Empirical and statistical comparison of intermediate steps of AES-128 and RSA in terms of time consumption

Prashant Pranav · Sandip Dutta · Soubhik Chakraborty

1 Introduction

Algorithm analysis (Aho et al. 1975; Cormen et al. 2009; Greene and Knuth 1991; Knuth 1973) 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 (Stallings 2009; Forouzan and Mukhopadhyay 2010; Diffie and Landau 2007; Stallings 2005) are composed of many complex mathematical functions which are generally compounded, a group of single operations. Not much of research has been done on analyzing a cryptographic protocol from its time complexity point of view. For a better understanding of how closely cryptography and complexity are related, see Talbot and Welsh (2006) and Rothe (2005). Both the encryption and decryption of two well-known symmetric key algorithms, namely RSA and ElGamal, 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 toward 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 Tanenbaum (2001).

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 inputs. 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. Time complexity of AES has been analyzed in Talbot and Welsh (2006) and Bogdanov et al. (2015).

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.

The authors’ published work in the related domain is given in Sect. 2. Section 3 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 Sect. 4, with a statistical comparison of power and mod operations used in encryption and decryption function of the RSA. Section 5 gives the empirical time complexity of the power and the mod operations. Introduction to AES-128 and its intermediate steps is discussed in Sect. 6, with an empirical analysis of all the intermediate steps to predict which of the sub-steps consumes the most time.

2 Authors’ published work, and of those who worked very closely, in the related domain

In this section, the contributions of the authors, and of those who worked very closely, in the related domain, are provided. We have split this section into two parts:

a) Existing research (Sect. 2.1)

This section provides a list of existing contributions of the authors, and of those who worked very closely, in the related domain before 2019.
b) New research (Sect. 2.2)

This section provides a list of new contributions of the authors, and of those who worked very closely in the related domain from 2019 onward.

| Sl. no. | Authors with year | Method/model | Steps/formula used | Program/tool/software used | Complexity |
|---------|------------------|--------------|-------------------|---------------------------|------------|
| 1       | Chakraborty and Choudhury (2000) | Fundamental theorem of finite difference | Step 1: Bubble sort algorithm was executed for varying input sizes and the execution time for each specific input was noted  
Step 2: The first and the second difference table of the execution time of bubble sort was calculated  
Step 3: From the inverse of the fundamental theorem of finite difference, the execution time was estimated to be \(O(x^2)\) in the worst case since a second-degree polynomial was able to fit the execution time | BASIC Programming language | Worst-case execution time of Bubble sort algorithm was estimated to be empirical \(O(x^2)\), where \(x\) is the input size |
| 2       | Coffin and Saltzman (2000) | Exploratory data analysis and summary statistics and MANOVA | Three different case studies were designed  
The first case compared the running times of two nonlinear optimization packages run on a set of 35 test problems  
The second case described an implementation of a primal–dual Newton barrier algorithm for linear programming  
The third case presented a more detailed analysis of the Fourer and Mehrotra data | PROC GLM and SAS/STAT user guide | For the first case study, the execution time was estimated to be: \(y = O(x)\)  
For the second case study, the execution time was estimated to be: \(y = 3.25x^{0.72}\)  
For the third case study, the execution time was estimated to be: \(y = 1.53x^{0.97}\)  
where \(y\) is the estimated execution time and \(x\) is the number of inputs |
| 3       | Cotta and Moscato (2004) | Evolutionary algorithms and statistical analysis | Two sorting algorithms, namely bubble sort and shell sort, are considered in the work  
Input to both the algorithms is the sequence of elements to be sorted and a key value  
Sequence to be sorted is chosen as a permutation of a set of natural numbers  
The task of evolutionary algorithm is to find the best suited hard permutation for a particular sorting algorithm | C Programming language | For bubble sort algorithm, the worst-case time complexity empirically was estimated to be: \(y = an^2 + bn\)  
with \(a = 0.5\) and \(b = -0.5\)  
For shell sort algorithm, the worst-case time complexity empirically was estimated to be: \(y = an^2 + bn\)  
with \(a = 0.0902\) and \(b = 8.5189\)  
where “\(y\)” is the estimated execution time and “\(n\)” is the input size |
| Sl. no. | Authors with year       | Method/model                          | Steps/formula used                                                                 | Program/tool/software used     | Complexity |
|--------|-------------------------|---------------------------------------|-----------------------------------------------------------------------------------|--------------------------------|------------|
| 4      | Chakraborty and Sundararajan, (2007) | Revised difference method, bootstrap method, Box–Muller transformation and t-test | Compounding operations “+” and “*” were compared to check whether the two operations are statistically similar when operating in a group  
Ho: The operations are statistically similar  
H1: The operations are statistically dissimilar  
From all the three methods, namely revised difference method, bootstrap method and Box–Muller transformation, the t-statistics showed a significant result and so in the null hypothesis was accepted  
| C Programming language | The operations “+” and “*” were statistically similar and equally complex when operating in a group |
| 5      | Chakraborty and Sundararajan (2007) | Fundamental theorem of finite difference | Winograd’s algorithm for two $n \times n$ matrix multiplication was carried out, and the time for varying input sizes was noted and the complexity of the algorithm was measured empirically using the inverse of the fundamental theorem of finite difference  
| C++ Programming language and Minitab statistical software | Empirical $O(n^2)$ complexity in two $n \times n$ matrix multiplication ($n$ even) in the worst case |
| 6      | Chakraborty and Sourabh (2007) | Revised difference method | The revised difference method was applied on Amir Schoor algorithm for multiplying two sparse and dense matrices, and the result shows that the Amir Schoor algorithm can be executed in empirical $O(n^2)$ complexity  
| C++ Programming language and SPSS analysis tool | Empirical $O(n^2)$ complexity is certainly gettable in the worst case with both matrices dense, the pre-factor matrix approximately as dense as triangular and the post-factor matrix fully dense |
| 7      | Goldsmith et al. (2007) | Empirical study method | Choose a program to profile  
Choose workloads \{$w_1,...,w_k$} for the program  
Describe the workloads with numerical features \{$f_1,...,f_k$, $g_1,...,g_k$}, for example, the number of bytes in an input file or the number of nodes in a graph  
Measure program performance; run the program on each workload; and record the cost of each basic block, $\ell$, as a $k$-vector: \{$y_1,\ell,...,y_k,\ell$}  
Trend-prof predicts performance in terms of features, fitting the performance measurements, $y$, to features of the program’s input, $f$. The authors used linear models, $y = a + bf$, and power-law models, $y = af^b$  
| C Programming language | Quicksort is $O(n \log n)$ in the size of the array; depth-first search is $O(e)$ in the number of edges in the graph in the worst case |
2.2 New research

| Sl. no. | Authors with year | Method/model | Steps/formula used | Program/tool/software used | Complexity |
|---------|-------------------|--------------|---------------------|---------------------------|------------|
| 8       | Singh and Chakraborty (2011) | Empirical study method | Introduced a new sorting algorithm, namely partition sort, and compared its execution time empirically | C++ Programming language and SPSS statistical tool | The complexity of the algorithm was estimated to be empirical $O(n^2)$ in the worst case |
| 9       | Singh et al. (2013) | Regression analysis | Experimental analysis is focused around three models, namely $y = b_0 + b_1n$, $y = b_0 + b_1n + b_2n\log_2n$ and $y = b_0 + b_1n + b_2n\log_2n + b_3n^2$, which are referred to as n/linear, $n\log_2n$ and $n^2$/quadratic model, respectively | Minitab statistical software and SPSS statistical tool | $y_{\text{avg}}(n) = O(n^2)$, at least for $k \leq 25,000$ in the worst case |
| 10      | Singh et al. (2014) | Regression analysis and t-test | The samples generated consist of all distinct elements (at least theoretically). The program runtime data obtained for sorted sequences are fitted for a quadratic model. The regression analysis result is given in Box 1. With a very significant t-value (194.92) of quadratic term, the regression analysis statistic strongly supports a quadratic model. The quadratic model goodness is further tested through cubic fit for the same runtime dataset | SPSS statistical tool | Worst-case complexity model for quick sort was proposed as: $y_{\text{worst}}(n, t_d) = b_0 + b_1n^2 + g(n, t_d) + \varepsilon$ |

The smart sort, which is designed by combining the key functions of standard quick and heap sort algorithms, is expected to behave optimally with respect to the different input parameters. The robustness of the average-case $O(\text{emp}(n\log_2n))$ complexity for smart sort is conjectured as result of study for regression models and factorial design experiments. The claimed robustness is further reverified by studying its behavior for inputs obtained from Cauchy distribution model which is an unusual continuous distribution model for which expectation does not exist theoretically.
3 Related work

The statistical similarity among three well-known operations addition, multiplication and equality has previously been studied through the method of revised difference [see (Chakraborty 2007)]. 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 Chakraborty et al. 2007). 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 \( O(x^2) \), 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). 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 (Chakraborty and Sourabh 2010)]. 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 give 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, \( \Delta^n P_n(x) = C = \text{Constant} \)

\[ \Delta^{n+r} P_n(x) = 0, \quad r = 1, 2, 3, \ldots \]

where \( \Delta \) is the forward difference operator defined as \( \Delta f(x) = f(x + h) - f(x) \) for values of \( f(x) \) tabulated at \( x = a, a + h, a + 2h, \ldots, h \) 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 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 the 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.” (Chakraborty and Sourabh 2010).

The authors in Chakraborty and Sourabh (2007) 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 \( O \), are 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 (Singh et al. 2013). 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 Schoor (1982), 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 Schoor 1982)) 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 (Chakraborty et al. 2007)]. In Singh and Chakraborty (2011), the authors verified the complexity of a Smart Sort algorithm through computer experiments. An improved Pig Latin algorithm was proposed in Dutta and Sanyukta. (2020), and its complexity was measured empirically.

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

\[
C = M^e \mod n,
\]

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:

\[
M = C^d \mod n,
\]

where \( d \) is the private exponent.

4 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 three 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 \( (t_p, and t_m) \) be the mean run time of the three trials for the “power” and “mod” operations for \( i = 1, 2, 3, \ldots \) corresponding to \( r \) instances of input size. The revised difference is defined as:

\[
d_i = \frac{(t_{p_i} - t_{m_i})}{(t_{p_i} + t_{m_i})/2}, \quad i = 1, 2, 3, \ldots, r. \tag{1}
\]

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 Eq. (1) generally has a small variance, when \( c \) and \( d \) are the simplest operations considered. However, this may not be so always (e.g., 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 \( t \) (Eq. 2) is same as that of

| 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 |
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 $E(X) = E(Y)$.

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:

$H_0 = \text{The mean of the population of revised difference is zero.}$

$H_1 = \text{The mean of the population of revised difference is not zero.}$

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

| Table 2 | Execution time for “mod” operation (small dataset) |
|---------|-----------------------------------------------|
| 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 & 0.000669 & 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 |

| 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 & 0.0001 |
| 4000 & 0.00284 & 0.00032 & 0.0001 |
| 4500 & 0.00316 & 0.00033 & 0.0001 |
| 5000 & 0.00349 & 0.00033 & 0.0001 |
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:

$$
t = \frac{d_m}{d_s / \sqrt{r}}
$$

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.
\[ d_m = \frac{d_1 + d_2 + \cdots + d_r}{r} \quad \text{and} \quad (3) \]
\[ d_s = \left[ \frac{1}{(r-1)} \sum_{i=1}^{r} (d_i - d_m)^2 \right]^{1/2}. \tag{4} \]

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 \( \alpha = 0.05 \) is 2.262. Since the calculated \( t \)-statistics is greater than the tabulated one, 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.

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.

5 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 Sect. 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

**Table 5** First differences for “mod” operation

| Input (in bits) | Mean execution time in sec for \( x \mod y \) | 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                                    |                  |

- 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.

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, 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**.
emphasizing that as the input keeps varying, the second 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 first-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).

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 in Figs. 1 and 2, respectively. These were plotted using the statistical package MINITAB version 16.

The fitted line plot in Fig. 1 shows a linear fitting of the power operation, and so, it can be termed to be $O(n)$. The residual plot above has four 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 are 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. Table 5 shows 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.

5.1 Implications of various plots

**Normal probability plot** The normal probability plot is a graphical technique for addressing the distribution of data, i.e., whether the data are distributed normally or not. The points of the data should form an approximately straight line for the data to be considered as normally distributed. This is the case in our plot, and so, the data seem to be coming from a normal distribution.

**Versus fit** Residual versus fits plot is a scatter plot with the y-axis containing the residuals and the x-axis the corresponding estimates (fitted value). Our versus fit plot suggests that there is no unusual data point in the dataset. It also suggests that the residuals and the fitted values are correlated, which means that the fitted model which we predicted in our case to be linear is actually the case as the residuals are centered on zero throughout the range of the fitted values.

**Histogram**: Histograms of the residuals suggest whether the data are skewed or include outliers. If we get one long tail in one direction, this means that the residuals are skewed, and if one bar is far away from others, it means the
residuals include outliers. These both conditions are not seen in our Histogram.

Versus order This plot displays the order of collection of residual data. This is used to investigate whether the residuals are independent of each other or not. In an ideal scenario, the residuals on the plot should be distributed randomly around the normal line. This is what we observe in our encryption residual plot.

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 three 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 in Fig. 3.
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 Tables 6 and 7, and the revised differences are shown in Table 8.

From Table 3 of the revised difference, we get:

$$d_m = 1.449099, \quad d_i = 0.356954 \quad \text{and} \quad t = \frac{d_m}{d_i} = 24.01709.$$ 

Since $t = 24.01709$ which is greater than 1.96 at 95% level of confidence, 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 $O(1)$.

### 6 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 AES (2001), Daemen and Rijmen (2003), The Advanced Encryption Standard (Rijndael) (2010) and Trenholme S “S-box.” AES (2010).

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

We applied the fundamental theorem of finite difference approach discussed in Sect. 2 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., $\Delta y$, 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

| Table 8 Revised difference between “power” and “mod” operations |
|-----------------|----------------|----------------|-----------------|
| Input (in bits) | Mean execution | Mean execution | Revised         |
|                | time in sec for | time in sec for | difference      |
|                | $x^2$           | $xmody$         |                 |
| 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        |
shown in Figs. 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.

For the “mix column” step, the first differences \( \Delta y \) are almost constant and the second differences \( \Delta y^2 \) are almost zero. So, a first-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 Fig. 7. The residual plot for this step is shown in Fig. 8.

By 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.

### 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                   |
7 Conclusion

In the case of compounding operations, the common proximity is toward 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.

| 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        | 0.000014 | 0.000010 | 0.000012 |
| 64                                       | 0.0000019        | 0.000012 | 0.000007 | 0.000012 |
| 80                                       | 0.0000014        | 0.000017 | 0.000008 | 0.000033 |
| 96                                       | 0.0000013        | 0.000016 | 0.000009 | 0.000039 |
| 112                                      | 0.0000016        | 0.000015 | 0.000019 | 0.0000106 |
| 128                                      | 0.0000017        | 0.000018 | 0.000007 | 0.000035 |
| 144                                      | 0.0000020        | 0.000014 | 0.000011 | 0.000015 |
| 160                                      | 0.0000016        | 0.000017 | 0.000010 | 0.000014 |
| 176                                      | 0.0000021        | 0.000016 | 0.000013 | 0.0000106 |
| 192                                      | 0.0000017        | 0.000019 | 0.000012 | 0.000016 |
| 208                                      | 0.0000018        | 0.000016 | 0.000017 | 0.000017 |
| 224                                      | 0.0000022        | 0.000019 | 0.000016 | 0.000019 |
| 240                                      | 0.0000020        | 0.000023 | 0.000019 | 0.0000206 |
| 256                                      | 0.0000023        | 0.000021 | 0.000019 | 0.000021 |

| 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 |
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 four sub-steps used in AES-128 with an empirical complexity of $O(n)$.

The method proposed can be used to estimate the time complexity of the stated cryptographic algorithms before deploying them for use in any real-life application. The running time of an algorithm greatly enhances the overall cost of any system, and so knowing it before hand is always
a better approach while keeping in check that the appropriate security is not loosened.

As far as the limitation of the proposed approach is concerned, we can say that since algorithms are platform independent (i.e., a given algorithm can be implemented in an arbitrary programming language on an arbitrary computer running an arbitrary operating system), there are additional significant drawbacks to using an empirical approach to gauge the comparative performance of a given set of algorithms. The time factor depends much on the computer clocks, and the results can get affected if the clocks do not function properly.
Declarations

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

References

Aho A, Hopcroft JE, Ullman JD (1975) The design and analysis of algorithms. Addison-Wesley, Boston
Announcing the ADVANCED ENCRYPTION STANDARD (AES) (2012) Federal Information Processing Standards Publication 197. United States National Institute of Standards and Technology (NIST). November 26, 2001. Retrieved October 2, 2012
