Empirical and Statistical Comparison of Intermediate Steps of AES-128 and RSA in Terms of Time Consumption

Prashant Pranav (prashantpranav19@gmail.com)
Birla Institute of Technology
Sandip Dutta
Birla Institute of Technology
Soubhik Chakraborty
Birla Institute of Technology

Research Article

Keywords: Cryptography, Algorithm Analysis, Empirical Complexity, Revised Difference, RSA, AES-128

Posted Date: May 3rd, 2021

DOI: https://doi.org/10.21203/rs.3.rs-189456/v1

Empirical and Statistical Comparison of Intermediate Steps of AES-128 and RSA in Terms of Time Consumption

[1*]Prashant Pranav, [2]Sandip Dutta, [3]Soubhik Chakraborty,

[1]Department of Computer Science and Engineering, Birla Institute of Technology, Mesra, Ranchi (835215), India
{prashantpranav19@gmail.com}
[2]Department of Computer Science and Engineering, Birla Institute of Technology, Mesra, Ranchi (835215), India
{sandipdutta@bitmesra.ac.in}
[3]Department of Mathematics, Birla Institute of Technology, Mesra, Ranchi (835215), India
{soubhikc@yahoo.co.in}
*corresponding author

Abstract: Cryptographic algorithms are composed of many complex mathematical functions. When analyzing the complexity of these algorithms, one fixes priory the overall complexity of the algorithm as the complexity of the most dominating operations for a group of operations. Generally, it is the *count* of this operation which determines the complexity of the algorithm in case of compounding operations. We have instead used the *weight* factor to determine the complexity of an algorithm with many operating functions working simultaneously and have taken time of the operation as a measure of the weight factor. We statistically analyze the two most used operations in RSA, namely "power" and "mod" through a method of revised difference to compare if these are statistically similar or dissimilar. We have also calculated the empirical computational complexity of these two operations through the fundamental theorem of finite differences to verify whether these operations are statistically dissimilar and if so then which of the two is dominant. We have also analyzed empirically the complexity of each of the four sub-steps involved in the encryption and decryption of AES-128, to determine which operation dominates the most and consumes most of the time in an overall run time of AES-128.

Keywords: Cryptography, Algorithm Analysis, Empirical Complexity, Revised Difference, RSA, AES-128

1. Introduction

Algorithm analysis (1, 2, 3 and 4) is an integral part of computer science wherein one tries to estimate the overall complexity in terms of both space and time before running the algorithm for any application. This helps to differentiate two algorithms with similar use when implemented in a particular application. Analysis of the running code by programmers has shown that often much of the time is spent in a small portion of the code and it is difficult to find the inefficient places in the code. The inefficient portion of the code is found by running the program for varying inputs and then replacing the inefficient portion with a better and more efficient code.

Cryptographic algorithms (5, 6, 7 and 8) are composed of many complex mathematical functions which are generally compounded, a group of singular operations. For example, both the encryption and decryption of two well known symmetric key algorithms RSA and El Gamal are composed of the *power* operation and *modular arithmetic* operation working as a single unit. Generally, while analyzing these two cryptographic algorithms, one tends to predict the overall complexity of the two algorithms tending towards the more dominant of the two operations “power” and “mod”. This may give a biased theoretical result and the algorithm may be unfit for many real time applications due to its excess execution time. We have evaluated the statistical similarity between the “power” and “mod” operations by analyzing them empirically. If the two operations are statistically similar, then none of them is dominant if analyzed in a group. But, if the two operations are statistically dissimilar then one out of the two may be dominant operating and will account for most of the execution time of the overall compounding operation. To find which one of the operations is more dominant, we have used the *fundamental theorem of finite differences*. When analyzing a compound operation theoretically, it is the *count* factor which determines the complexity of the algorithm. We have instead used a *weight* factor, namely, the time of the operation to measure the actual complexity. On the positive side, it allows mixing of operations which, in implementation, do work collectively and hence the approach is realistic. on the negative side, the time factor depends much on the computer clocks and the results can get affected if the clocks do not function properly.

To develop any working result, we first have to check the idiosyncrasies of computer clocks. Often these clocks are not much accurate and one cannot rely on the execution time achieved by running the program. The remedy is to increase the input until the total time required to process any operation is large enough to predict any result. It suffices to point out here that computer clocks (also called timers) have two components: the hardware part and the software part. All that the hardware part does, is generate rapid pulses at equal intervals (called ‘‘clock ticks’’); the rest is achieved by programming. A detailed description of the working of the computer clocks can be seen in (9).

The theoretical analyses of RSA for encryption and decryption processes give different complexity measures. The complexity of encrypting with the compounding operation $C = M^e \, mod \, n$ is $O\,(\log(N)^2)$ as it is done with a fixed or bounded number of modular multiplications. In cases, where e is of the same order as N, the complexity changes to $O\,(\log(N)^3)$. For decrypting with the compound operation $M = C^d \, mod \, n$, the time complexity is $O\,(\log(N)^3)$ as the factor d has the same size in bits or digits proportional to that of N which remains the same irrespective of using the CRT technique. Using “squaring and multiplying” algorithms for encryption and decryption, the time complexity can be seen to be $O(N)^2$.

Similarly, the block cipher algorithms such as AES-128, work normally on fixed block size. AES-128 works by encrypting a fixed block of length 128 bits and takes approximately the same time despite varying input. Thus AES-128 can be termed to be $O\ (1)$ for both encryption and decryption. If one wants to encrypt messages larger than 128 bits, then they need to be put into one of the modes of operations. In such a case, as we have O (n) blocks of data to encrypt/decrypt, this brings the order of complexity to $O\ (n)$, where n is the size of the message. So, the time complexity of AES-128 for both encryption and decryption in any of the standard modes of operations like the CBC or CTR is a polynomial, or to be more precise, linear with respect to the size of the message.

Theoretical computational complexity is based on count of the underlying functions comprising the algorithm. This may lead to erroneous result as sometimes the apparently more dominant of the operations may not be that much dominant. We have analyzed the compounding operation power and mod of RSA for their statistical relationship. A power function is a function where $y = x^n$ (n is any real constant number). The mod function finds a remainder after a number (dividend) is divided by another number (divisor).

We have also analyzed these two operations empirically wherein as opposed to theoretical complexity measure based on counts, the weights are taken into account as a measure for deriving the complexity. Time of the operation, as mentioned earlier, has been used as a weight parameter for doing the empirical analysis. Four sub steps of AES-128 are also analyzed empirically to find the most time consuming of the four sub-steps.

Section 2 of the paper covers similar research carried out in the field of statistical and empirical analysis and an introduction to the *fundamental theorem of finite difference* and *RSA* cryptographic algorithm. The Revised Difference Method is introduced in section 3, with a statistical comparison of power and mod operations used in encryption and decryption function of the RSA. Section 4 gives the Empirical time complexity of the power and the mod operations. Introduction to AES-128 and its intermediate steps is discussed in section 5, with an empirical analysis of all the intermediate steps to predict which of the sub-steps consumes the most time.

2. Related Work

The statistical similarity among three well known operations *addition*, *multiplication* and *equality* has previously been studies through the method of revised difference (see(10)). The result generated using the method of *Hypothesis Testing* approach shows that *addition, multiplication* and *equality* are statistically dissimilar when working in a group. The bubble sort program execution time with different parameters has been analyzed statistically (see (11)). A statistical analysis has been done to show the specific quadratic pattern of the execution time on the items sorted. The result states that the order of complexity of bubble sorting is $0\ (x^2)$ where x is the number of items sorted. The statistical analysis shows the specific quadratic pattern of any dependent variable y on the number of items sorted x. The empirical execution time is predicted using the *fundamental theorem of finite difference* (***See Comment below***). An empirical O is the empirical estimate of the non-trivial and conceptual weight based statistical bound. A statistical bound, unlike a mathematical bound, weighs the computing operation instead of counting them and it takes all the operations collectively and mixes them in assessing the bound (see (12)). The empirical analysis is rather a practical approach for analyzing an algorithm's complexity. First running a program constituting the algorithm and then fitting a statistical model gives a practical view of the algorithm's complexity in empirical form.

Comment: Fundamental theorem of finite difference states that the n^{th} difference of an n^{th} degree polynomial is constant and the higher differences are zero. Mathematically

$$\Delta^n P_n(x) = C = Constant$$

$$\Delta^{n+r} P_n(x) = 0, r = 1,2,3, \dots \dots$$

where Δ is the forwad difference operator defined as Δf(x) = f(x+h) - f(x) for values of f(x) tabulated at x = a, a+h, a+2h..... h being the interval of differencing.

The converse of the aforesaid theorem is also true. This means if the n^{th} difference of a tabulated function is constant and higher differences are zero, the function is an n^{th} degree the polynomial. No function has this unique property except a polynomial.

When a statistician tells that the empirical O of the algorithm in question is *O(g(n))* all he means is the following:

''The simplest model one can fit (to the algorithmic time complexity data) that prevents ill-conditioning by sacrificing a minimum amount of predictive power, if need be, must in my opinion have a leading functional term g(n) in the model.'' (13).

The authors in (14) presented a statistical view for investigating the average case complexity behavior of computer algorithms. A statistical bound and its empirical estimate, the so called ‘empirical-O’ is used. The result depicted few complexity data, ensuring a clear linear pattern recommending an empirical linear model [$Y_{avg} = O_{emp}$ (n)]. These resulted in conjectures in the realm of algorithmic analysis. Based on the statistical analysis, the robustness claim of average case behavior of the quick sort algorithm has been rejected (12). From the empirical side of the approach used in the paper with an urge in computer experiments and applied statistics offers a helping hand by supporting its theoretical equivalents. In (13) the authors propounded a study to demonstrate that an empirical O (n^2) complexity was persuasively gettable with two dense matrices in *nxn* matrix multiplication. The result showed that an empirical $O\ (n^2)$ complexity was undoubtedly gettable with both the matrices dense, i.e., the pre- factor matrix was roughly as dense as triangular and the post factor matrix was fully dense. The algorithm used in the study was Amir Schoor's algorithm (see (15)) which is fast for both sparse matrices as well as for dense matrices as it is faster to work with rows than with both rows and columns. Further study can be undertaken to observe whether the same can be done if the density of the pre- factor matrix is increased keeping the post matrix fully dense. For certain algorithms such as sorting, the parameters of the input distribution must also be taken into account, apart from the input size, for a more precise evaluation of computational and time complexity (average case only) of the algorithm in question, the so-called ‘‘gold standard’’ in the context of parameterized complexity (see (16)).

We have analyzed the RSA cryptographic algorithm and AES-128. The RSA algorithm is an asymmetric cryptography algorithm. The term asymmetric actually means that it works on two different keys, i.e. public key and private key**.** As the name indicates, the public key is given to everyone and the private key is kept private.

The encryption and decryption in the algorithm is carried out using the *power* and *mod* operation working in a group. The encryption procedure follows the following equation:

$$C = M^e \bmod n,$$

where, C is the generated *Cipher Text,* M is the *Plain Text,* N is the *Modulus* and e is the *public exponent.* Similarly, the decryption routine follows the following equation:

$$M = C^d \bmod n,$$

where, d is the *private exponent.*

3. Revised Difference Method

Let “*p*” and “*m*” be referring to the power and mod operations respectively. Since computer clocks are not reliable for small input, the input size keeps on increasing. This helps to draw a relation between the size of the input and the corresponding execution time. We have taken 3 trials for each input size in order to defy the dominance of cache hit which is the case of finding an input in the cache memory itself instead of the main memory. This can significantly affect the execution time of a specific input as finding the input in cache instead of the main memory takes considerably less amount of time. Let $\left(t_{p_i} and\ t_{m_i}\right)$ = the mean run time of the three trials for the “power” and “mod” operations for *i = 1, 2, 3,….,* are corresponding to r instances of input size respectively. The revised difference is defined as:

$$d_i = \frac{\left(t_{p_i} - t_{m_i}\right)}{\left(t_{p_i} + t_{m_i}\right)/2},\ i = 1,2,3,\ldots.,r \qquad (1)$$

The revised difference is unit free, statistically random with a small variance (more precisely stated, the small variance means there is no sign of such variance with respect to the standard deviation). The magnitude of the difference between two times should be reckoned relative to the magnitude of the terms being differenced. Further, through trial and error, we discovered that the function given in equation (1) generally has a small variance, when c and d are the simplest operations considered. However, this may not be so always (for example, in a program with a partial differential equation) for which the formula will have to be further improved. The unit freeness follows from the definition itself. For testing statistical randomness, we recommend the well-known run test for randomness. Although the formula for our t (see equation 2 below) is same as that of paired t, there is a difference. Paired t-test is both a test for difference of means and a test of mean difference since testing E(X) = E(Y) is equivalent to testing E(X-Y) = 0. But here we shall be making a test whether the mean of the population of revised differences is zero. Thus, it is a test for mean revised difference and not a test of mean difference and therefore not a test for difference of means. More precisely E[(X - Y)/{(X + Y)/2}] = 0 does not imply E(X) = E(Y).(see (10))

To check whether the mean of the revised differences is zero or not we have applied the hypothesis testing approach which is best suited to estimate the population behavior from a small sample of the population. We set the null and alternative hypothesis as:

H_0 = The mean of the population of revised difference is zero

H_1 = The mean of the population of revised difference is not zero.

The term "population" in general refers to a set of similar items or events which is of interest for some question or experiment. A common aim of statistical analysis is to produce information about some chosen population. In this context, the term "population" signifies a large set of both the operations in terms of execution time. In general a small sample is collected from the whole population by some sampling techniques to estimate the overall behavior of the population. This is done through the "hypothesis testing" approach and we have taken a "sample" of 10 and 30 of both the operations to estimate the population behavior. The null hypothesis is the hypothesis under test and it must be a hypothesis of zero (null) difference which depicts the unbiased attitude of the statistician. The alternative hypothesis is that against which the null hypothesis is to be tested.

We compute the test statistics "t" as:

$$t = \frac{d_m}{ds/\sqrt{r}} \tag{2}$$

where,

r is the total number of observations for each operation, d_m is the mean of the revised difference (of r observations) and d_s is its standard deviation

$$d_m = \frac{d_1+d_2+\cdots+d_r}{r} \text{ and} \tag{3}$$

$$d_s = [\frac{1}{(r-1)}\sum_{i=1}^{r}(d_i - d_m)^2]^{1/2} \tag{4}$$

This may assume to follow a Student's t-distribution which is any member of a family of continuous probability distributions that arises when estimating the mean of a normally distributed population in situations where the sample size is small and the population standard deviation is unknown with r-1 degree of freedom. The degree of freedom represents the minimum number of values of the sample which can be used to precisely estimate the whole population. Most of the times, a sample of size N has N-1 degrees of freedom. We reject the null hypothesis if the magnitude of the calculated test statistic exceeds the table value at the 95% level of confidence (and hence level of significance α = 0.05) and r-1 degree of freedom. The level of significance, also denoted as alpha or α, is the probability of rejecting the null hypothesis when it is true. For example, a significance level of 0.05 indicates a 5% risk of concluding that a difference exists when there is no actual difference. In case of acceptance of the null hypothesis, we say that the computing operations are statistically similar and in case of rejection, the computing operations are termed as statistically dissimilar.

The Student's t-distribution is valid only for sample size less than 30. For sample size greater than 30, instead of Student's t-test, Normal table should be used with the same formula (as used for calculating the t-statistic). In this case, if the calculated statistic exceeds 1.96 in magnitude, we reject the null hypothesis; else may accept it at a 95% level of confidence.

Table 1 and Table 2 show the execution time for three different trials for both the operations say x^y and $(x \; mod \; y)$.

System Specification:

Processor: Intel (R) Core ™ i3

RAM: 4.00 GB

Operating System: 64 bit Windows 7

All the implementations in this work were done in MATLAB version 2015a

Table 1: Execution Time for "power" Operation (small data set)

Input (in bits)	Trial 1	Trial 2	Trial 3	Mean Execution Time in sec for x^y
500	0.000713	0.000646	0.000683	0.000681
1000	0.001116	0.000971	0.001037	0.001041
1500	0.001246	0.001341	0.001242	0.001276
2000	0.001612	0.001552	0.001585	0.001583
2500	0.001896	0.001813	0.002023	0.001911
3000	0.002582	0.002604	0.002422	0.002536
3500	0.002949	0.002808	0.002964	0.002907
4000	0.003091	0.002711	0.003146	0.002983
4500	0.003137	0.003887	0.002855	0.003293
5000	0.003771	0.002961	0.003371	0.003368

Table 2: Execution Time for "mod" Operation (small data set)

Input (in bits)	Trial 1	Trial 2	Trial 3	Mean Execution Time in sec for $x \; mod \; y$
500	0.000529	0.000505	0.000393	0.000476
1000	0.000576	0.000433	0.000418	0.000476
1500	0.000686	0.00051	0.000452	0.000549
2000	0.000604	0.000522	0.000514	0.000547
2500	0.00069	0.000659	0.000773	0.000707

3000	0.000538	0.000585	0.00052	0.000548
3500	0.000799	0.00054	0.000721	0.000687
4000	0.000798	0.000595	0.000691	0.000695
4500	0.000895	0.000568	0.000853	0.000772
5000	0.000711	0.000825	0.00066	0.000732

Table 3: Revised Difference Between "power" and "mod" Operations

Input (in bits)	Mean (Execution Time) for x^y	Mean (Execution Time) for $x\ mod\ y$	Revised Difference
500	0.000681	0.000476	0.354569
1000	0.001041	0.000476	0.74577
1500	0.001276	0.000549	0.796421
2000	0.001583	0.000547	0.973235
2500	0.001911	0.000707	0.919277
3000	0.002536	0.000548	1.289613
3500	0.002907	0.000687	1.235692
4000	0.002983	0.000695	1.24438
4500	0.003293	0.000772	1.240344
5000	0.003368	0.000732	1.285796

From Table 3 of the revised difference we get

d_m = 1.00851

d_s = 0.310512

As the number of observations 'r' is 10, the degree of freedom = 9. So, calculated t-statistic at the 95 % level of confidence and hence level of significance $\alpha = 0.05$ is 10.27074 and tabulated $t_{(0.05,9)}$ = 2.262. Since the calculated t-statistics is greater than the tabulated one, so we reject the null hypothesis and conclude at this stage that power and mod operations involved in the encryption and decryption processes of RSA are statistically dissimilar. Now, we check the empirical run time of these two operations through the *fundamental theorem of finite difference.*

Remark: The term statistic refers to a function of sample observations (e.g. sample mean, sample range), as opposed to the term parameter meaning a function of population observations. The term statistics when used as a plural of the singular statistic will mean more than one such sample functions. The term test statistic means a statistic that is used to carry out a test of significance.

4. Empirical Complexity Estimate of "Power" and "Mod Operations"

We applied empirical analysis for measuring the complexity of power and mod operations. We took 'x' to be the size of the input used in each of the two operations and 'y' the execution time in seconds in order to determine a function 'f' of the form $y = f(x)$. The estimated run time empirically can be found by using the converse of the fundamental theorem of finite difference discussed in section 2.

For the difference Table 3 of power operation, we found that the first difference is almost constant and the second difference is almost zero. This can be proved by emphasizing that as the input keeps varying, the 2nd difference tends to give values that are negligible and can eventually be excluded from the final calculation. Hence, by the converse of the fundamental theorem of finite difference, a 1st-degree polynomial will approximate the execution time for both encryption and decryption. So the empirical complexity of the power operation can be termed to be empirical O *(n)*, where 'n' is the length of the message. This can be proved by fitting a statistical model corresponding to the mean execution time of power operation.

Table 4: First and Second Differences for "power" Operation

Input (in bits)	Mean Execution Time in sec for x^y	First Difference	Second Difference
500	0.00068	0.00036	-0.00012
1000	0.00104	0.00024	0.00006
1500	0.00128	0.0003	0.00003
2000	0.00158	0.00033	-0.00002
2500	0.00191	0.00031	0
3000	0.00222	0.00031	0
3500	0.00253	0.00031	0.00001
4000	0.00284	0.00032	0.00001
4500	0.00316	0.00033	
5000	0.00349		

The goodness of fit of the fitted model can be tested using a residual plot method. If an almost horizontal pattern is observed, then, we can say that the model predicted is the actual empirical complexity of the run time. The fitted line plot and the residual plot for the encryption process are shown below in figure 1 and figure 2 respectively. These were plotted using the statistical package MINITAB version 16.

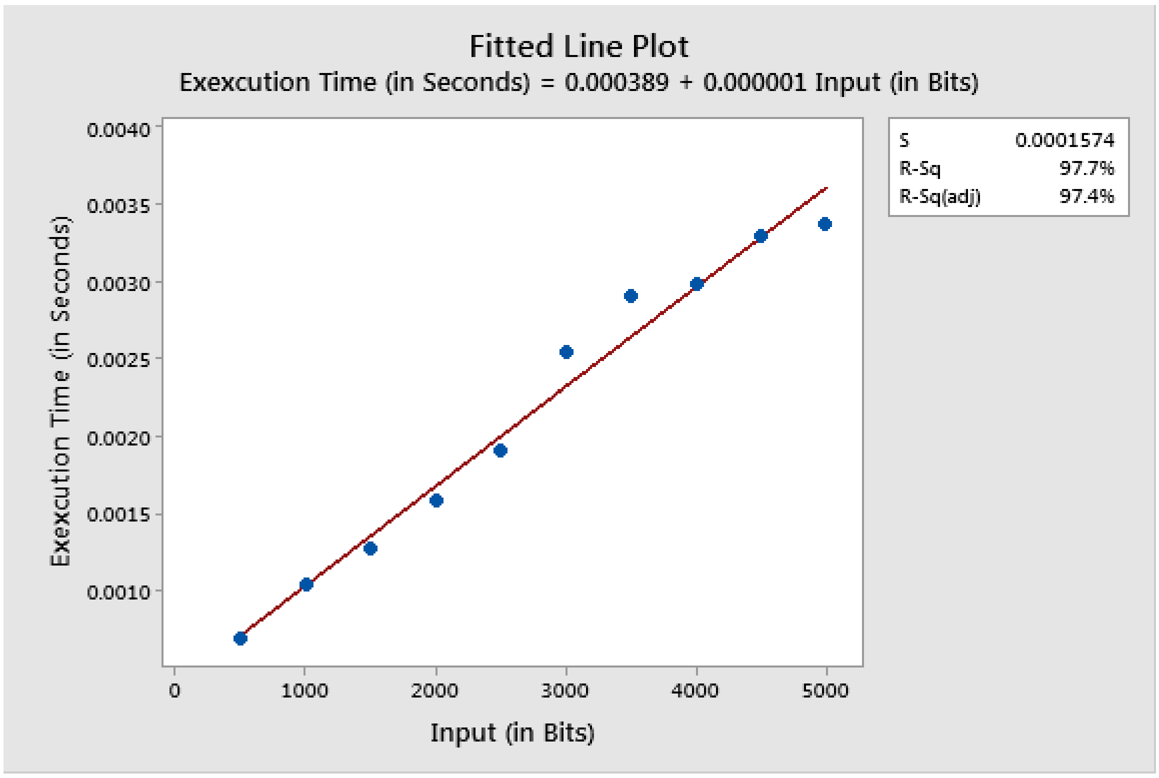

Fig 1. Statistical Model Fitting of Power Operation

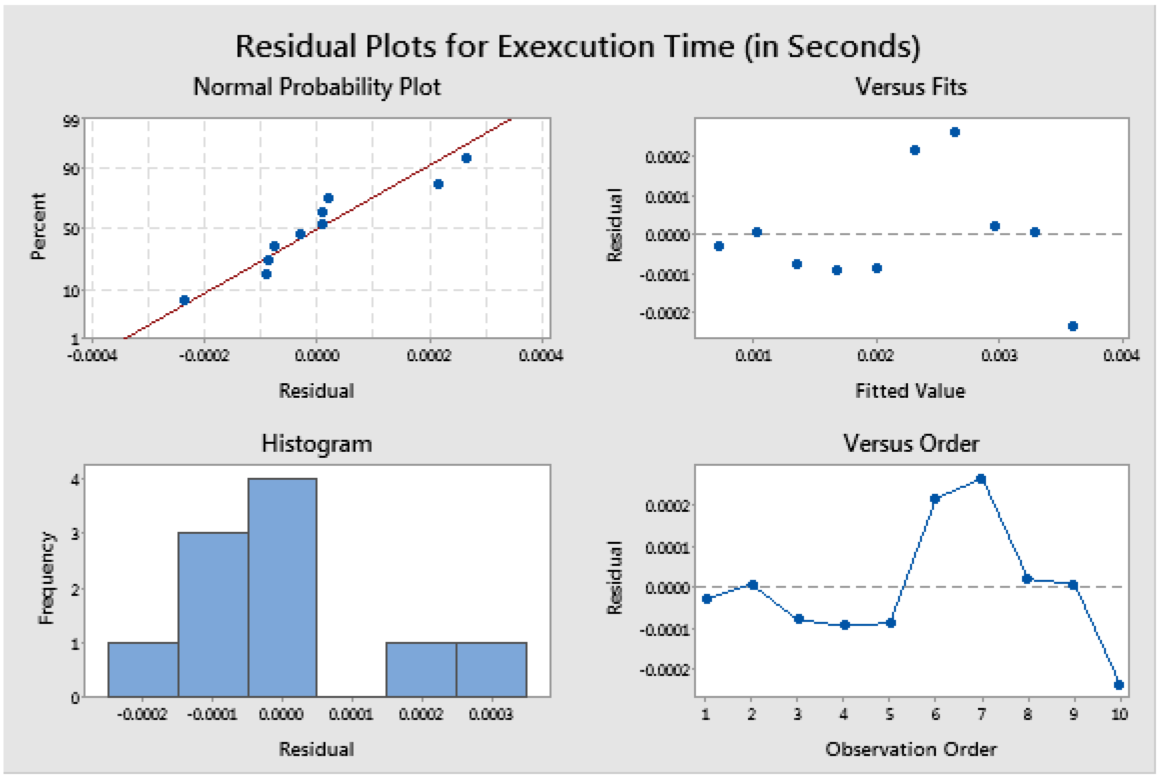

Fig 2. Residual Plot of Power Operation

The fitted line plot in figure 1 above shows a linear fitting of the power operation and so it can be termed to be *O (n)*. The residual plot above has 4 sub-plots. The *Normal Probability Plot* depicts an almost linear pattern and thus shows that the data are distributed normally. The *Versus Fit Plot* shows that there is no outlier data present in the execution time.

Similarly, the *Histogram* shows whether the data are skewed or not. The plotted histogram does not show any skew or outlier. The *Residual Versus Order* shows the independence of residuals from one another. In normal scenario, if the residuals are distributed randomly around the normal line, then the residuals are almost independent of each other which is what we have obtained in our case.

Similarly, from the difference table 5 of *mod* operation, we see that the first difference tends to zero. As can be seen from the table 5 below that first difference does not show any reasonable pattern in the values and some of the values in the table are negative, which cannot be the case for the execution time of any operation and so no polynomial can estimate the empirical run time of *mod* operation and hence we term it to be empirical *O (1)*. This can be proved from the following discussion.

Table 5: First Differences for "mod" Operation

Input (in bits)	Mean Execution Time in sec for $x \bmod y$	First Difference
500	0.000476	0
1000	0.000476	0.000073
1500	0.000549	-0.000002
2000	0.000547	0.00016
2500	0.000707	-0.000159
3000	0.000548	0.000139
3500	0.000687	0.000008
4000	0.000695	0.000077
4500	0.000772	-0.00004
5000	0.000732	

The estimated empirical complexity is evidently *O (1)*, through the scatter plot as it correctly represents how one variable (execution time) is affected by other (number of inputs) and can be used to understand the relationship between two variables. The execution time for the *mod* operation is the mean of the 3 trials. If we plot the mean execution time in a scatter plot and the plot does not show any pattern, then we can confirm that no well behaved function such as a polynomial can approximate the empirical run time of the algorithm. So we plotted the mean of the execution time and what we observed is shown below in figure 3:

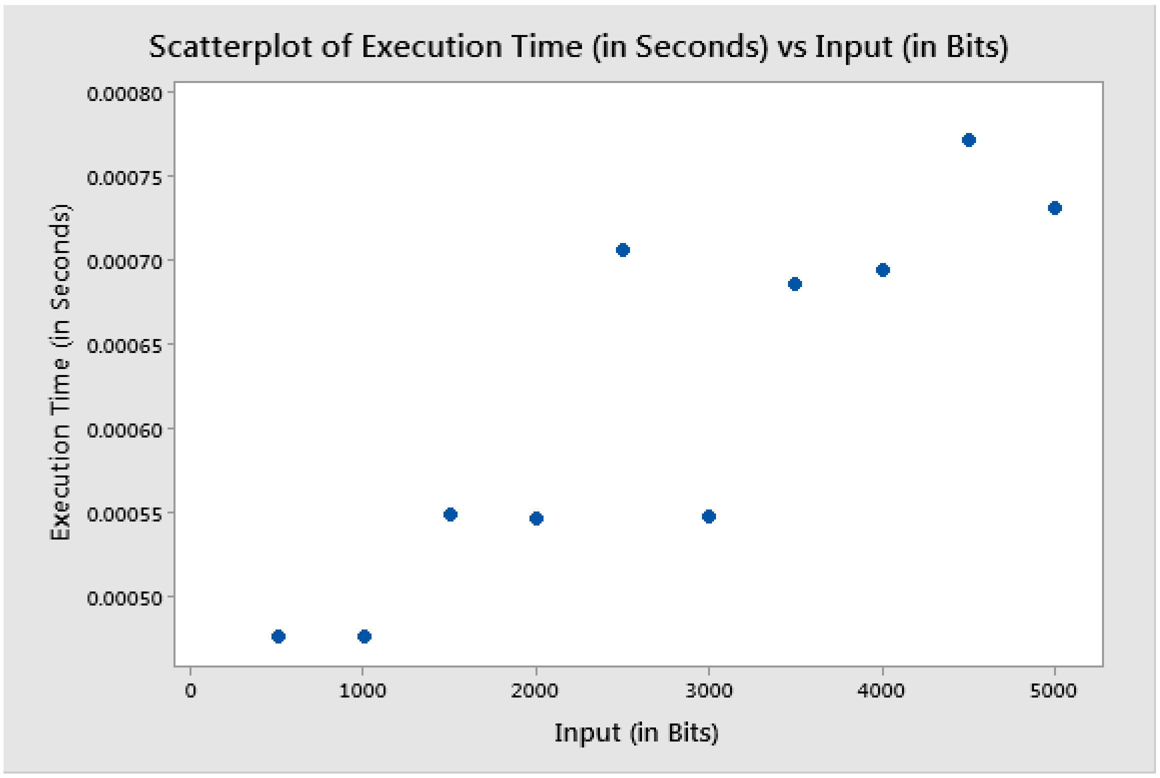

Fig 3. Scatter Plot of First Difference of “mod” Operation

The scatter plot of the *mod* operation shows a random distribution of the execution time and thus the *mod* operation can be termed to be empirical *O (1)*.

We conclude that the *power* and *mod* operations are statistically dissimilar and the *power* operation with an empirical complexity of *O (n)* is more dominant than the *mod* operation which has an empirical complexity of *O (1)*.

To further focus on the results we tried both the methods with an input greater than 30. The results of execution time for three different trials with an input greater than 30 for both the operations are shown in table 6 and table 7 and the revised differences are shown in table 8.

Table 6: Execution Time for “power” Operation (large data set)

Input (in bits)	Trial 1	Trial 2	Trial 3	Mean Execution Time in sec for x^y
100	0.000208	0.000144	0.000137	0.000163
200	0.000221	0.000193	0.000176	0.000197
300	0.000302	0.000233	0.00024	0.000258
400	0.000331	0.000321	0.000318	0.000323
500	0.000408	0.000376	0.000361	0.000382
600	0.00044	0.000432	0.00042	0.000431
700	0.000576	0.000511	0.000479	0.000522
800	0.000558	0.000586	0.000546	0.000563
900	0.000685	0.000615	0.000655	0.000652
1000	0.000737	0.000665	0.000692	0.000698
1100	0.000805	0.000712	0.000726	0.000748
1200	0.000865	0.000784	0.000788	0.000812
1300	0.000946	0.00088	0.000822	0.000883

1400	0.000991	0.000897	0.000947	0.000945
1500	0.00105	0.000964	0.001009	0.001008
1600	0.00107	0.001021	0.001006	0.001032
1700	0.001166	0.001077	0.001088	0.00111
1800	0.001146	0.001124	0.001131	0.001134
1900	0.001333	0.001191	0.001259	0.001261
2000	0.001273	0.001253	0.001267	0.001264
2100	0.001431	0.001307	0.00129	0.001343
2200	0.001538	0.001354	0.001366	0.001419
2300	0.001593	0.001415	0.00141	0.001473
2400	0.001611	0.001483	0.001461	0.001518
2500	0.00177	0.001528	0.001603	0.001634
2600	0.001942	0.001859	0.00182	0.001874
2700	0.002172	0.002074	0.002052	0.002099
2800	0.002211	0.002145	0.002256	0.002204
2900	0.002348	0.002279	0.002323	0.002317
3000	0.002402	0.002349	0.002418	0.00239
3100	0.002561	0.002537	0.002489	0.002529
3200	0.002612	0.002599	0.002605	0.002605
3300	0.002746	0.002783	0.002694	0.002741
3400	0.002802	0.002814	0.002782	0.002799
3500	0.002956	0.002902	0.002987	0.002948

Table 7: Execution Time for “mod” Operation (large data set)

Input (in bits)	Trial 1	Trial 2	Trial 3	Mean Execution Time in sec for $x \bmod y$
100	0.000045	0.00004	0.000059	0.000048
200	0.000071	0.000037	0.000036	0.000048
300	0.000066	0.000039	0.000059	0.000164
400	0.000063	0.000049	0.00005	0.000054
500	0.000075	0.000066	0.000053	0.000194
600	0.000074	0.000054	0.000064	0.000064
700	0.000118	0.000076	0.000079	0.000091
800	0.000077	0.000102	0.000068	0.000247
900	0.000106	0.000081	0.000103	0.00029
1000	0.000112	0.00007	0.000068	0.00025
1100	0.00013	0.000083	0.000075	0.000096
1200	0.00013	0.00012	0.000074	0.000108
1300	0.000129	0.000099	0.000083	0.000104
1400	0.000155	0.000081	0.000134	0.000123
1500	0.000141	0.000098	0.000113	0.000117
1600	0.000111	0.000091	0.000097	0.000299
1700	0.000155	0.000087	0.000094	0.000112
1800	0.000107	0.000091	0.000094	0.000292
1900	0.000182	0.000098	0.000133	0.000138
2000	0.000107	0.000102	0.000118	0.000109
2100	0.000155	0.000107	0.000104	0.000122
2200	0.000199	0.000102	0.000104	0.000135
2300	0.000191	0.000118	0.000106	0.000138
2400	0.000185	0.000109	0.000108	0.000134

2500	0.00023	0.000114	0.000114	0.000153
2600	0.000134	0.000156	0.000178	0.000156
2700	0.000144	0.000159	0.000182	0.000162
2800	0.000219	0.000182	0.000142	0.000181
2900	0.000186	0.000172	0.000162	0.000173
3000	0.000195	0.000187	0.000191	0.000191
3100	0.000193	0.000189	0.000187	0.00019
3200	0.000194	0.000198	0.000184	0.000192
3300	0.000196	0.000194	0.000182	0.000191
3400	0.000185	0.000184	0.000193	0.000187
3500	0.000198	0.000189	0.000199	0.000195

Table 8: Revised Difference Between "power" and "mod" Operations

Input (in bits)	Mean Execution Time in sec for x^y	Mean Execution Time in sec for $x \bmod y$	Revised Difference
100	0.000163	0.000048	1.090047
200	0.000197	0.000048	1.215259
300	0.000258	0.000164	0.445146
400	0.000323	0.000054	1.427562
500	0.000382	0.000194	0.651998
600	0.000431	0.000064	1.48248
700	0.000522	0.000091	1.406199
800	0.000563	0.000247	0.780749
900	0.000652	0.00029	0.768142
1000	0.000698	0.00025	0.945148
1100	0.000748	0.000096	1.544844
1200	0.000812	0.000108	1.530605
1300	0.000883	0.000104	1.579588
1400	0.000945	0.000123	1.538222
1500	0.001008	0.000117	1.582815
1600	0.001032	0.000299	1.101652
1700	0.00111	0.000112	1.633488
1800	0.001134	0.000292	1.180734
1900	0.001261	0.000138	1.606292
2000	0.001264	0.000109	1.682524
2100	0.001343	0.000122	1.666818
2200	0.001419	0.000135	1.652584
2300	0.001473	0.000138	1.656528
2400	0.001518	0.000134	1.67561
2500	0.001634	0.000153	1.658145
2600	0.001874	0.000156	1.69256
2700	0.002099	0.000162	1.713991
2800	0.002204	0.000181	1.696436
2900	0.002317	0.000173	1.721553
3000	0.00239	0.000191	1.703952
3100	0.002529	0.00019	1.720942
3200	0.002605	0.000192	1.725453
3300	0.002741	0.000191	1.739852
3400	0.002799	0.000187	1.749107
3500	0.002948	0.000195	1.751458

From Table 3 of the revised difference we get

d_m = 1.449099

d_s = 0.356954

And t = $\frac{d_m}{ds/\sqrt{r}}$ = 24.01709

Since t = 24.01709 which is greater than 1.96 at 95% level of confidence, so we reject the null hypothesis and conclude that power and mod operations are statistically dissimilar if the data are normally distributed. We leave the method of finite difference with a comment that it also shows the same result i.e. power operation is empirical O (n) and mod operation is empirical O (1) for inputs greater than 30 and so both are statistically dissimilar.

From, the discussion above we can say that, in encryption and decryption processes of RSA, the two used operations viz. power and mod are not similar when analyzed statistically. Rather, *power* is a dominant operation with the empirical complexity of *O (n)* as compared to the *mod* operation which has an empirical complexity of *O (1)*.

5. Analysis of Intermediate Steps of AES-128

The Advanced Encryption Standard (AES), also known by its original name Rijndael is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001. AES is a subset of the Rijndael block cipher developed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen, who submitted a proposal to NIST during the AES selection process. For a proper introduction of AES, *see* (17, 18, 19 and 20)

AES works by encrypting a block of size 128 bits and has a key length of 128 bits, 192 bits, and 256 bits so giving the AES standards the name as AES-128, AES-192 and AES-256 with each standards using a round of 10, 12 and 14 respectively.

We analyzed each of the 4 sub-stages of AES-128. The values for the run of three trials for the four intermediate steps are shown in table 9, 10, 11 and 12

Table 9: Execution Time for Substitute Bytes

Input (In Bytes)	Trial 1	Trial 2	Trial 3	Mean Execution Time in sec
16	0.000011	0.000009	0.000010	0.00001
32	0.000013	0.000011	0.000006	0.00001
48	0.000012	0.000012	0.000010	0.0000113
64	0.000010	0.000013	0.000008	0.0000103
80	0.000014	0.000015	0.000011	0.0000103
96	0.000017	0.000016	0.000009	0.000014
112	0.000012	0.000010	0.000013	0.0000116
128	0.000017	0.000013	0.000007	0.0000123
144	0.000019	0.000015	0.000012	0.0000153
160	0.000015	0.000019	0.000006	0.0000103
176	0.000020	0.000018	0.000011	0.0000163
192	0.000016	0.000017	0.000009	0.000014

208	0.000012	0.000015	0.000011	0.0000126
224	0.000013	0.000018	0.000013	0.0000146
240	0.000014	0.000020	0.000019	0.0000176
256	0.000021	0.000017	0.000020	0.0000193

Table 10: Execution Time for Shift Rows

Input (In Bytes)	Trial 1	Trial 2	Trial 3	Mean Execution Time in sec
16	0.000023	0.000019	0.000017	0.0000196
32	0.000019	0.000014	0.000010	0.0000143
48	0.000025	0.000017	0.000020	0.0000206
64	0.000021	0.000015	0.000011	0.0000156
80	0.000022	0.000013	0.000011	0.0000153
96	0.000019	0.000018	0.000014	0.000017
112	0.000027	0.000020	0.000025	0.000024
128	0.000021	0.000026	0.000017	0.0000213
144	0.000029	0.000017	0.000022	0.0000226
160	0.000024	0.000018	0.000019	0.0000203
176	0.000020	0.000021	0.000017	0.0000173
192	0.000022	0.000028	0.000019	0.000023
208	0.000030	0.000025	0.000029	0.000026
224	0.000022	0.000017	0.000019	0.0000193
240	0.000027	0.000020	0.000016	0.000021
256	0.000026	0.000022	0.000023	0.0000236

Table 11: Execution Time for Add Round Key

Input (In Bytes)	Trial 1	Trial 2	Trial 3	Mean Execution Time in sec
16	0.0000011	0.000009	0.000002	0.00000403
32	0.0000017	0.000008	0.000005	0.0000049
48	0.0000013	0.0000014	0.0000010	0.0000012
64	0.0000019	0.0000012	0.0000007	0.0000012
80	0.0000014	0.0000017	0.000008	0.0000033
96	0.0000013	0.0000016	0.000009	0.0000039
112	0.0000016	0.0000015	0.0000019	0.00000106
128	0.0000017	0.0000018	0.000007	0.0000035
144	0.0000020	0.0000014	0.0000011	0.0000015
160	0.0000016	0.0000017	0.0000010	0.0000014
176	0.0000021	0.0000016	0.0000013	0.00000106
192	0.0000017	0.0000019	0.0000012	0.0000016
208	0.0000018	0.0000016	0.0000017	0.0000017
224	0.0000022	0.0000019	0.0000016	0.0000019
240	0.0000020	0.0000023	0.0000019	0.00000206
256	0.0000023	0.0000021	0.0000019	0.0000021

Table 12: Execution Time for Mix Column

Input (In Bytes)	Trial 1	Trial 2	Trial 3	Mean Execution Time in sec
16	0.0229	0.0258	0.0219	0.023533
32	0.0531	0.0542	0.0553	0.0542
48	0.0862	0.0863	0.0851	0.085867
64	0.1141	0.1139	0.1161	0.1147
80	0.1348	0.1472	0.1451	0.142367
96	0.1757	0.1585	0.1783	0.170833
112	0.2057	0.2084	0.1956	0.203233
128	0.2354	0.2286	0.233	0.232333
144	0.293	0.2495	0.2534	0.2653
160	0.294	0.3014	0.3076	0.301
176	0.3233	0.3061	0.3239	0.317767
192	0.358	0.3331	0.3554	0.348833
208	0.395	0.4789	0.468	0.4473
224	0.5531	0.4092	0.4936	0.4853
240	0.4464	0.5187	0.6189	0.528
256	0.5339	0.5921	0.5352	0.553733

We applied the *fundamental theorem of finite difference* approach discussed in section 2 above to evaluate the empirical computational complexity of the four sub-steps of AES-128. This is shown in table 13. We observed that the first difference, i.e. Δy is itself almost zero for the three processes viz. "Substitute bytes", "Shift Rows" and "Add Round Key". So, no polynomial can approximate the empirical run time of these three steps and rather the mean execution times for these three steps are random which has been proved using the scatter plot approach and shown in figure 4, 5 and 6 respectively. So, we can say that these three steps are empirical *O (1)* as the scatter plots for all these sub-steps show an almost random pattern.

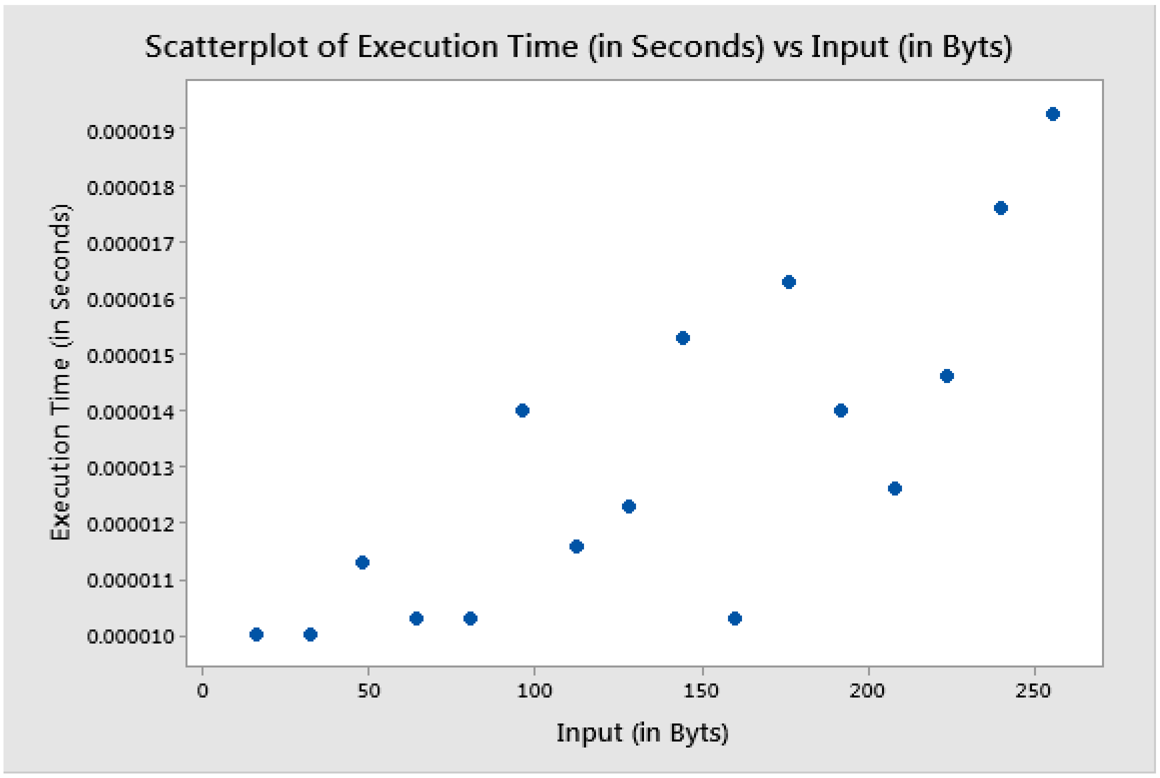

Fig 4. Scatter Plot of Substitute Byte

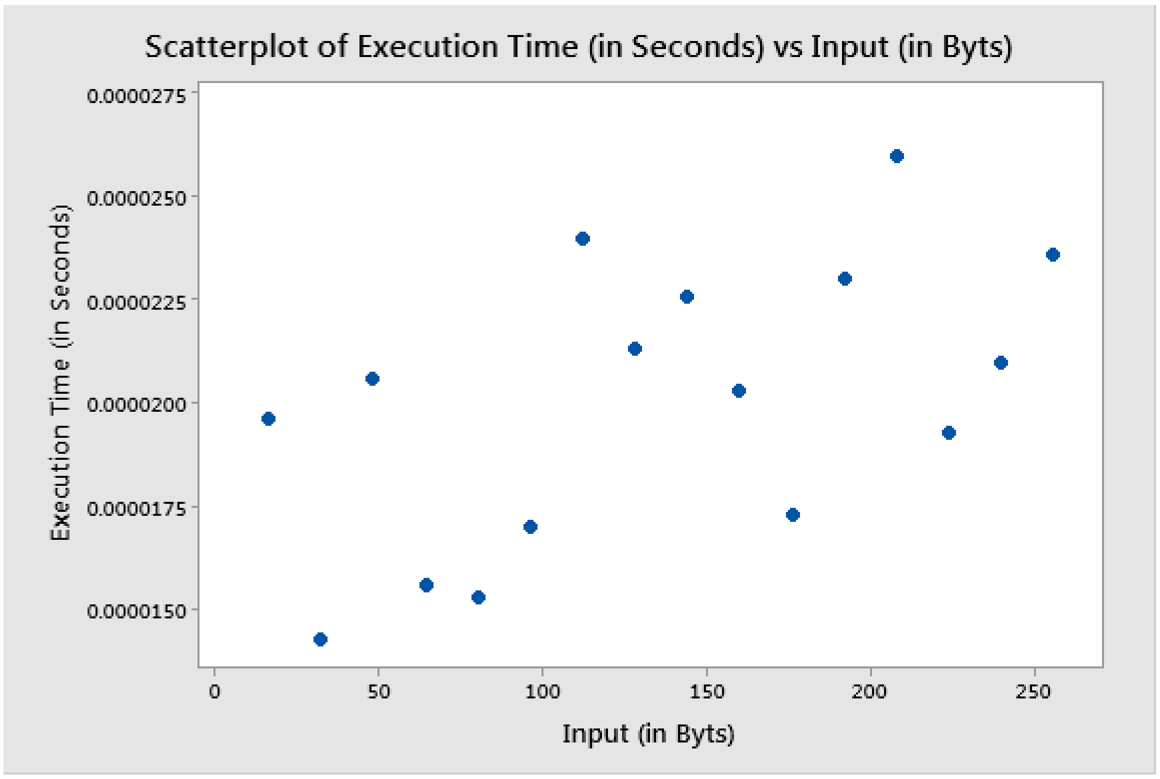

Fig 5. Scatter Plot of Shift Rows

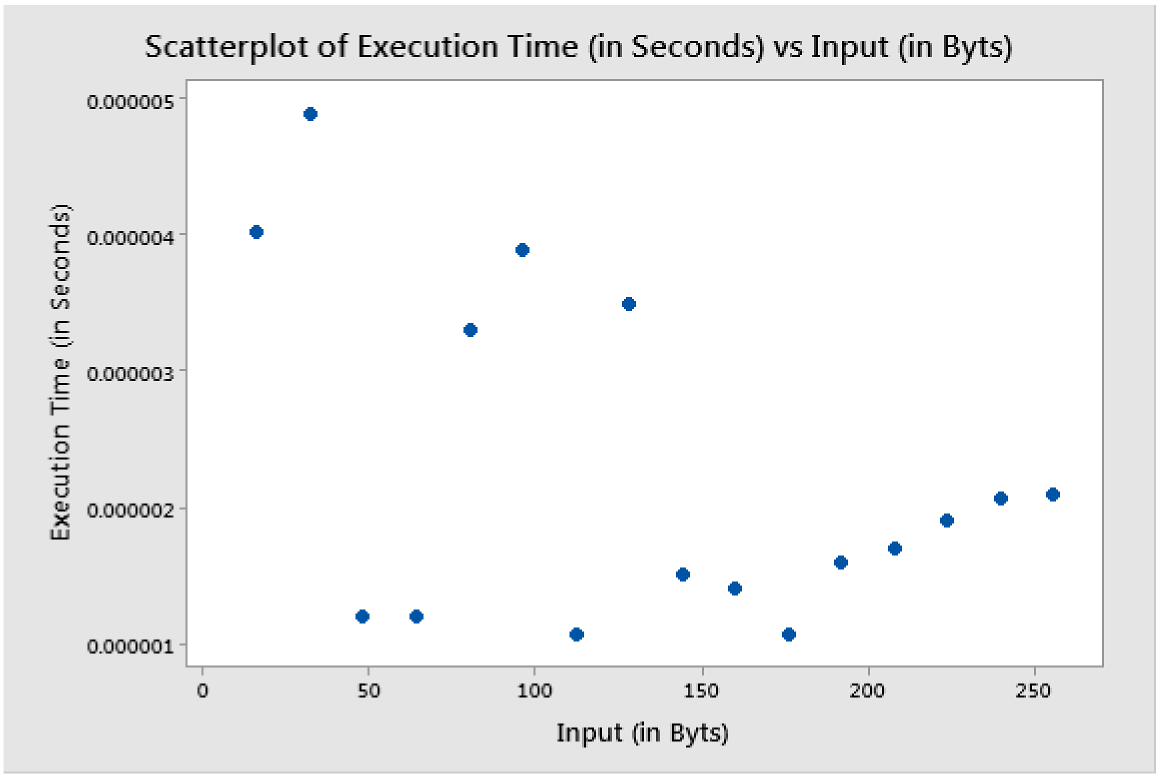

Fig 6. Scatter Plot of Add Round Key

For the "Mix Column" step, the first differences Δy are almost constant and the second differences Δy^2 are almost zero. So, a 1st-degree polynomial will approximate the empirical run time of mix column step as predicted through the *fundamental theorem of finite difference* and hence its empirical complexity can be termed to be *O (n)*. On fitting a statistical model on the mean execution time of mix column step, it shows almost a linear pattern which is shown in figure 7 below. The residual plot for this step is shown in figure 8.

Table 13: Difference Table for Four Intermediate Steps of AES-128

Input (in bytes)	Substitute Bytes (Δy)	Sift Rows (Δy)	Add Round Key (Δy)	Mix Column (Δy)	Mix Column (Δy^2)
16	0	-0.0000053	0.0000087	0.030667	
32	0.0000013	0.0000063	-0.0000037		
48	-0.000001	-0.000005	0	0.028833	-0.00183
64	0.0000037	-0.0000003	0.0000021		
80	0.0000024	0.0000017	0.0000006	0.028467	
96	0.0000007	0.000007	-0.00000014		
112	0.000003	-0.0000027	0.00000244	0.0291	0.000633
128	0.000005	0.0000013	-0.000002		
144	0.000006	-0.0000023	-0.0000001	0.0357	
160	0.0000023	-0.000003	0.00000034		
176	0.0000014	0.0000053	0.00000054	0.031067	-0.00463
192	0.0000023	0.0000003	0.0000001		
208	0.000003	-0.0000067	0.0000002	0.038	
224	0.0000017	0.0000017	0.00000016		
240	0	0.0000026	0.0000001	0.025733	
256	0	0	0		-0.01227

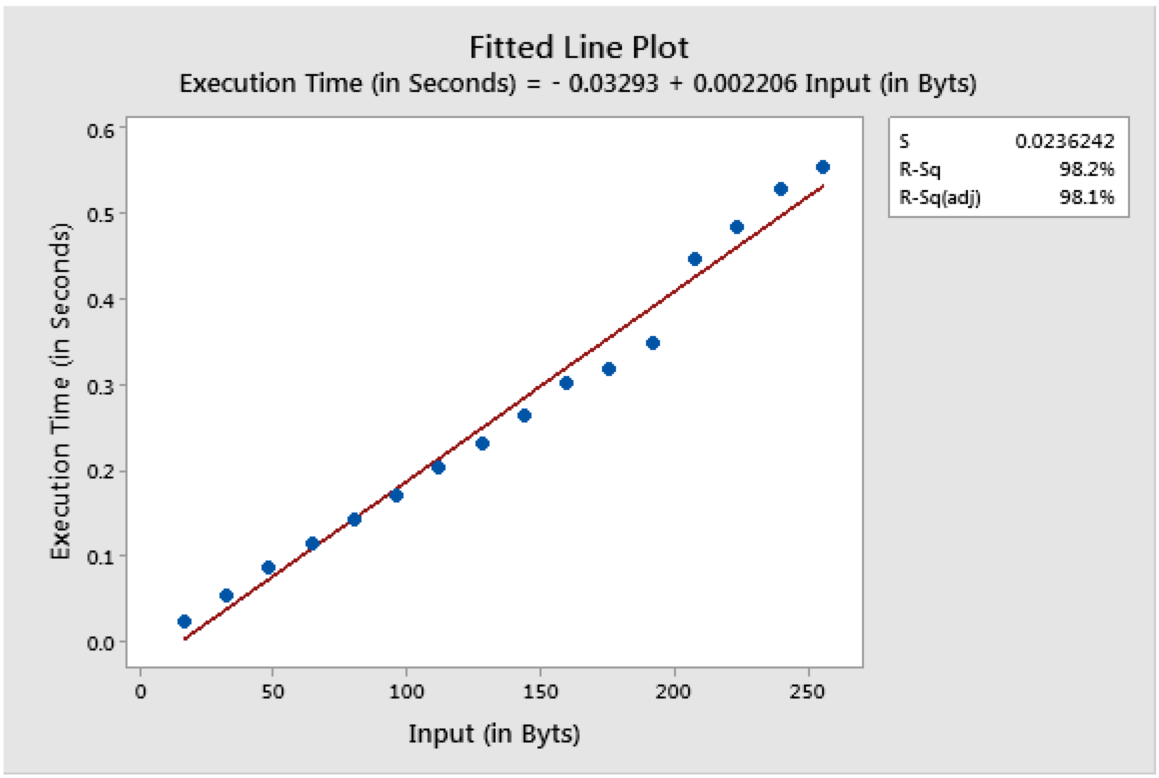

Fig 7: Statistical Model Fitting of “Mix Column”

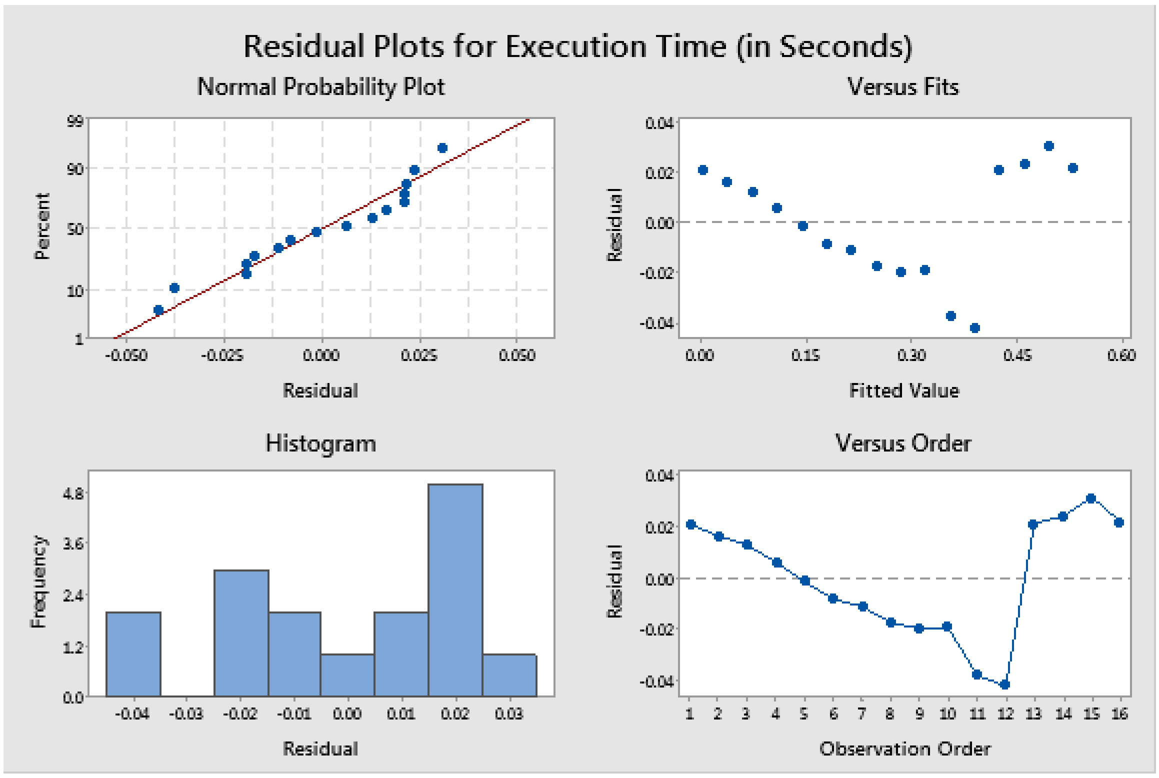

Fig 8. Residual Plot for "Mix Column"

The empirical complexity analysis of the four sub-steps involved in AES-128, we conclude that the mix column step has the highest complexity of *O (n)* while the rest others are *O (1)*. So, AES-128 empirical complexity accounts maximum for the empirical complexity of the mix column step. The overall complexity of AES-128 is *O (n)* by using any standard modes of operations because the most dominant operation is the mix column process.

6. Conclusion

In the case of compounding operations, the common proximity is towards the most dominant of the operations, working in a group when predicting the overall computational complexity of an algorithm. This may give vague result and whatever be analyzed theoretically may not be the actual run time of an algorithm specifically in case of compounding operations. So, we have used a statistical approach through the method of revised difference to analyze the statistical similarity between compounding operations for a well known cryptographic algorithm RSA. The two operations used in encryption and decryption procedure viz. *power* and modular arithmetic have been analyzed and the analysis shows that both the operations are statistically dissimilar. But, when predicted theoretically it is the modular arithmetic operation which is predicted to be more dominant of the two. So, we further used the fundamental theorem of finite difference method to empirically analyze the running time of both the operations separately. The empirical run time for power operation is *O (n)*, while in the case of mod operation, it is an empirical *O (1)*. So, we conclude that power and mod operations when working as a single operation in the encryption and decryption procedure of RSA are statistically dissimilar and it is the power operation with an empirical complexity of *O (n)* which is the more dominant.

Similarly, through the method of the fundamental theorem of finite difference, we analyzed the empirical complexity of all the sub-steps used in AES-128 which shows that the mix column sub-step is the most time consuming of all the 4 sub-steps used in AES-128 with an empirical complexity of *O (n)*.

Ethical Statement:

The authors declare that this research did not receive any funding and that they have no conflict of interest.

Authorship Contribution:

Prashant Pranav: Methodology and Writing

Sandip Dutta: Supervisor

Soubhik Chakraborty: Supervisor

References

1. A. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Algorithms,* Addison-Wesley, 1975
2. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms,* MIT Press, New York, 3rd edition, 2009.
3. D. H. Greene and D. E. Knuth. *Mathematics for the Analysis of Algorithms,* Birkhäuser, Boston, 3rd edition, 1991.
4. D.E. Knuth, “Fundamental Algorithms, Volume 1”, Addison-Wisley, 1973
5. W. Stallings, Network and Internetwork Security: Principles and Practice, Prentice Hall, 5th Ed., November 2009.
6. B.A. Forouzan, D. Mukhopadhyay, “Cryptography and Network Security”, 2nd edition, Tata McGraw-Hill, 2010
7. W. Diffie, S. Landau, Privacy on the line. The politics of wiretapping and encryption, MIT Press, 2007.
8. W. Stallings. “Cryptography and Network Security”, 4th edition, Pearson, 2005
9. A. Tanenbaum, Modern Operating Systems, Prentice Hall of India, 2001.
10. S. Chakraborty, (2007). A simple empirical formula for categorizing computing operations. Applied Mathematics and Computation, 189, 326–340. https://doi.org/10.1016/j.amc.2006.11.097
11. S. Chakraborty, P. Chaudhary, “A Statistical Analysis of an Algorithm’s Complexity”. Applied Mathematics Letters, 13, 2000, 121–126.
12. S. Chakraborty, S. K. Sourabh, “A Computer Experiment Oriented Approach to Algorithmic Complexity”, LAP LAMBERT Academic, 2010
13. S. Chakraborty, S.K. Sourabh, “On why an algorithmic time complexity measure can be system invariant rather than system independent", Applied Mathematics and Computation 190 (2007) 195–204
14. N.K. Singh, S. Chakraborty and D.K. Mallik, “A Statistical Peek into Average Case Complexity” International Journal of Experimental Desigh and Process Optimisation, Vol. 4. No. 2, 2014, 116-142
15. A. Schoor, Fast Algorithm for Sparse Matrix Multiplication, Information Processing letters, Vol. 15, No. 2, 1982, 87-89
16. S. Chakraborty, S.K. Sourabh, M. Bose, K. Sushant, “Replacement sort revisited: The ‘‘gold standard’’ unearthed!”, Applied Mathematics and Computation 189 (2007) 384–394
17. Announcing the ADVANCED ENCRYPTION STANDARD (AES)" Federal Information Processing Standards Publication 197. United States National Institute of Standards and Technology (NIST). November 26, 2001. Retrieved October 2, 2012.
18. Daemen, Joan; Rijmen, Vincent (March 9, 2003). "AES Proposal: Rijndael"
19. The Advanced Encryption Standard (Rijndael). 2010. http://www.quadibloc.com/crypto/co040401.htm (accessed March 10, 2019)
20. Trenholme, S. "S-box." AES. 2010. http://www.samiam.org/s-box.html (accessed March 10, 2019).

Figures

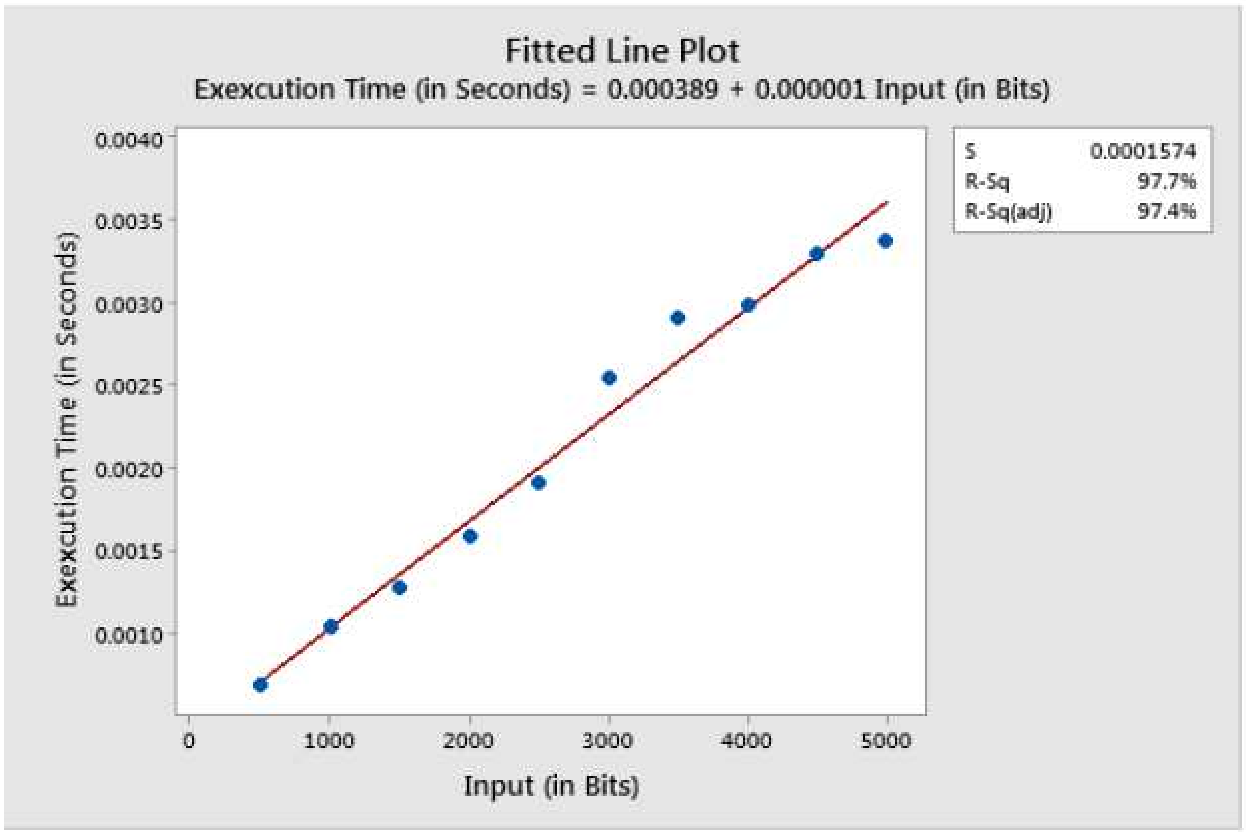

Figure 1

Statistical Model Fitting of Power Operation

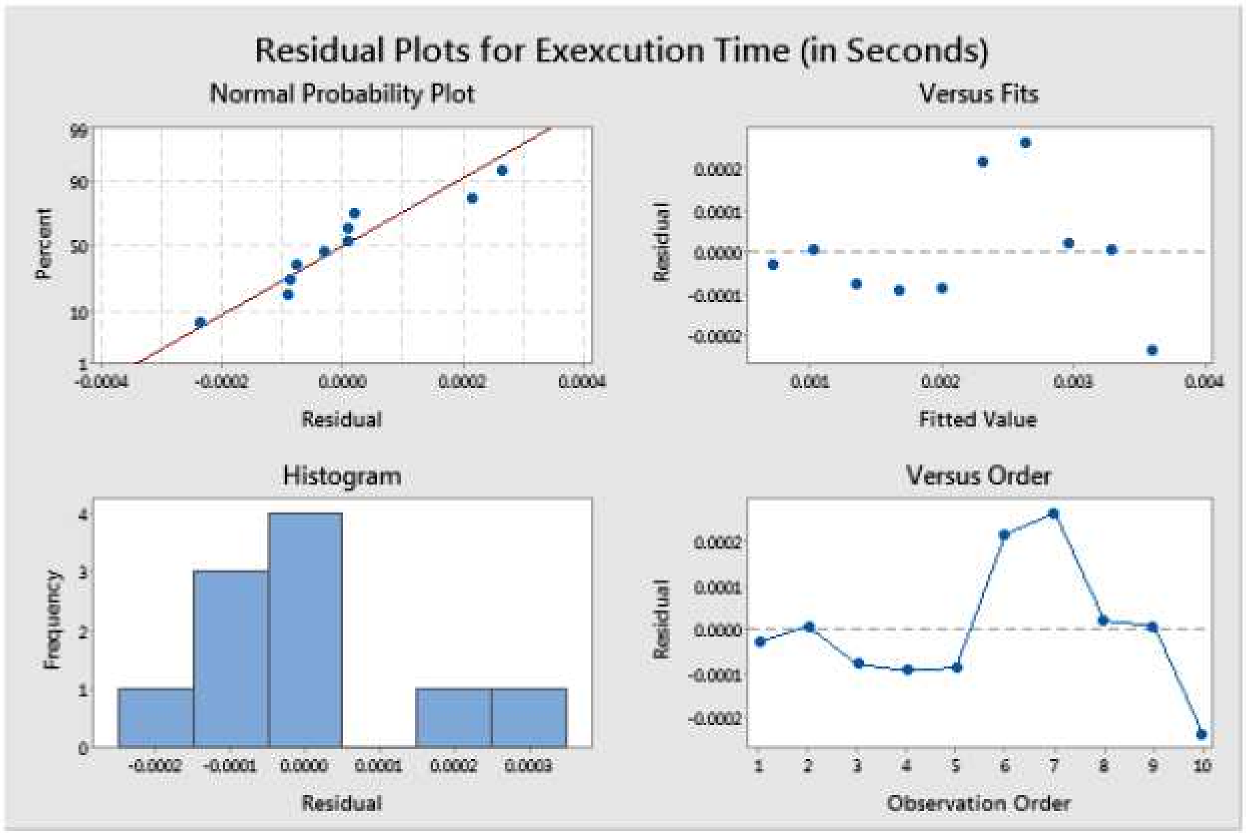

Figure 2

Residual Plot of Power Operation

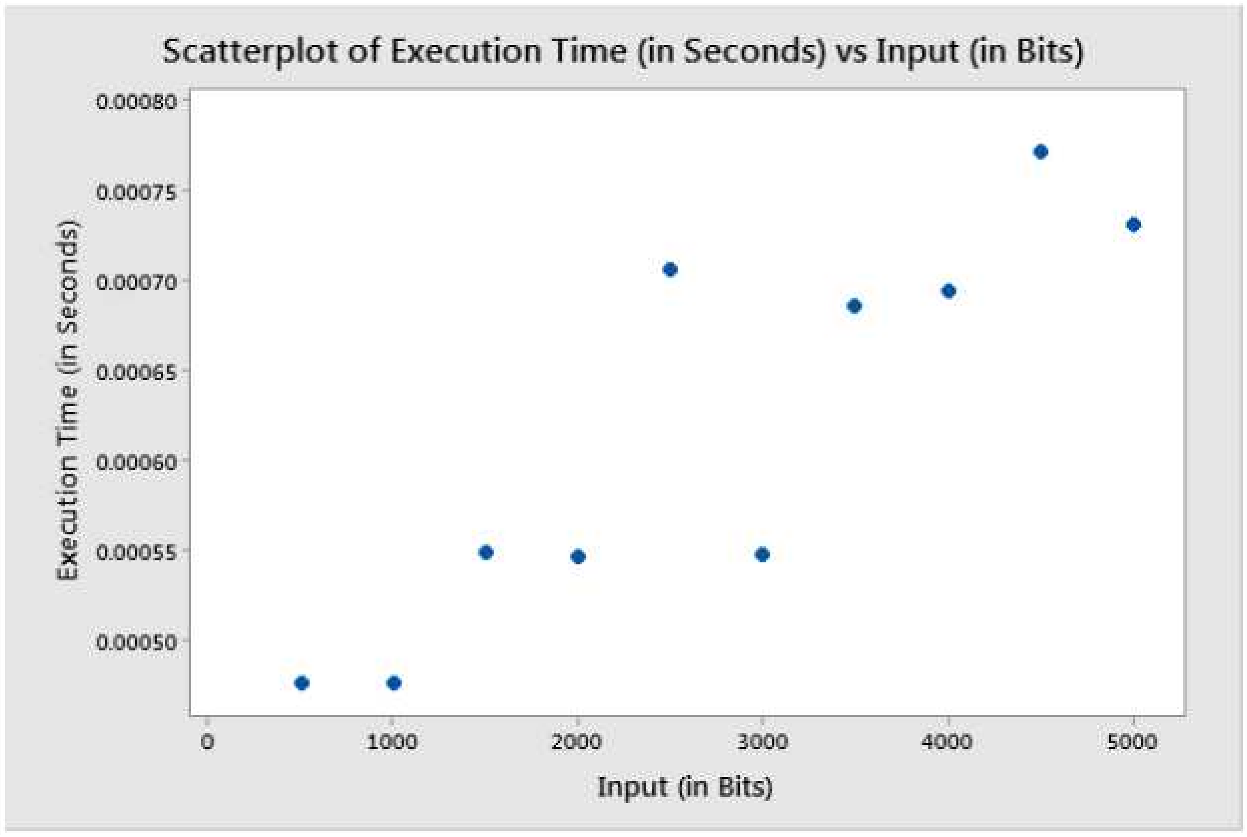

Figure 3

Scatter Plot of First Difference of “mod” Operation

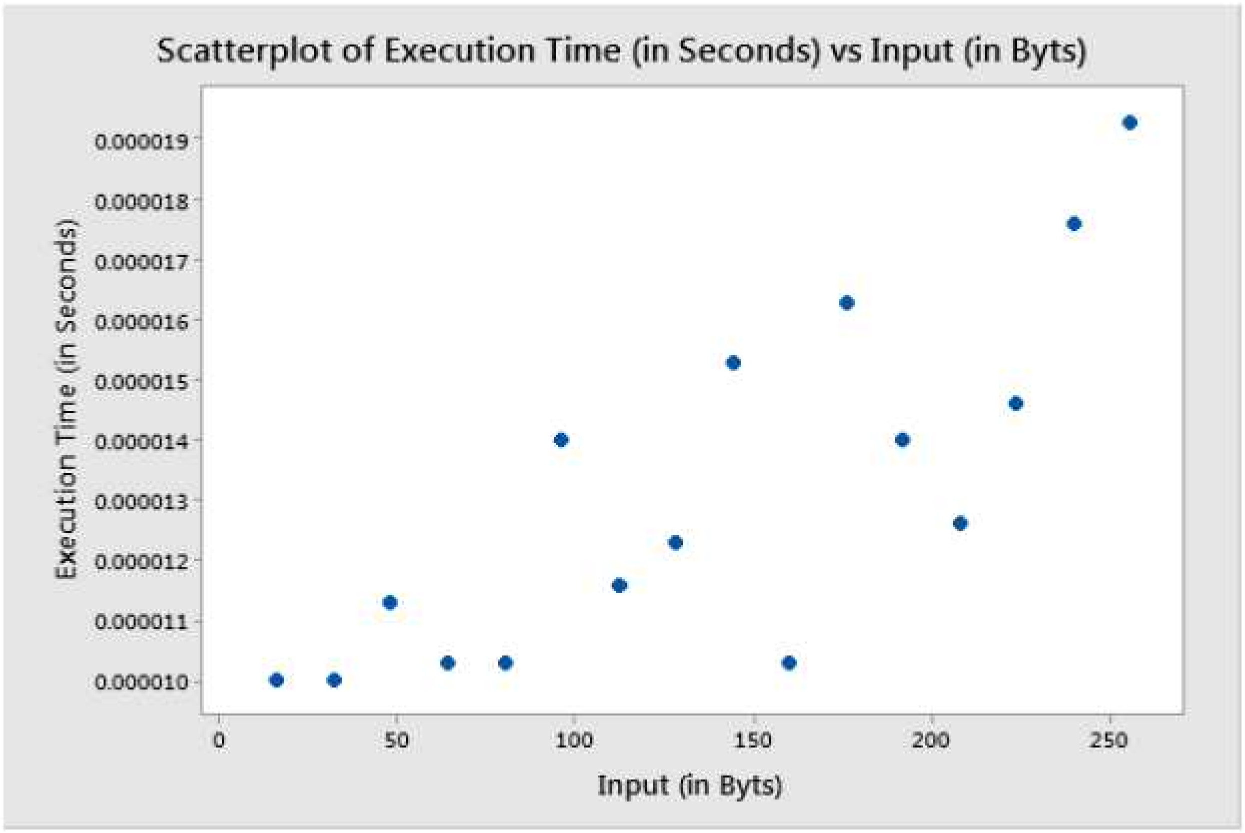

Figure 4

Scatter Plot of Substitute Byte

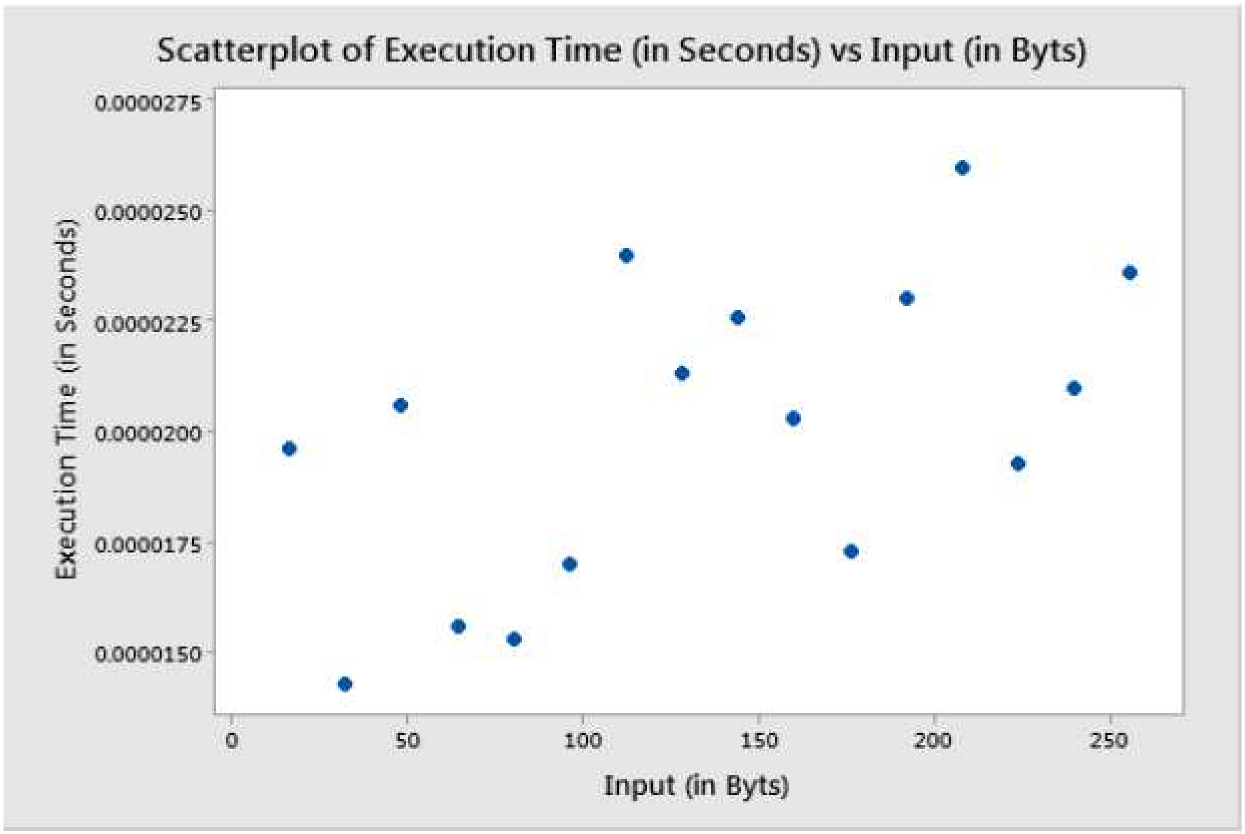

Figure 5

Scatter Plot of Shift Rows

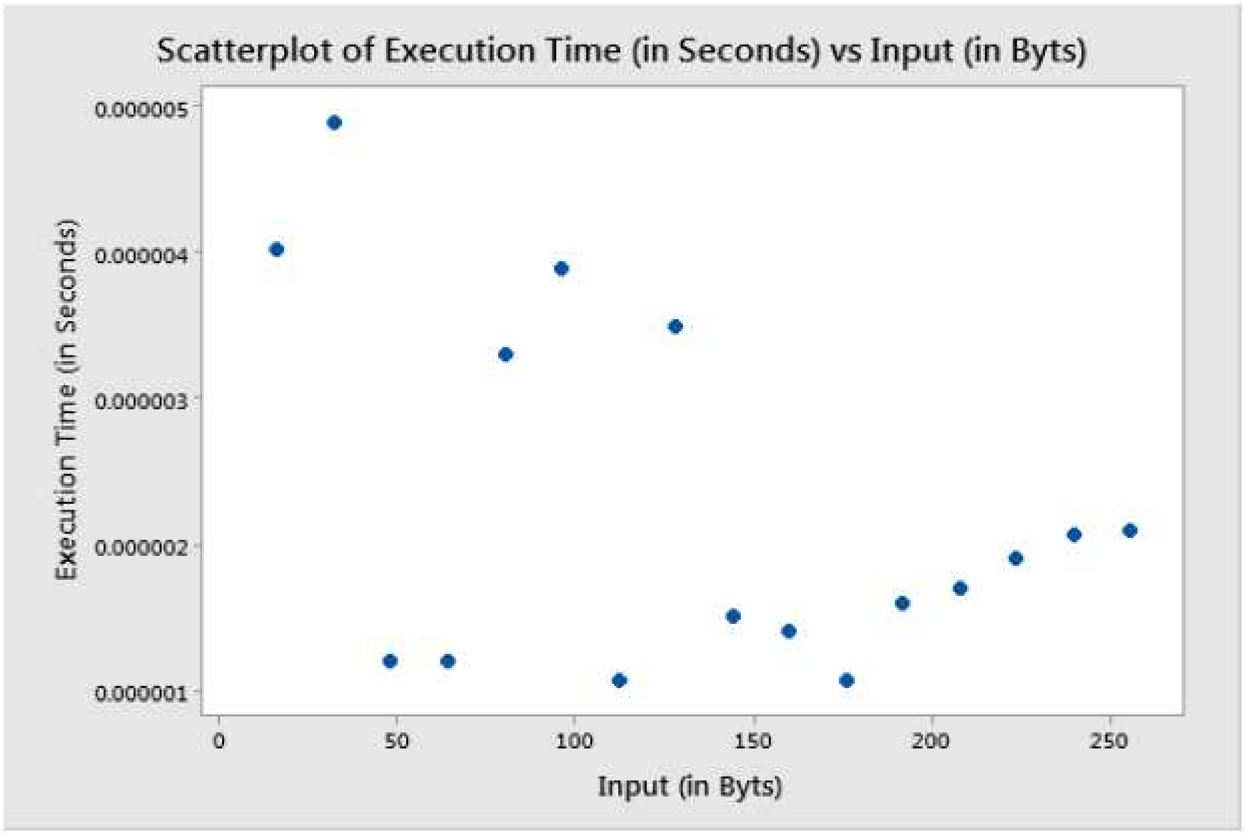

Figure 6

Scatter Plot of Add Round Key

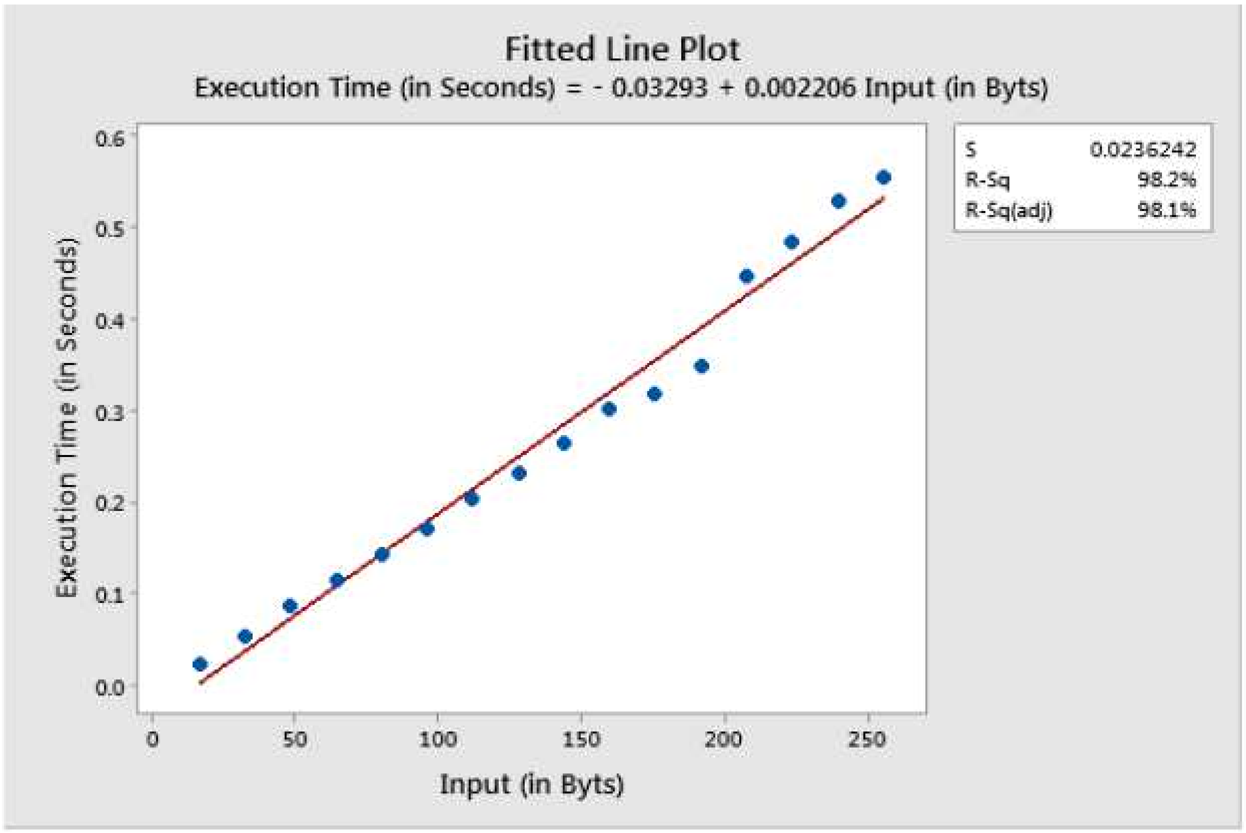

Figure 7

Statistical Model Fitting of “Mix Column”

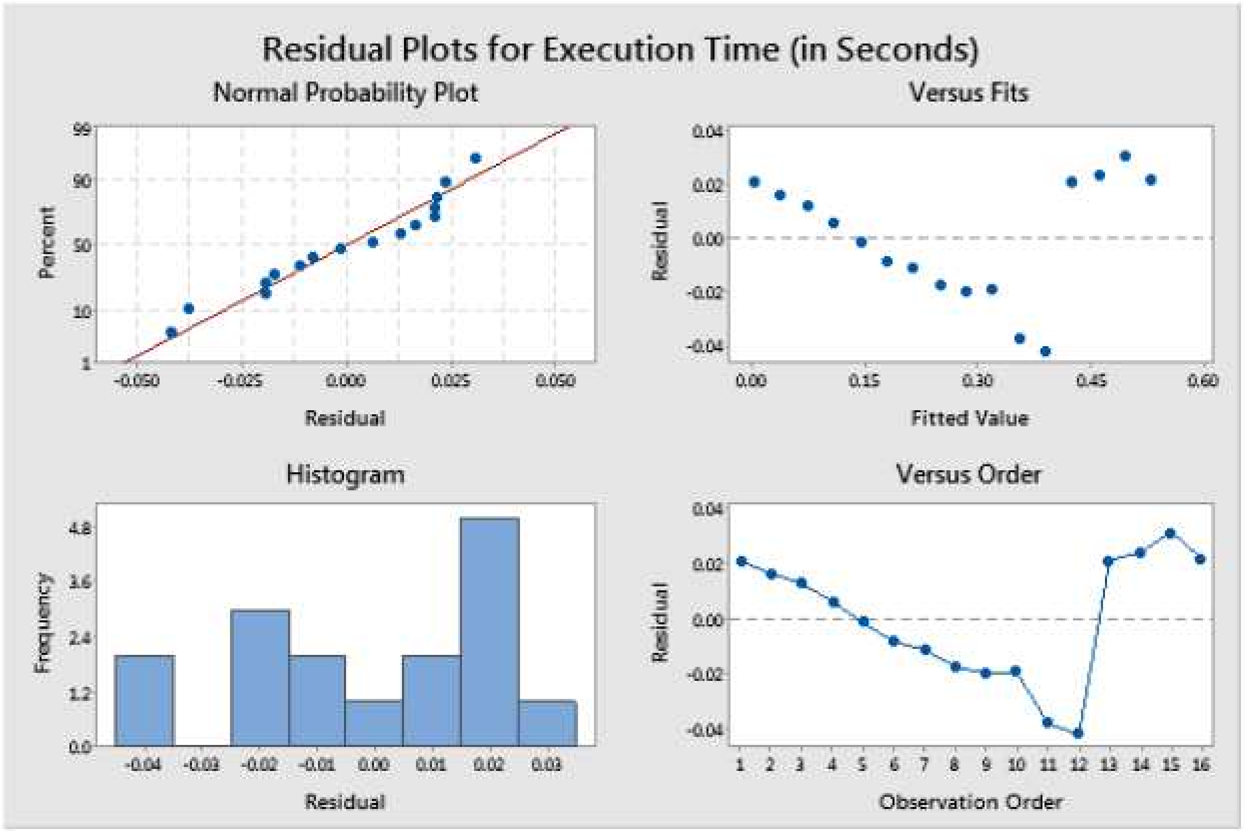

Figure 8

Residual Plot for “Mix Column”