[10] O. M. Barukab, A. I. Khan, M. Sharief Shaik and M. V. Ramana Murthy, "Secure Communication using Symmetric and Asymmetric Cryptographic Techniques", I. J. Information Engineering and Electronic Business, vol. 4, no. 2, **(2012)**, pp. 36-42, http://www.mecs-press.org/ijieeb/ijieeb-v4-n2/IJIEEB-V4-N2-6.pdf.
[11] D. S. Abd Elminaam, K. H. M. Abdual and M. Mohamed Hadhoud, "Evaluating the Performance of Sysmmetric Encryption Algorithms", International Journal of Network Security, vol. 10, no. 3, **(2010)** May, pp. 216-222.
[12] S. P. Singh and R. Maini, "Comparison of Data Encryption Algorithms", International Journal of Computer Science and Communication, vol. 2, no. 1, **(2011)** January-June, pp. 125-127.
[13] N. A. Kofahi, "Java Implementation and Performance Evaluation of Some Cryptographic Ciphers under WinXP and Linux Operating System Platforms", Information and Knowledge Management Journal, vol. 3, no. 4, **(2013)**, pp. 45-56.
[14] M. Anoop, "Elliptic Curve Cryptography: An Implementation Tutorial", http://www.dkrypt.com/home/ecc. (Referenced 30/4/2013).
[15] Elliptic curve cryptography, http://en.wikipedia.org/wiki/Elliptic_curve_cryptography (Referenced 30/4/2013).
[16] P. P. Deepthi and P. S. Sathidevi, "New stream ciphers based on elliptic curve point multiplication", Journal of computer communications, vol. 32, no. 1, **(2009)**, pp. 25-33.
[17] R. Kumar and A. Anil, "Implementation of Elliptical Curve Cryptography", IJCSI International Journal of Computer Science Issues, vol. 8, Issue 4, no. 2, **(2011)** July.

[18] P. P. Deepthi, V. S. Nithin and P. S. Sathidevi, "Implementation and analysis of stream ciphers based on the elliptic curves", Elsevier (Computers and electrical Engineering), vol. 35, no. 2, **(2009)** March, pp. 300-314.
[19] http://en.wikipedia.org/wiki/Triple_DES. (Referenced 2/5/2013).
[20] H. O. Alanazi, B. B. Zaidan, A. A. Zaidan, H. A. Jalab, M. Shabbir and Y. Al-Nabhan, "New Comparative Study Between DES, 3DES and AES within Nine Factors", Journal Of Computing, vol. 2, no. 3, **(2010)**, pp. 152-157.
[21] http://en.wikipedia.org/wiki/Advanced_Encryption_Standard. (Referenced 2/5/2013).
[22] M. Anand Kumar and S. Karthikeyan, "Investigating the Efficiency of Blowfish and Rejindael (AES) Algorithms", I. J. Computer Network and Information Security, vol. 4, no. 2, **(2012)**, pp. 22-28.
[23] http://en.wikipedia.org/wiki/Java_cryptography (Referenced on 2/5/2013).
[24] M. Pistoia, "Java 2 Network Security", Prentice Hall, **(1999)**.
[25] J. Jaworski, "Java Security Handbook", Sams Publishing **(2000)**.
[26] http://java.sun.com/j2se/.
[27] http://java.sun.com/products/jdk/1.2/docs/guide/security/CryptoSpec.html.
[28] M. Pistoia, D. F. Reller Deepak Gupta, M. Nagnur and A. K. Ramani, "Java 2 Network Security", http://www.redbooks.ibm.com/redbooks/SG242109.html (Referenced 8/5/2013).
[29] L. Koved, M. Pistoia and A. Kershenbaum, "Understanding Java™ 2 Platform Security Permissions Practical Approach", Sun's 2001 worldwide java developer conference.
[30] D. Flanagan, "Java in Nutshell, 3rd Edition", O'REILY publishing, **(1999)**.
[31] http://bouncycastle.org/ (Referenced on 8/5/2013).
[32] http://www.cdc.informatik.tu-darmstadt.de/ (referenced 8/5/2013).

Author

Najib A. Kofahi is a professor of computer science at Yarmouk University (YU). He received his Ph.D. in computer science from the University of Missouri-Rolla (USA) in 1987. He was granted scholarships for MSc. and Ph.D. from YU in 1981. Currently he is on sabbatical leave from Yarmouk University and he is a faculty member in the department of computer sciences at YU – Jordan since 1987.

During his stay at YU he was the dean of the Faculty of Information Technology and Computer Sciences. He also worked as the chairman of the computer science department during the years 1990-1992. During his stay at yarmouk he worked on the computer science curriculum development since his appointment. Also, he worked extensively in the curriculum development at Philadelphia University-Jordan in the academic year 1993-1994 during which he worked as the chairman of science department, while on sabbatical leave from YU. In September 2000 he went on leave from YU for four years during which he worked as a visiting associate professor at King Fahd University of Petroleum and Minerals (FUPM). While at KFUPM, he led an online course development team to develop online course material for algorithms course. He has several journal and conference research publications in a number of research areas. His research interest focuses on E-Learning, operating systems, distributed systems, performance evaluation and computer and data security.