Performance Evaluation of AES/Triple-DES/Blowfish Ciphers under W2K and Linux Operating System Platforms

Najib A. Kofahl *

Received on July 31, 2005 Accepted for publication on April 2, 2006

Abstract

In this paper, the results obtained from Java implementation of three cryptographic algorithms on two operating system platforms, namely: Windows 2000 Professional (W2K) and Linux are presented. These algorithms are: Advanced Encryption Standard (AES), Triple Data Encryption Standard (T-DES) and Blowfish. The selected algorithms are all symmetric block ciphers. The performance of the three algorithms is compared based on the CPU execution time as well as the real (elapsed) time. The CPU execution time is broken further into user time and kernel (system) time. The evaluation of the performance of the three cryptography algorithms is done for secret key generation, encryption and decryption operations. Java and JCA (Java Cryptography Architecture) are used to implement the three ciphers. The results show that the Blowfish algorithm is the fastest, followed by the AES algorithm then the Triple-DES algorithm. The results of this work will support the selection of the best encryption algorithm in terms of speed and will help in capacity planning of the overall system. From the results obtained, we propose to either embed a security module within the processor to provide ultimate speeds for cryptography operations or to dedicate a special server, which is responsible for providing cryptographic services for highly secured environments.

Keywords: AES, Blowfish, Cryptography, JCA, Operating system, Performance evaluation, Symmetric ciphers, Triple-DES.

Introduction

In the past, securing sensitive information using encryption was only restricted to key governmental agencies and diplomats. These days, secure encryption on the Internet is the key to confidence for people wanting to protect their sensitive data/information, or doing E-Commerce or doing business online.

Ensuring the safety of data/information using encryption has been used in several important newly emerging applications including: E-Commerce, secure messaging, virtual private networks ...etc. Several medium size and large companies that have
Kofahl

proprietary or sensitive information are required to encrypt their entire data/information fearing that in the wrong hands this information could cause millions of dollars in damage.

Efficient implementation of cryptography algorithms plays an important role in the new era of information technology (IT). This is mainly due to the growing need for high-speed and highly secured communication channels [1]. Also, modern applied cryptography in the field of wireless communication networks demands high processing rate to fully utilize the available network bandwidth. In addition, it is important that any implementation should follow the variety and advancements in algorithms and standards.

Cryptography provides the mechanisms necessary to provide the following data/information security concerns:

- **Authentication**: the knowledge of who sent a message and to determine whether a user's identity is authentic.
- **Confidentiality**: ensuring that information is accessible only to those authorized to have access.
- **Integrity**: safeguarding the accuracy and completeness of information and processing methods.
- **Non-repudiation**: the sender cannot deny sending the message if there is a specimen signature.
- **Availability**: ensuring that authorized users have access to data/information and associated assets when required.

There are two general types of key-based encryption/decryption algorithms (also known as ciphers): asymmetric (or public-key) and symmetric (conventional or private key).

Asymmetric ciphers are designed to allow for encrypting data using one key and decrypting it using another key. Public-key algorithms are much slower than Symmetric algorithms.

Symmetric algorithms, on the other hand are designed in a way such that the same key is used to encrypt and decrypt the message. Hence, any two parties interested in encrypting/decrypting data have to use the same (secret) key generated for both encryption and decryption. Symmetric encryption was the only available option prior to the advent of Public Key encryption in 1976.

Symmetric algorithms can further be divided into two main categories: **block ciphers** and **stream ciphers** [1, 2]. Block ciphers are based on the principles of Feistel ciphers and operate on data in groups or blocks where the message is broken into blocks of bits, each of which is encrypted separately. Stream ciphers, on the other hand only operate on a single bit (i.e., a block of one bit size) at a time, which makes them more suitable for real time applications such as multimedia. Block ciphers seem to be
applicable to a broader range of applications than stream ciphers. It is easy to see that a
block cipher can be used in a way to make it operate as a stream cipher.

A symmetric encryption scheme has five major components as follows [1, 2, 3]:

- **Plaintext** - this is the text to which the encryption algorithm is applied.
- **Encryption Algorithm** - the algorithm that is applied to the plaintext to perform
  mathematical operations to conduct substitutions and transformations to the
  plaintext.
- **Secret Key** - This is the input for the algorithm as the key dictates the encrypted
  outcome.
- **Ciphertext** - This is the encrypted or scrambled text generated by applying the
  algorithm to the plaintext message using the secret key.
- **Decryption Algorithm** - This is the encryption algorithm in reverse. It uses the
  ciphertext, and the secret key to derive the plaintext message.

When using this form of encryption, it is essential that the sender and receiver have
a way to exchange secret keys in a secure manner. Knowing the secret key one can
figure out the algorithm, hence communications will be insecure.

Therefore, there is always a need for a strong encryption algorithm, so that if
someone were to have a ciphertext and a corresponding plaintext message, they would
be unable to determine the encryption algorithm.

In general, there are two methods of attacking conventional encryption - brute force
and cryptanalysis:

- **Brute force technique**: Using a method (computer) to find all possible combinations
  and eventually determine the plaintext message.
- **Cryptanalysis technique**: It is an attack on the characteristics of the algorithm to
deduces a specific plaintext or the key used. One would then be able to figure out the
plaintext for all text that continues to use this compromised setup.

Table 1 shows the average time required for exhaustive key search (brute force
technique) for some key lengths

<table>
<thead>
<tr>
<th>Key Size (Bits)</th>
<th>Number of Alternative Keys</th>
<th>Time required at 10^9 Decryption/µs</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>$2^{56} \approx 7.2 \times 10^{16}$</td>
<td>20 hours</td>
</tr>
<tr>
<td>112</td>
<td>$2^{112} \approx 3.9 \times 10^{34}$</td>
<td>$1.6 \times 10^{17}$ years</td>
</tr>
<tr>
<td>128</td>
<td>$2^{128} \approx 3.4 \times 10^{38}$</td>
<td>$5.4 \times 10^{18}$ years</td>
</tr>
<tr>
<td>168</td>
<td>$2^{168} \approx 3.7 \times 10^{50}$</td>
<td>$3.9 \times 10^{28}$ years</td>
</tr>
</tbody>
</table>
From Table 1, the AES-128 algorithm gives about $3.4 \times 10^{38}$ possible 128-bit AES keys compared with $7.2 \times 10^{16}$ possible 56-bit Blowfish keys. This makes AES more immune to "brute-force" exhaustive attacks as compared to the Blowfish algorithm, but less than the TDES algorithm.

There are several factors behind the selection of a particular encryption/decryption algorithm such as its strength and immunity. The running time of the algorithm is also another important factor that should be considered as well. Other factors include flexibility, memory requirement and software & hardware suitability. Encryption algorithms may consume a large amount of system resources for generating the secret key and for the actual work needed for encrypting or decrypting the data. If the running time of the algorithm is neglected then this might lead to jeopardizing the overall performance of the operating system. Because of this reason, the primary objective of this paper is to evaluate the selected encryption algorithms based on their running time including both execution time as well as the total real time.

Three symmetric block ciphers have been selected for our study, namely, the Advanced Encryption Standard (AES), the Blowfish, and Triple Data Encryption Standard (TDES or TDES or 3DES) [1, 4, 5, 6, 7]. These three block ciphers have been chosen because AES is now replacing the outdated Data Encryption Standard (DES), which has been a worldwide standard for more than 25 years. TDES resolves the flaws in DES which was the de facto standard. The blowfish is a fast cipher with a different structure and functionality.

The selected three block ciphers were implemented using Java and Java Cryptography Architecture (JCA). Java will enable the implemented program to be portable across platforms. Java was used by other researchers including [8] to implement other algorithms. Other encryption/decryption algorithms have been incorporated into several Java technology offerings.

A comparison of the processing times needed for both the kernel (system) and the user to generate the secret key, to encrypt and to decrypt the data will be recorded. Also, the real time, which includes in addition to the kernel and user times the I/O times, is recorded as well.

Evaluating the performance of ciphers becomes of interest to many researchers [9, 10]. AES origin, description, evaluation, implementation, attacks, software implementation, hardware implementation and performance evaluation are discussed in [4, 11-16]. The blowfish structure and characteristics is discussed in [4, 17, 18, 19], and finally, the TDES is well addressed in [4, 7].

Several authors were involved in research to achieve the maximum performance of operating systems [3]. The research includes benchmarks and a comparison to other famous benchmarks [3, 20, 21]. Other researches focused on the kernel's point of view and have been categorized into different areas. Performance evaluation of kernels has been explored in [22, 23]. Several performance analysis tools [24] and measurement techniques [25] have been discussed in the literature. Some of these measurement techniques were interested in measuring the hardware performance based on specific
kernels such as [26]. Other studies targeted the best application optimization reachable by measuring the application performance [27] or by optimizing the application [28].

Research efforts have also been accomplished in the area of operating system architecture and design [3, 29, 30] or concentrated on the overall system performance [31]. This type of research might be accomplished using an operating system simulator [32] or by using a normal PC operating system [33, 34]. The outcomes of these studies have helped enormously in both fine-tuning the operating systems and the hardware needed [35, 36].

The importance, necessity and future of security in the web and e-commerce were best studied in [37]. Cryptography and its implementations in applied fields are exposed and explored in [33, 38].

Section 2 introduces the three ciphers under consideration. An overview of JCA and the Java implementation of the three ciphers is discussed in section 3. In section 4, the implementation section, we describe the experiment setup and the implemented program using JCA and Java. Section 5 introduces the testing methodology and presents the results obtained from the implementation. Finally, section 6 concludes the study and points to the future work.

The Three Ciphers

Cryptography is widely applied to protect digital data. Nowadays, there are many kinds of cryptography and most of them require a secret key to encode digital data. In this section we briefly describe the structure of the three selected ciphers: the (AES) [4, 5], Blowfish [4, 6, 7, 8] and TDES [4, 7] that have been selected for Java and JCA implementation.

In 1997, the National Institute of Standards and Technology (NIST) issued a call for proposals for an encryption standard “the DES of the 21st century”, as a new official standard. The new standard is called the advanced encryption standard (AES). NIST required that AES should have security strength equal to or better than the TDES and significantly improved efficiency [1]. AES was approved by NIST in September 2000, and became effective on May 26, 2002 as a replacement for the out-of-date DES. It consists of four easily reversible different stages that make up a standard round [1]. The stages are iterated 10 times for a 128-bit key, 12 times for a 192-bit key, and 14 times for a 256-bit key. The four stages are as follows:

- Substitute bytes: A non-linear byte substitution that uses an S-box to perform a byte-by-byte substitution of the data block.
- Shift rows: A simple transformation that uses permutation that cyclically shifts (permutes) the bytes within the block.
- Mix columns: A transformation using substitution that makes use of arithmetic over GF (2^8). It groups 4-bytes together forming 4-term polynomials and 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. This is the only stage that uses the key.

The final round of both encryption and decryption consists only of three stages (i.e., no mix columns stage). The main steps involved in the encryption process in the AES are illustrated in Figure 1.

Figure 1. The main steps involved in the encryption process in the AES.

AES is a symmetric block cipher that supports combinations of key lengths of 128, 192, or 256 bits and block sizes of 128, 192, or 256 bits. With three different key sizes defined by the AES standard, the corresponding algorithms are named AES-128, AES-192 and AES-256.

On October 2, 2000, NIST announced that Rijndael has been accepted as the proposed algorithm for AES. Rijndael is a substitution-linear transformation network with 10, 12 or 14 rounds, depending on the key size. A data block to be processed using Rijndael is partitioned into an array of bytes, and each of the cipher operations is byte-oriented [5].

The Blowfish algorithm was designed by Bruce Schneier in 1993 to be simple, fast, compact and variably secure. It is a cipher with a different structure and functionality...
than the other two ciphers. It is a symmetric block cipher with a block size of 64 bits. The plaintext is divided into two 32-bit halves called, the left half \((LE_0)\) and the right half \((RE_0)\). \(LE_i\) means the left and right half of the data after round \(i\) has completed. Blowfish is a variable-length key of at most 448 bits and has a fair amount of acceptance in a number of applications. The secret key of Blowfish algorithm ranges from 32 bits to 448 bits. With 448 bits key, the Blowfish algorithm requires \(2^{448}\) combinations to examine all keys. The Blowfish algorithm consists of two parts: a key-expansion part and a data-encryption part. Key expansion converts the key into several sub key arrays totaling 4168 bytes. Data encryption occurs via a 16-round Feistel network. 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. No attacks are known against the 448-bits Blowfish algorithm. The Blowfish algorithm has many advantages. It is suitable and efficient for hardware implementation. Besides, it is unpatented and no license is required [19]. A complete definition of Blowfish algorithm is found in [1]. Figure 2 shows a block diagram of the blowfish algorithm.

Figure 2. Blowfish algorithm block diagram.
TDES simply extends the key size of DES by applying the algorithm three times in succession with three different keys as shown in Figure 3.

![Diagram of TDES](image)

**Figure 3.** The TDES as DES algorithm applied three times in succession with three keys

The combined key size in the TDES algorithm is thus 168 bits (3 times 56), which is beyond the reach of brute-force techniques. TDES was the answer to the security flaws of the DES without designing a whole new cryptosystem. It is a more complicated version of the DES algorithm. In DES, a 16-cycle Feistel system is used for encryption, with an overall 56-bit key permuted into 16 48-bit sub keys, one for each cycle. For decryption, an identical algorithm is used, but the order of sub keys is reversed. The left (L) and right (R) blocks are 32 bits each, yielding an overall block size of 64 bits. The hash function "f", specified by the standard using the so-called "S-boxes", takes a 32-bit data block and one of the 48-bit sub keys as input and produces 32 bits of output. Sometimes DES is said to use a 64-bit key, but 8 of the 64 bits are used only for parity checking, so the effective key size is 56 bits.

Since the TDES algorithm is based on the DES, modification of legacy software developed based on DES is very easy. It also has the advantage of proven reliability and a longer key length compared to the DES. A two-key version of TDES exists where the cipher text, C is obtained as:

\[ C = EK_1[DK_2[EK_1[P]]] \]

Where,

- \( P \) is the plain text,
- \( EK_1 \) means apply the encryption algorithm using Key(1), and
- \( DK_1 \) means decrypt using Key(1)
The Triple data encryption algorithm (TDEA or 3DEA) and AES will coexist in FIPS approved algorithms.

**Java Cryptography Architecture**

This section gives an overview of Java Cryptography Architecture (JCA) used in our implementation. 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 functionality into their programs [40, 41, 42, 43]. It encompasses the parts of the Java 2 SDK Security API related to cryptography, as well as a set of conventions and specifications. It also includes a "provider" architecture that allows for multiple and interoperable cryptography implementations. The JCA was designed around two principles, namely: Implementation independence & interoperability; and algorithm independence & extensibility [42, 43].

Implementation independence and algorithm independence means that cryptographic services can be used without worrying about the implementation details or even the algorithms. When complete algorithm-independence is not possible, the JCA provides standardized, algorithm-specific APIs. When implementation-independence is not desirable, the JCA lets developers indicate a specific implementation.

Due to its implementation interoperability feature, JCA allows various implementations to work with each other, use each other's keys, or verify each other's signatures. This would mean, for example, that for the same algorithms, a key generated by one provider would be usable by another. Algorithm extensibility means that new algorithms that fit in one of the supported engine classes can be added easily. The Java Cryptography Extension (JCE) extends the JCA API to provide a framework and implementations for encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms [42].

Starting with Java 2 Software Development Kit (SDK), Standard Edition (J2SE) Version 1.4.2, the Java Cryptography Extension (JCE) was integrated with the SDK and the Java Runtime Environment (JRE). Therefore, it is no longer necessary to install the JCE optional package, as support for cryptography is now available as part of J2SE.

**Implementation**

In this section, we give a general overview of the experimental set up and how the selected ciphers are implemented using JCA and Java programming language.

**Experiment Setup**

The three ciphers were run on Pentium III processor with a CPU speed of 800 MHz and a total of 256 MB RAM. The algorithms were tested on two different operating system platforms. The first test was conducted under Windows 2000 Professional operating system and the second test was conducted under Mandrake/Linux 8.2 Kernel version 2.4.18.

The three encryption/decryption algorithms are performed on the same file of size 10 MB. Java version 1.4.2 has been used for implementing the three ciphers. The
algorithms were tested using key sizes of 128 bits for the AES, 56 bits for Blowfish, and
112 bits for the TDES algorithm.

The experiment was repeated ten times for each algorithm and for each of the three
operations, the secret key generation, the encryption operation and decryption operation.
The average of the ten runs for each operation was recorded in the corresponding table.

The Implemented Program

A program was developed to implement the three ciphers using Java and JCA. The
program is constructed into 3 modules. The first module is responsible for generating
secret keys to be used for encrypting and decrypting messages. A secret key has to be
generated for each cipher. The second module is the encryption module that takes a
secret key generated for the same cipher algorithm and uses this key to encrypt a
message (e.g. an AES secret key to encrypt an AES message, a blowfish secret key to
encrypt a blowfish message, etc.). The decryption module is the third module that takes a
secret key and decrypts a message. The message will be decrypted successfully only if
the key used for decryption is the same key used for encrypting the message initially.

Using this methodology in the design of the encryption/decryption program, allows
for timing each operation (key generation, encryption and decryption) separately. The
program also does not interact with the user while it is running (prompting where to save
the secret key or the name of the file to encrypt), instead it expects all of the required
parameters to be supplied when invoking the application. This is necessary to obtain
accurate measurements of the time spent for each operation without being influenced by
the time it took the user to interact with the program. The following packages and classes
are used in the implementation:

```
javax.crypto.Cipher;
javax.crypto.KeyGenerator;
javax.crypto.CipherInputStream;
javax.crypto.SecretKey;
javax.crypto.spec.SecretKeySpec;
java.security.Key;
java.security.AlgorithmParameters;
java.security.SecureRandom;
java.security.NoSuchAlgorithmException;
java.io.*;
```

A brief description of the three main modules using Java and JCA follows.
Secret Key Generation
Package: javax.crypto
Class: KeyGenerator
Create a secret key to be used for encryption and decryption by using the KeyGenerator class and supplying it with a string to select the algorithm (e.g. blowfish) as in the following code snippet:

```java
KeyGenerator KG = KeyGenerator.getInstance(algorithm);
Key key = KG.generateKey();
```

Encryption Module

```java
// initialize the Cipher with the secret key
cipher = Cipher.getInstance(algorithm);
cipher.init(Cipher.ENCRYPT_MODE, key);
```

// creating the encrypted cipher stream
```java
File fileIn = new File(fileName);
FileInputStream fis = new FileInputStream(fileIn);
cipherInputStream = new CipherInputStream(fis, cipher);
```

byte[] b = new byte[1024];

// writing the encrypted data to the output file
```java
file = new File(fileOut);
FileOutputStream fos = new FileOutputStream(file);
int i = cipherInputStream.read(b);
fos.write(b, 0, i);
```
4.2.3 Decryption Module

// initialize the Cipher with the secret key
cipher = Cipher.getInstance(algorithm);
cipher.init(Cipher.DECRYPT_MODE, key);

// creating the decrypted cipher stream
File fileIn = new File(fileName);
FileInputStream fis = new FileInputStream(fileIn);
cipherInputStream = new CipherInputStream(fis, cipher);

byte[] b = new byte[1024];

// writing the decrypted data to the output file
file = new File(fileOut);
FileOutputStream fos = new FileOutputStream(file);
int i = cipherInputStream.read(b);
fos.write(b, 0, i);

Testing Methodology and Experimental Results

In the experiment, both the elapsed time (real time or wall-clock time) and the CPU
time are measured. The elapsed time takes into account everything such as disk or
memory access, idle time, I/O, operating system overheads, etc. This gives a useful
number but not for the comparison purposes we intend. The CPU executing time (or the
CPU time) is the time the CPU spends computing a task and doesn’t consider the time
spent waiting for I/O or running other programs. The CPU time is further broken down
to system (kernel) time and user time. The system time is the time spent executing
instructions in the kernel (kernel mode) on behalf of the user program. The user time is
the time spent executing instructions in the user program (user mode). For programs
running on dedicated systems and spending most of their time doing computation, the
elapsed time and CPU time should be approximately equal. Discrepancies between the
CPU time and elapsed time will usually occur in cases where a program is paging (or
swapping) due to insufficient memory on the system.
Performance Evaluation of AES/Triple-DES/Blowfish Ciphers under Windows and Linux Operating System Platforms

The `time` Linux utility was used to measure the elapsed time, system time and user time of the implemented program running on the Linux platform. The results obtained are shown in Table 2.

**Table 2. Algorithms Processing Time in Seconds on Linux**

<table>
<thead>
<tr>
<th></th>
<th>AES User</th>
<th>AES Kernel</th>
<th>AES Real</th>
<th>Triple-DES User</th>
<th>Triple-DES Kernel</th>
<th>Triple-DES Real</th>
<th>Blowfish User</th>
<th>Blowfish Kernel</th>
<th>Blowfish Real</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Generation</td>
<td>0.86</td>
<td>0.026</td>
<td>1.301</td>
<td>0.822</td>
<td>0.029</td>
<td>1.257</td>
<td>0.795</td>
<td>0.036</td>
<td>1.234</td>
</tr>
<tr>
<td>Encryption</td>
<td>2.493</td>
<td>0.43</td>
<td>4.928</td>
<td>9.748</td>
<td>0.475</td>
<td>26.165</td>
<td>2.31</td>
<td>0.372</td>
<td>6.37</td>
</tr>
<tr>
<td>Decryption</td>
<td>2.406</td>
<td>0.461</td>
<td>4.158</td>
<td>11.531</td>
<td>0.547</td>
<td>28.801</td>
<td>2.329</td>
<td>0.366</td>
<td>6.458</td>
</tr>
</tbody>
</table>

Figure 4 illustrates these results (the user time, kernel time and real time) for the three algorithms under Linux environment.

![Mandrake/Linux 8.2 Diagram](image)

**Figure 4.** Linux User, kernel and real times in seconds for the three algorithms

To perform the measurements under Windows platform, a small program was written to record the user, kernel, and the total (real) time, which is the summation of the user time, kernel time, and I/O time. The program was written in C# language and it uses the `System.Diagnostics.Process` class included within the .Net Framework. The results obtained for the three algorithms are shown in Table 3.

**Table 3. Algorithms Processing Time in Seconds under Windows 2000 Pro.**

<table>
<thead>
<tr>
<th></th>
<th>AES User</th>
<th>AES Kernel</th>
<th>AES Real</th>
<th>Triple-DES User</th>
<th>Triple-DES Kernel</th>
<th>Triple-DES Real</th>
<th>Blowfish User</th>
<th>Blowfish Kernel</th>
<th>Blowfish Real</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Generation</td>
<td>1.01</td>
<td>2.59</td>
<td>1.07</td>
<td>1.11</td>
<td>2.81</td>
<td>0.69</td>
<td>0.65</td>
<td>2.14</td>
<td></td>
</tr>
<tr>
<td>Encryption</td>
<td>2.06</td>
<td>2.20</td>
<td>6.47</td>
<td>5.87</td>
<td>6.29</td>
<td>22.67</td>
<td>1.68</td>
<td>1.82</td>
<td>6.38</td>
</tr>
<tr>
<td>Decryption</td>
<td>2.08</td>
<td>2.06</td>
<td>6.27</td>
<td>5.87</td>
<td>6.33</td>
<td>22.40</td>
<td>1.70</td>
<td>1.79</td>
<td>6.57</td>
</tr>
</tbody>
</table>

277
Figure 5 illustrates these results (the user time, kernel time and real time) for the three algorithms under Windows 2000 Professional environment.

![Windows 2000 Professional](image)

**Figure 5.** User, kernel and real times in seconds for the three algorithms under Windows 2000

Both hardware and software implementation of encryption/decryption algorithms are available [1, 4, 5, 8, 13, 44]. However, implementing encryption algorithms in software have a tremendous impact on the processor resulting in degrading its overall performance.

Table 4 shows the CPU execution time obtained from the implemented program when run under Linux environment.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AES</th>
<th>Triple-DES</th>
<th>Blowfish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Generation</td>
<td>0.886</td>
<td>0.851</td>
<td>0.831</td>
</tr>
<tr>
<td>Encryption</td>
<td>2.923</td>
<td>10.223</td>
<td>2.682</td>
</tr>
<tr>
<td>Decryption</td>
<td>2.867</td>
<td>12.078</td>
<td>2.695</td>
</tr>
</tbody>
</table>

**Table 4.** CPU Execution Time in Seconds (Linux)

Figure 6 illustrates the CPU execution time, which includes both the system time and user time for the three algorithms under Linux environment.
Performance Evaluation of AES/Triple-DES/Blowfish Ciphers under W2K and Linux Operating System Platforms

![Graph showing CPU execution time in seconds.

Figure 6. CPU executing time in seconds (Linux)

The CPU execution time which includes both, system and user time for the three algorithms under Windows 2000 Professional is shown in Table 5 and illustrated in Figure 7.

<table>
<thead>
<tr>
<th></th>
<th>AES</th>
<th>Triple-DES</th>
<th>Blowfish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Generation</td>
<td>2.01</td>
<td>2.18</td>
<td>1.34</td>
</tr>
<tr>
<td>Encryption</td>
<td>4.26</td>
<td>12.16</td>
<td>3.5</td>
</tr>
<tr>
<td>Decryption</td>
<td>4.14</td>
<td>12.20</td>
<td>3.5</td>
</tr>
</tbody>
</table>

Table 5. CPU Execution Time in Seconds (Windows 2000 Professional)

![Graph showing CPU execution time in seconds.

Figure 7. CPU executing time in seconds (W2K Professional)
Kofahl

From Figures 6 & 7, it is clear that the key generation operation for all algorithms is almost identical. The major difference between these algorithms is the CPU time spent while performing the encryption and decryption operations.

It is clear also from Figures 4 and 5 that the AES algorithm has the shortest total (real) time followed by the Blowfish algorithm and then the TDES algorithm under both Linux and Windows platforms. However, regarding the CPU time, it is clear from Figures 6 & 7 that the Blowfish algorithm is the fastest followed by the AES algorithm and then the TDES algorithm. One reason that the Blowfish algorithm is the fastest is due to the mechanism it follows for ciphering data. It is obvious that because of the increased complexity incorporated within the TDES, which is needed to address the security issues that exist in the DES algorithm, is degrading the performance in terms of the CPU time.

Regarding the CPU time under Linux operating system platform, the results indicate that the Blowfish algorithm is faster than the AES algorithm by 10% for encryption and 6.4% for decryption. However, the blowfish algorithm is faster than the TDES by 281% for encryption and 348% for decryption.

For the CPU time under Windows 2000 platform, the Blowfish algorithm is faster than the AES algorithm by 21.7% for encryption and 18.3% for decryption. However, the blowfish algorithm is faster than the TDES by 247% for encryption and 248.5% for decryption.

In order to increase the speed of these algorithms we propose to have a built-in cipher module within the processor. This will eliminate any overheads introduced when implementing encryption algorithms in software. Another proposal is to have a dedicated server responsible for secret key generation and distribution, encryption, decryption, authentication ... etc. This approach is very useful for highly secured environments (e.g. the military).

Conclusions and Future Work

In this study we presented an implementation of three symmetric block encryption algorithms using Java and JCA. The main objective was to evaluate the performance of these algorithms in terms of CPU execution time and total run time. The measurements were performed on two operating system platforms, namely Windows 2000 Professional and Linux. The analyzed time was the CPU execution time for generating the secret key, encryption and decryption on a 10MB file. The results showed that the AES algorithm was the fastest followed by the blowfish algorithm then the TDES algorithm. The TDES algorithm was slow in its performance due to the added complexity and security it has over the DES algorithm.

The results obtained play a major role on selecting the appropriate encryption algorithm to use for software implementations. The author supports the idea of using a built-in module within the processor dedicated for security considerations. This will have the advantage of relieving the processor from the overhead associated with implementing encryption algorithms in software. Having a built-in module within the
Performance Evaluation of AES/Triple-DES/Blowfish Ciphers under Win and Linux Operating System Platforms

processor will provide speed for encryption algorithms used in e-commerce and m-commerce which will be heavily demanded in the near future. Using a dedicated server is another possible proposal. This server will have the responsibility of controlling and performing all security tasks such as key generation and distribution, encryption and decryption. This server will prove beneficial in highly secured environments.

Other implementations of the three algorithms undertaken in this study are a future research interest of the author. It is important to compare different implementations with the JCE and Java implementation. We are planning to use both "C" as a high level programming language and Assembly as a low level machine language, which will enable us to optimize computation intensive parts of the ciphers. The implementation could also include dedicated types of hardware that will provide at the hardware level random number generator and built-in co-processors.

Also, as a future work, we propose two interesting ciphers to be investigated further. The first cipher belong to the Crossing Over Systems (COS) cipher family. COS ciphers are block ciphers, built only from stream ciphers primitives (nonlinear feedback shift registers and Boolean functions). These ciphers have been especially designed to yield a very high encryption security and a very high encryption speed with an internal secret 256-bit key [45].

The second cipher is the elliptic curves cryptography [46]. Elliptic Curve Cryptosystem (ECC), which was originally proposed by Niel Koblitz and Victor Miller in 1985, is seen as a serious alternative to RSA with a much shorter world length. ECC with a key size of 128-256 bits is shown to offer equal security to that of RSA with key size of 1-2Kbits. To date, no significant breakthroughs have been made in determining weaknesses in the ECC algorithm, which is based on the discrete logarithm problem over points on an elliptic curve.

Advanced Encryption Standard

Triple Data Encryption Standard (T-DES) (AES)

Blowfish

نجيب عبد الكريمن الكوفجي

ملخص

نال هذا البحث النتائج التي تم الحصول عليها لمحم أداء ثلاث خوارزميات تشفير البيانات (Java Cryptography Architecture) باستخدام لغة Java وبنية جاكا تشفير البيانات في بينتي نظم تشغيل ويندوز 2000 ولينكس.
Kofahi

Advanced Encryption Standard

Advanced Encryption Standard (AES) is a symmetric-key algorithm for the encryption of electronic data.

References


Performance Evaluation of AES/Triple-DES/Blowfish Ciphers under Windows and Linux Operating System Platforms


[16] [http://www.seeininf.net/ualablab/2/aes-performance.pdf]


283


Performance Evaluation of AES/Triple-DES/Blowfish Ciphers under Win32 and Linux Operating System Platforms


285