A Real-Time Performance Analysis Model for Cryptographic Protocols

Amos Olagunju, Jake Soenneker
Computer Networking and Applications Program
St. Cloud State University
aoolagunju@stcloudstate.edu

ABSTRACT

Several encryption algorithms exist today for securing data in storage and transmission over network systems. The choice of encryption algorithms must weigh performance requirements against the call for protection of sensitive data. This research investigated the processing times of alternative encryption algorithms under specific conditions. The paper presents the architecture of a model multiplatform tool for the evaluation of candidate encryption algorithms based on different data and key sizes. The model software was used to appraise the real-time performance of DES, AES, 3DES, MD5, SHA1, and SHA2 encryption algorithms.

Keywords: Encryption algorithm, security, performance measurement, software engineering.

1. INTRODUCTION

A symmetric encryption algorithm uses the same key for both encryption and decryption. This method is also known as single key, secret key, or private key encryption because only one key is required during the encryption and decryption operations.

The Digital Encryption Standard (DES) was developed by the American National Institute of Standards and Technology based on an encryption algorithm that was submitted by Horst Feistel at IBM Research. After a few modifications, the original block cipher algorithm, called Lucifer, was accepted as the cornerstone of the DES on January 15, 1977. Although DES was initially adopted as a national standard for encrypting data used in the US federal government, DES quickly became a world-wide standard for symmetric encryption.

Like all other encryption methods, DES consists of both an algorithm and a key. The DES key is made up of eight bytes of data. Each byte, in turn, comprises eight bits—seven data bits and one parity bit—for a total of 56 bits of key data and 8 bits of parity data. The parity data allows systems using the key to ensure that the data used to make up the key is not corrupted. The algorithm breaks the plaintext data into blocks of 16 bits. DES offers four distinct modes of operations to provide varying levels of complexity and protection for data encryption. Unfortunately, because DES uses a 56-bit key to perform encryption, there is sufficient computing power to crack the key. Triple DES (3DES) was created to overcome the inherent weakness of DES. But, it takes three times longer for 3DES to encrypt and decrypt. Therefore, the Advanced Encryption Standards (AES) that supports variable key and block lengths was created.

Public-key encryption, also known as asymmetric encryption, seeks to solve the encryption key exchanges problem by incorporating a method for securely sharing the necessary key information. The drawback to this family of algorithms, however, is that they are typically slower than symmetric encryption algorithms. The literature provides in-depth concepts of symmetric and asymmetric encryption [4].

This research focused on the design and implementation of a multiplatform software as a model for testing encryption algorithms. The research results are compared to other major security protocol evaluations in the literature, and to the performance of Open Secure Socket Layer's encryption software. Henceforth, we present the features of a Cryptographic Protocol Performance Program (C3P). C3P is console interface software that supports the evaluation of encryption protocols. The design, implementation, and experimental results of C3P can be replicated for all well-designed cryptographic algorithms.

The performance of security protocols has been a subject of attention in the literature. Yet, published practical software tools for assessing the performance of encryption algorithms against real-world data and security requirements are rare to find. The performance reviews of several popular encryption protocols such as, the RC6, DES, 3DES, AES, Blowfish, and Rjindael have been evaluated [1]. These security encryption algorithms have been evaluated on a 2.4GHz IV laptop, in light of their capabilities to encrypt different types of data (text, audio, and video). Though the research results in the literature are relevant, there is no clear discussion of the data used for the comparative evaluation of encryption algorithms [1]. Consequently, we have designed and implemented C3P as a tool for testing symmetric and asymmetric encryption algorithms.

The literature reveals that AES is more efficient than the comparable encryption protocols in its group [2]. The AES's throughput was proclaimed as more efficient compared to the counterpart symmetric algorithms. In fact, AES is also known to have better resistance against brute force attacks than other security encryption protocols. Although a third party application (Crypto++) was used for the testing the performance of security protocols in [2], it provides a good source of information.

Mono is a platform initiated by Novell that supports a compiler for generating and executing a Common Intermediate Language

byte code and a class library. Mono is an open source implementation of Microsoft's .NET Framework based on the ECMA/ISO standards for C# and the Common Language Runtime [5]. Mono is crucial for converting executable codes into a Linux-readable binary –for the execution of the C3P on a Linux machine. Essentially, on a Linux platform, Mono is installed and used to run the binary executable code. Mono's C# Compiler can also be installed on a Windows platform and used to execute C# classes. The compiler "gmcs" was used to compile the C3P, to overcome the requirements of generating a binary executable code in C# 4.0. The "gmcs" generates an executable binary code.

2. C3P DESIGN

The goal of the C3P software was to design a multiplatform for investigating the effects of the different data and key sizes on the performance of encryption algorithms. Consequently, efficiency and lightweight were major factors in the design of C3P. The C# platform with the existing encryption algorithmic classes made it easier to design the C3P software.

The architectural overview of the C3P in Figure 1 consists of a User Interface (UI) with a main menu, a protocol menu, and a menu for displaying the encryption performance results. The UI was constructed as a prevailing class for each individual encryption page. All interface classes have access to functions for printing the header and footer. Each screen utilizes these functions to build the basic overlay of a highly efficient interface in a standard format for each security protocol menu. The dynamic implementation of the screens makes it easy to enhance the C3P with features for future investigations of the performance of security protocols. The schematic view of the interacting program modules are displayed in Figure 2. Appendix A contains the snapshots of the major menus of the C3P.

The C3P begins execution through the "Runtimes" module prior to invoking the main menu with capabilities for activating any of the encryption protocol menu classes in this research. The inherited menu classes have access to the "Engine.Stopwatch" and "Engine.RandomGenerator" classes. These public static procedures allow the menu classes to derive the execution time. The latter engine also facilitates the generation of non-pseudo random strings.

The console environment of the C3P does not require any conversion of the Windows forms, and provides simplicity of project design and usability on non-GUI operating systems. Although it is possible to transport Windows forms to Linux using Mono, this encryption evaluation approach is not recommended because of its adverse testing effects on performance.

3. C3P IMPLEMENTATION

The need exists to use tools and libraries that conform to the Common Language Infrastructure for developing platform-independent software applications. Mono and DotGNU were two candidate .NET infrastructure conversion software bundles deemed relevant to the implementation of the C3P. Mono offers the .Net conversion to Common Language Runtime (CLR), while the project code base of the DotGNU provides a hundred percent Common Language Specification amenable in a class library [5]. However, Mono offers more advantages over the Dot GNU in the implementation of the "System.Security.Cryptography" class. Mono is easier and more suited to code conversion and the overall design of class-based software architecture. Moreover Mono offers a useful documentation on the compilation of the CLR binary.

The implementation of the C3P focused on streamlining every function, method, or class to avoid negating the true performance of the actual encryption algorithms. C# with existing cryptographic libraries [6, 7, 8] was used to implement the C3P. The codes of each data encryption algorithm were implemented in C3P within the limitation of the hardware. However, the codes were designed to be as fast as the operations of the original encryption algorithms.

The progression of instructions for the hash functions MD5, SHA1, and SHA2 in the literature [8] is:

a) Create an encoder object to encode the string.
b) Create a service provider for the appropriate hash.
c) Create a new array of bytes to capture the value of computed hash of the encoded string from the service provider.
d) Return the created array to the calling object.

The process for generating the AES in Cipher Block Chaining mode available in [6] is:

a) Create an array of bytes of the encoded 16 ASCII characters (the Initial Vector) of the AES algorithm.
b) Generate a password for the overridden encoded values (Plaintext, Password, and Salt) of the constructor.
c) Create a Rijndael managed symmetric key object.
d) Set the key object's cipher mode to the appropriate mode –Cipher Block Chaining was used in C3P.
e) Create the encrypting object and use a memory stream to write the blocks to a new array.
f) Close memory streams for efficiency and cleanup.

The course of actions for DES and 3DES (TDEA) defined in [7] is:

a) Encode a random string of 8 bytes long.
b) Create a DES Service provider object.
c) Create a new Memorystream object for the Cryptostream object to read into.
d) Write with a Streamwriter, the original string with the Cryptostream attached.
e) Clean up all objects. Close all memory sockets.

The MSDN library offers reference information, sample codes and technical articles for the use of Microsoft® tools, products and technologies in multiplatform software development. The MSDN project was valuable in building the C3P.

4. TESTING AND RESULTS

A 2.4GHz Intel Pentium 4, with single core processor similar to laptops normally used to test the performance of security

protocols in the literature, was applied to investigate the performances of the various security protocols. We recognize that machines with more processing power typically provide faster encryption performance results [1]. In this project, the performance of the C3P was compared to the encryption performance of OpenSSL [9] on Ubuntu 9.1. The milliseconds required by the different encryption algorithms to encrypt various data sizes are illuminated in Figure 3. The three data sizes of 20,527 bytes, 127,325 bytes, and 232,298 bytes used in this research are the mid-points of experimental data sizes reported in the literature [2]. Moreover, these data sizes provide reliable test bed for comparing the performances of the C3P and OpenSSL.

The experimental results derived from the C3P in Figure 3 exhibit similar patterns to those generated from the OpenSSL data. AES, DES, and 3DES all show similar in encryption times for small data sizes. However, there is a disparity in the encryption times as the data size increased. This result is not surprising given that AES requires the password to be generated, and consequently more encryption time as the data size increased. 3DES [TDEA] requires approximately triple the time it take the ordinary DES to encrypt data on an average data scale. The results are not surprising because they are consistent with the findings in the literature [2]. Figure 3 also illuminates the execution results of the hash functions on the C3P. The performance of SHA1 was slightly better than that of MD5, but SHA2 exhibited the lowest runtime performance. As the data size grows, the SHA2 tends to exhibit an exponential-like runtime growth.

The milliseconds required by the different encryption algorithms to encrypt various data and key sizes are presented in Figures 4, 5 and 6 for AES, 3DES and SHA2 respectively. Figure 4 reveals a noteworthy change in the encryption performance of the AES as the key size changes with the data size. The larger the data size, the more the effects of the key size on the AES encryption performance. This result is not surprising given that the OpenSSL exhibited a parallel outcome. Clearly, there is a striking difference in the encryption times of AES when the key size is doubled from 128 bits to 256 bits as the data size increases.

Figure 5 demonstrates that the key size insignificant effect on the encryption performance of 3DES as the data size grows. The experimental results from OpenSSL also attested this claim. In fact 5,807,450 bytes were encrypted in 3 seconds with 3DES-128 bits key –the amount of time to encrypt 5,780,580 bytes with 3DES-192 bits key. These results explain the minimal effects of modifying the key size from 128 bits to 192 bits on the encryption performance of 3DES.

Figure 6 shows that data encryption with SHA2-256 is faster than data encryption with both SHA2-384 and SHA2-512, as the data size increases. As the key size increases from 256 bits to 384 bits there is a remarkable amplification of the encryption time. But, the change of key size from 384 bits to 512 bits slightly increases the data encryption time. This observation is not surprising because the OpenSSL results also revealed no significant difference between the encryption performance of SHA2-384 and SHA2-512.

The coefficients of the regression equations for forecasting the times it take to encrypt bytes of information using the DES, MD5, AES and SHA algorithms are displayed in Tables 7, 8 and 9. Note that the coefficient of determination (R-Sq) of each equation is reasonably high. That is, the models for predicting the runtimes of the encryption algorithms are reliable. However, the runtimes derived from any of these prediction equations will depend on the speed of the CPU as illustrated in Tables 10 and 11. Notice that the execution time decreased by approximately 25 percent as the CPU speed increased from nearly 2 GHz to 2.66 GHz. The equation for forecasting the runtime (T) it takes to encrypt bytes (B) using the intercept (α), slope (β) and CPU speed (CS in MHz) is $T = (\alpha + \beta(B)) * 2405/CS$, where $\beta = S_{TB}/(S_B)^2$, α = Average(T) – β* Average(B), S_{TB} and $(S_B)^2$ are the covariance of T and B, and the variance of B respectively. Tables 12, 13 and 14 show the respective runtime data generated by a 2405 MHz CPU and used to derive the regression equations for DES, MD5, AES and SHA. As illuminated in Figure 15, the runtimes exhibited a linear pattern.

When the C3P was used to encrypt 500,000 random bytes 5 times with DES on a personal computer with a processor clock speed of 2659 MHz, the average time was 100 milliseconds. The graphical display of the runtimes is shown in Figure 16.

5. CONCLUSIONS

We have designed and implemented the C3P as replica of console-driven platform-independent software for testing the performance of security protocols. The current C3P implementation only supports a 32-bit long integer. The C3P supports the specification of data files to be encrypted. The C3P provides a graphical display of the encryption runtimes when the number of iterations is specified for the random bytes to be encrypted. The C3P can be easily modified to support decryption. The C3P architecture is generic and flexible to support the performance evaluation of emerging novel encryption protocols. The C3P is a valuable instructional tool for courses in cryptography, advanced network programming and software engineering. The C3P is a priceless tool for investigating the tradeoffs between enhanced securities versus processing performance.

6. REFERENCES

[1] D. S. Abd Elminaam, H. M. Abdual Kader & M. M. Hadhoud, "Evaluating the Performance of Symmetric Encryption Algorithms," *International Journal of Network Security, vol.10, no.3, pp. 213-219, 2010*. Retrieved Dec 5, 2010, from http://ijns.femto.com.tw/contents/ijns-v10-n3/ijns-2010-v10-n3-p213-219.pdf

[2] A. Al Tamimi, "*Performance Analysis of Data Encryption Algorithms,"* 2010. Retrieved Dec 5, 2010, from http://www1.cse.wustl.edu/~jain/cse567-06/ftp/encryption_perf/index.html

[3] *DotGNU Project Development Site.* (2010). Retrieved Dec 8, 2010, from http://www.gnu.org/software/dotgnu/

[4] D. Mackey, *Web Security for Network and System Administrator, 2003*, Boston, MA: Thomson Learning, Inc.

[5] *Mono Development Page*, 2010. Retrieved Dec 2, 2010, from http://www.mono-project.com

http://msdn.microsoft.com/en-us/library/system.security.cryptography.des.aspx

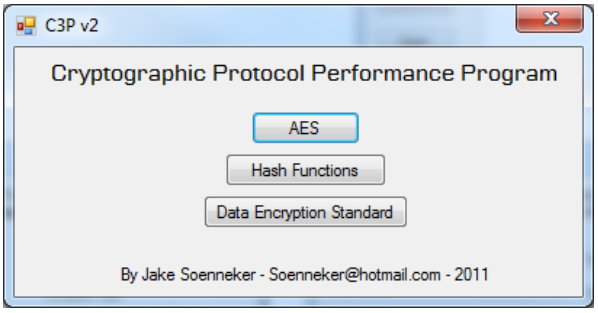
[7] *MSDN Library – 3DES Encryption*, 2010. Retrieved Dec 1, 2010, from http://msdn.microsoft.com/en-us/library/system.security.cryptography.tripledes.aspx

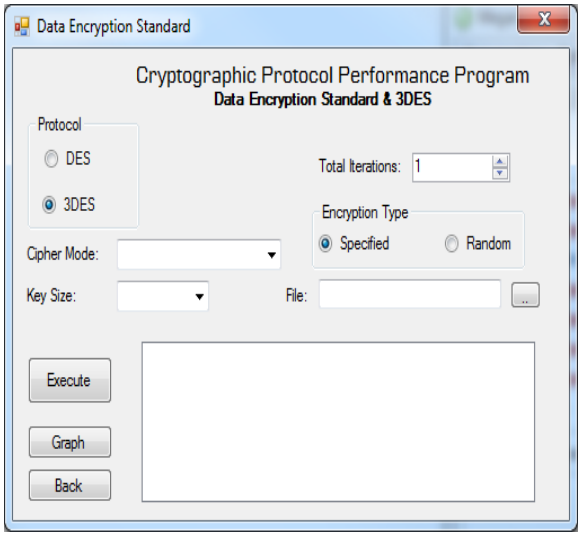
[6] *MSDN Library – DES Encryption and related Cryptographic Protocols*, 2010. Retrieved Dec 2, 2010, from

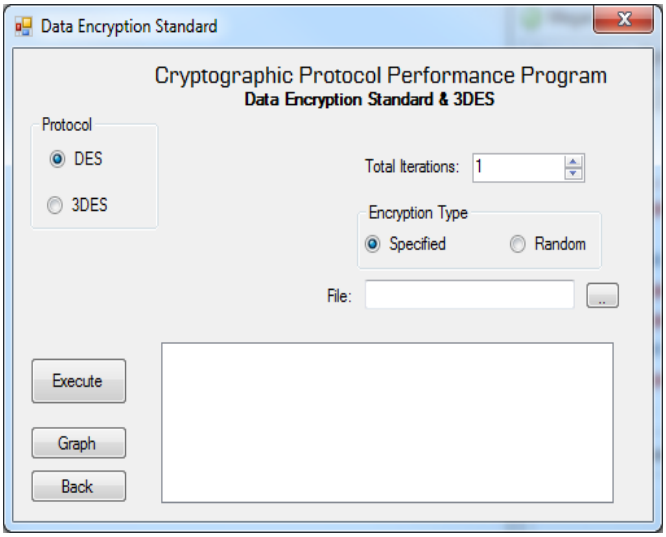
[8] *MSDN Library – SHA2 and related Message Digest Protocols*, 2010. Retrieved Dec 1, 2010, from http://msdn.microsoft.com/en-us/library/system.security.cryptography.sha256.aspx

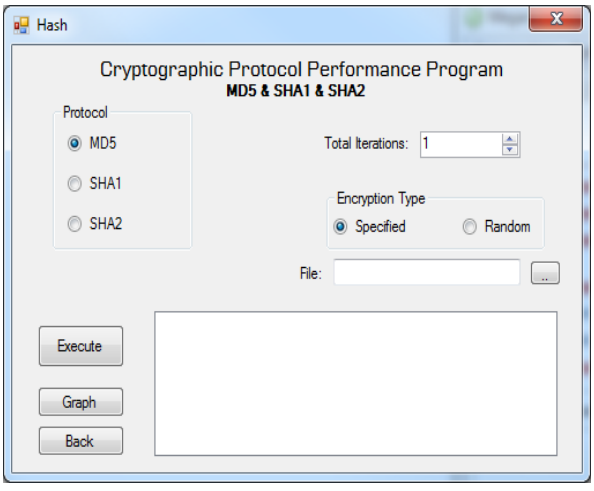
[9] *OpenSSL Command Line How-To*, 2010. Retrieved Dec 3, 2010, from http://www.madboa.com/geek/openssl/

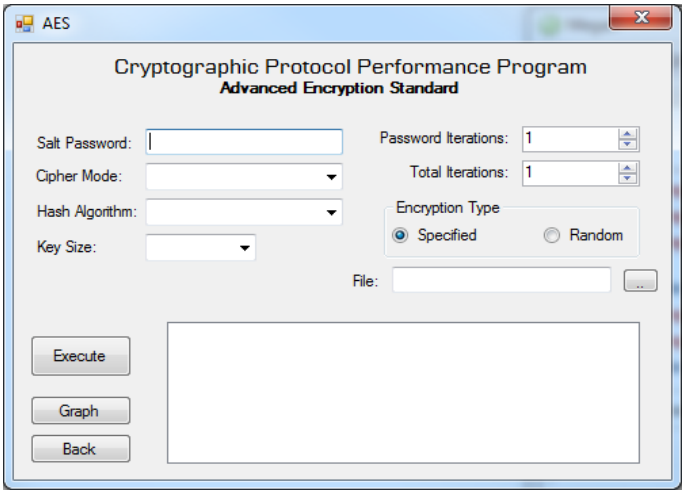
APPENDIX A. Snapshots of C3Pv2 Menus











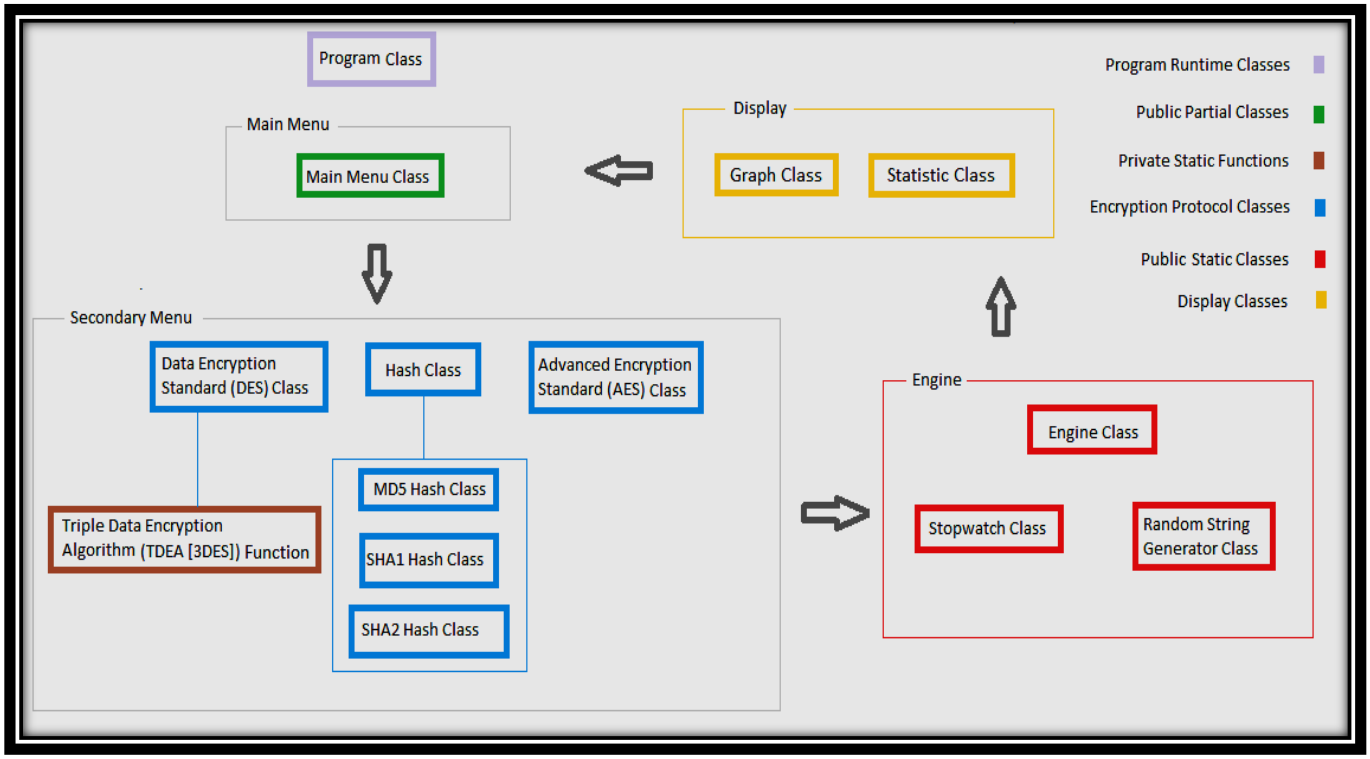


Figure 1. Architectural layout of the C3P

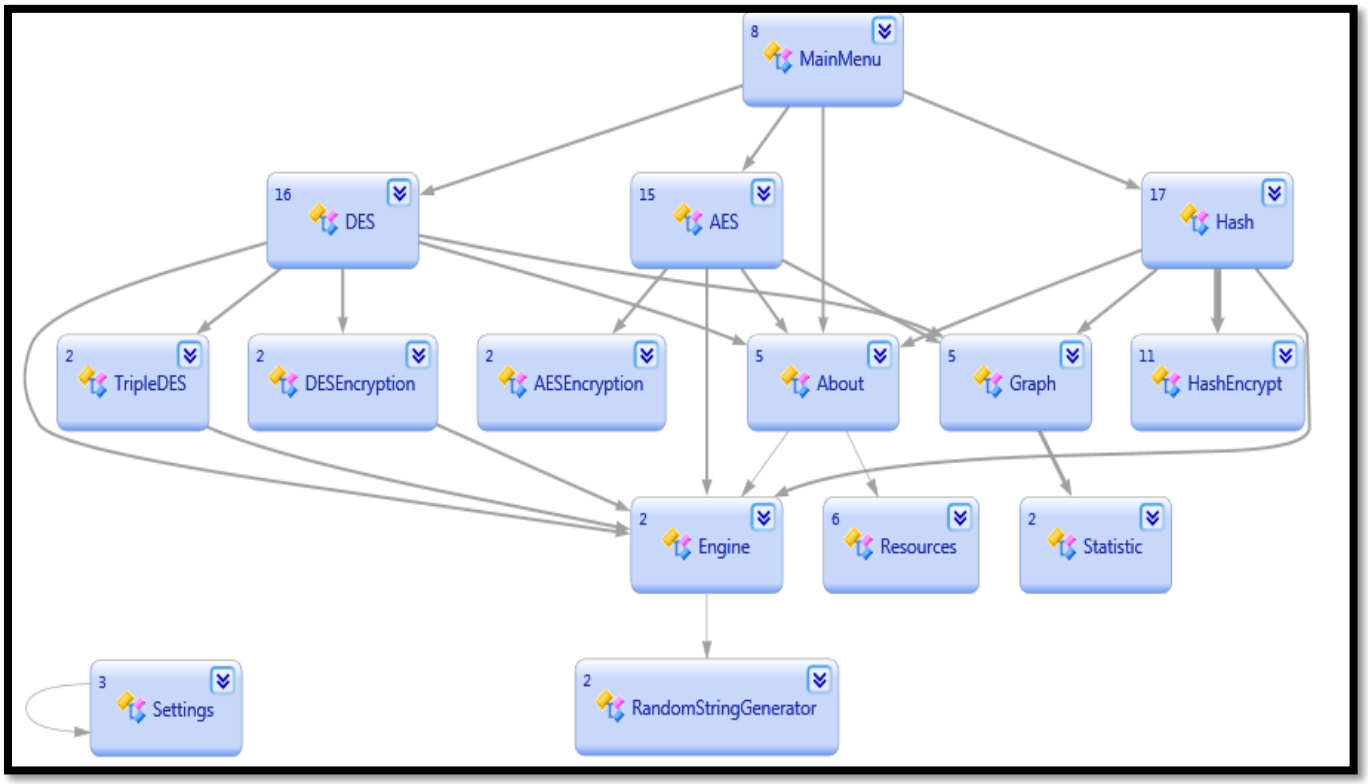


Figure 2. Schematic View of the Interacting Program Modules

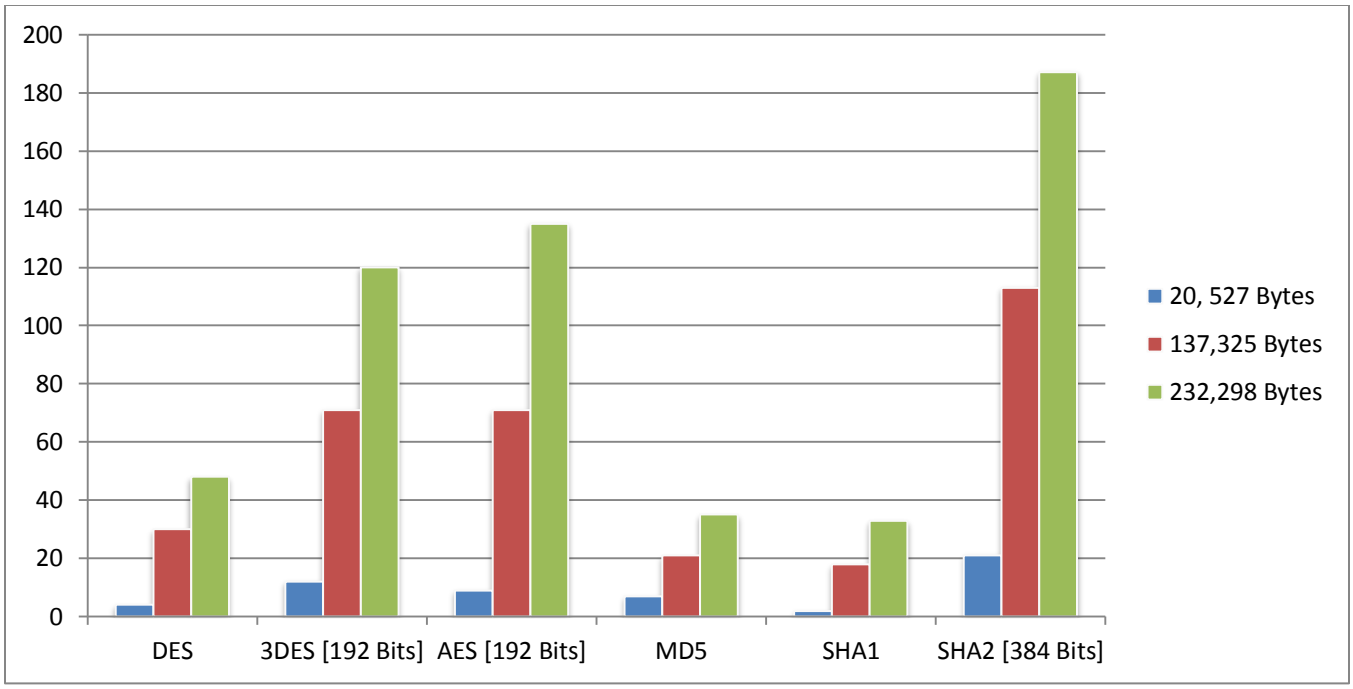


Figure 3. Time Performance of Different Encrypting Protocols for Various Random Data Sizes

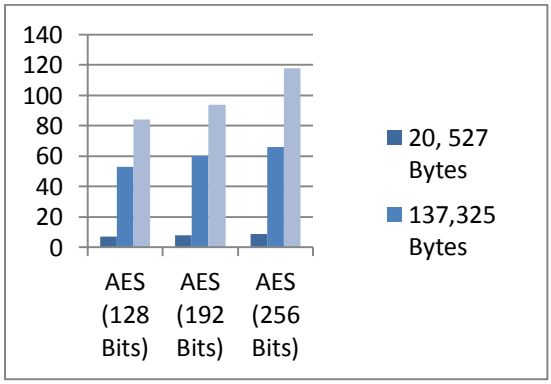


Figure 4. Execution Times for Various AES Key and Random Data Sizes

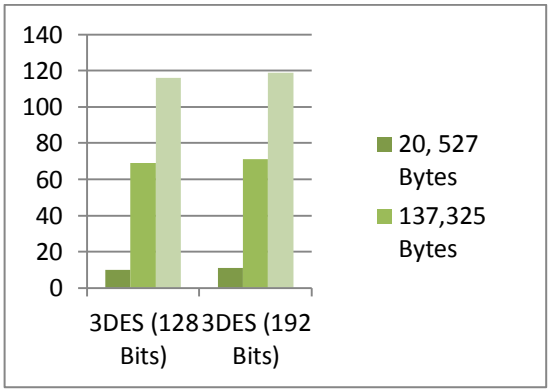


Figure 5. Execution Times for Various 3DES Key and Random Data Sizes

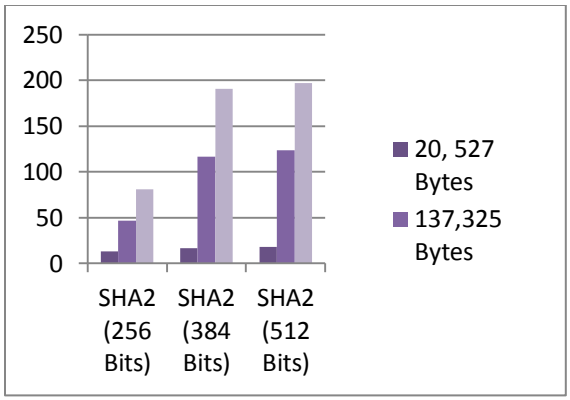


Figure 6. Execution Times for Various 3DES Key and Random Data Sizes

	DES	3DES128	3DES192	MD5
Intercept	0.8821	4.52459	5.70569	1.607
Slope	0.0002	0.00053	0.00054	0.0001
Correlation	0.9968	0.99446	0.99786	0.9983
R-Sq	0.9935	0.98895	0.99573	0.9965

Table 7. Regression Equations of DES and MD5

	AES128	AES192	AES256
Intercept	10.0601	11.791	19.4022
Slope	0.00028	0.0003	0.00036
Correlation	0.99142	0.9909	0.9843
R-Sq	0.98292	0.9819	0.96886

Table 8. Regression Equations of AES

	SHA1	SHA2-256	SHA2-384	SHA2-512
Intercept	0.5671	14.36323	9.413761	-27.077
Slope	0.0001	0.000249	0.000757	0.001094
Correlation	0.991	0.977094	0.998049	0.96722
R-Sq	0.9821	0.954712	0.996102	0.935515

Table 9. Regression Equations of SHA

BYTES	MD5	SHA2-384
50000	6	147
100000	14	175
200000	28	281
500000	72	579

Table 10. Execution Times for 1997 MHz CPU

BYTES	MD5	SHA2-384
50000	2	55
100000	5	72
200000	10	139
500000	29	359

Table 11. Execution Times for 2659 MHz CPU

BYTES	DES	3DES128	3DES192	MD5
20527	4	10	12	7
50000	17	29	33	10
100000	22	67	67	14
137325	30	69	71	21
150000	33	95	89	22
175000	39	109	107	27
200000	45	114	116	31
232298	48	116	120	35
250000	54	125	143	40
300000	71	153	169	47
350000	77	194	196	54
400000	93	219	222	61
450000	100	245	248	68
500000	109	268	273	75

Table 12. Execution Times of DES and MD5 on 2405 MHz CPU

BYTES	AES128	AES192	AES256
20527	7	8	9
50000	15	24	36
100000	41	36	56
137325	53	60	66
150000	58	65	71
175000	63	71	81
200000	67	78	95
232298	84	94	118
250000	89	102	124
300000	94	110	132
350000	106	123	151
400000	122	131	162
450000	137	154	174
500000	147	166	183

Table 13. Execution Times of AES on 2405 MHz CPU

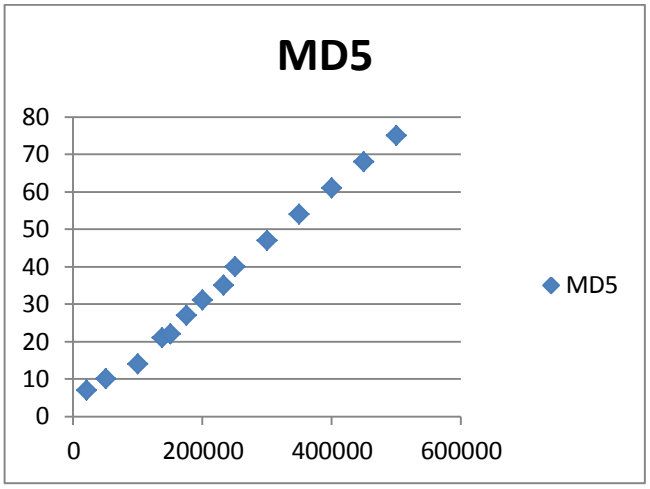


Figure 15. Graph of the Runtimes of MD5

BYTES	SHA1	SHA2-256	SHA2-384	SHA2-512
20527	2	13	17	18
50000	4	19	40	46
100000	11	28	85	102
137325	18	47	117	124
150000	21	52	128	143
175000	26	65	141	162
200000	29	72	161	182
232298	33	81	191	197
250000	36	89	210	231
300000	40	97	235	291
350000	43	105	284	318
400000	48	111	298	378
450000	56	118	349	413
500000	67	129	384	644

Table 14. Execution Times of SHA on 2405 MHz CPU

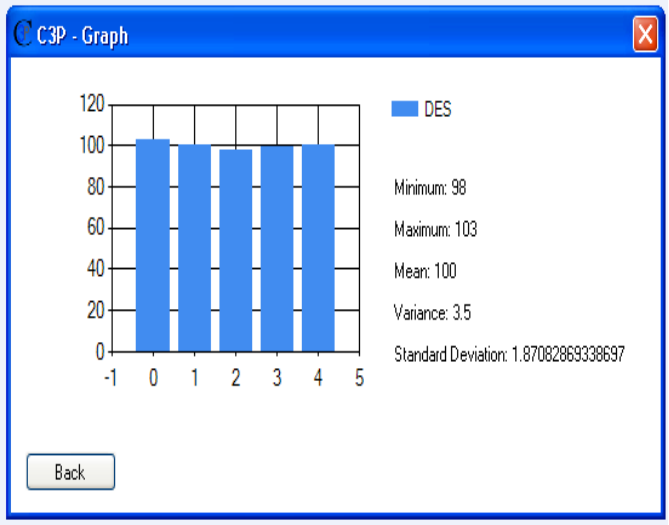


Figure 16. Sample Graph of Runtimes for DES