# Shor's Algorithm

## Exploring the Use and Modification of the Difference of Two Cubes in Post-Processing

Paper written by Darcy Adams

May 2023

Supervisor: Sam Braunstein

Submitted for the degree of MSc in Computer Science at the University of York

## ACKNOWLEDGEMENTS

Firstly, I would like to thank my project supervisor, Sam Braunstein for his support and expertise during this project.

And thank you to my friends and family, who's help and encouragement was never unappreciated throughout this year.

## TABLE OF CONTENTS

## TABLE OF FIGURES

## TABLE OF TABLES

## EXECUTIVE SUMMARY

The aim of this project is to demonstrate the potential to improve the efficiency of Shor's factoring algorithm to solve large number factoring. Many current encryption methods rely on the premise that factoring large composite numbers is difficult using classical computing due to time complexity. Shor's algorithm demonstrates the potential for using quantum algorithms to outperform classical computers to obtain the multiplicative period finding required by later steps in the algorithm. The post-processing factorisation step requires the discard of all odd orders, and it is not always successful in finding nontrivial factors for even orders, all requiring a restart of the computationally expensive quantum order finding step.

The success of this project was determined by the identification of improvements to the efficiency in the post-processing factorisation step achieved with modifications to the factoring method. It was found that using a generalised formula based on the difference of two cubes extended the number of useable period orders to include odd orders and so provided multiple opportunities for calculation in the post-processing step. The generalised formula created in this project provided a higher probability of success to obtain nontrivial factors than the current standard of Shor's algorithm within a single call to the quantum subroutine. The scope of this project investigated only where the number to be factorised is a product of two primes. The results of this project provide empirical evidence that the generalised formula demonstrated greater than 95% success in finding nontrivial factors for 5 digit numbers with any order whereas the standard Shor's algorithm success probability was 81% for 5 digit numbers with even orders.

Today's dominant cryptographic standard of RSA continues to be based on an accepted understanding that factoring a very large composite number has an unfeasible time complexity for classical computers, making the encryption key unbreakable in practicality. Current quantum computing technology is time consuming to set up specified circuits to perform a single calculation to obtain the period order. The motivation for this project was to investigate the post-processing steps to identify if reductions in the discard of this expensive quantum step could be made. Whilst current quantum computing is insufficient to calculate the period order for the magnitude of the numbers used in these current cryptographic standards (2048-bit keys, approximately 617-digit length), any efficiency improvement in the post-processing steps will continue to be applicable as the quantum technology develops, improving the ability of the algorithm to break current encryption within a usable timeframe, making this type of encryption method insecure.

The improvements in the post-processing steps demonstrated by this project have ethical and commercial implications for the field of cryptography. Many companies, individuals and governments use cryptographic key methods based on the difficulty of factorising large numbers to secure sensitive information. Any improvement in the ability to factorise large numbers has the potential to reduce the security of this data and undermine the foundation of these cryptographic methods. Whilst the efficiency gains achieved in this project are significant, the scope of the project was limited to 5-digit numbers due to time complexity constraints using classical computing and insufficient ability of current quantum computing technology to find the multiplicative order period of large numbers. With current cryptography standards creating number keys with over 600 digits, this current project remains more of a theoretical possibility without the ability to be tested with these large numbers. Significant improvements in quantum computing would be required before it can be applied to such large numbers and sufficient improvements are not expected in the near future. As such, this project does not have any immediate ethical or commercial concerns that need to be considered.

.

# 1. INTRODUCTION

In this project, I aim to demonstrate the potential of using the difference of two cubes factorisation method as a post-processing step to expand the range of r periods that can be utilised to find nontrivial factors of a composite number efficiently using only a single call to the period function f(x).

Shor's algorithm, first introduced by mathematician Peter Shor in 1994, is a quantum algorithm for integer factorisation, which is widely regarded as one of the most important problems in cryptography. The algorithm has been a subject of intensive research and development in the field of quantum computing and cryptography due to its potential to break public key cryptography systems such as RSA.

Many current cryptography schemes, with RSA being the leading scheme used, rely on the premise that large numbers that are a product of two large primes are extremely computationally expensive to factorise using classical computers. There is an exponential increase in the time required to break that number using classical computers for each bit increase in the number size. Since 2015, the NIST (National Institute of Standard Technology) standard for RSA has been 2048-bits for encryption which has effectively been unbreakable for brute force attacks in a reasonable timeframe. Shor’s algorithm is important as it is the first algorithm that demonstrates that a quantum computer has the potential to factorise a large number into primes in logarithmic time which would render current encryption methods obsolete.

The period-finding subroutine of Shor's algorithm is the key component that enables efficient factorisation of composite numbers. The subroutine is based on finding the period of a modular exponential function and has been implemented using various mathematical functions, including trigonometric functions and elliptic curves to identify the periodic patterns for a number(s). The period finding subroutine is the computationally expensive part of Shor’s algorithm and is the only part of the algorithm that requires quantum computing. Not all period values calculated can be used in the standard Shor’s algorithm and odd periods are always discarded.

In recent years, there has been growing interest in the application of quantum computing in cryptography and the development of quantum algorithms for solving mathematical problems, however, the majority of research has focused on the quantum aspects of the algorithm and less attention has been given to the pre -or post-processing parts of the algorithm.

The difference of two cubes factorisation method is an alternative to the difference of two squares method used in the standard Shor's algorithm. The difference of two cubes method has been found to be less efficient than other factorisation methods, but it has the advantage of allowing for an odd period r.

My motivation for this project was to investigate whether the difference of two cubes, or a modification of this, could be used as an alternative algorithm to the current difference of two squares. The main project aim was to investigate whether the use of odd periods could improve the probability of success in finding nontrivial factors of a composite number using a single call to the period function f(x), reducing the waste of the computationally expensive order finding. As quantum computing evolves to find the r period of larger numbers, any post-processing efficiency gains from this project will remain valid and applicable, rendering many current encryption methods insecure.

In order to investigate the aim of this project, the following approach will be implemented:

A feasibility analysis to explore if there is a sufficient occurrence of odd order periods to justify an investigation into an alternative post-processing factoring step.

Investigation into the probability of success using the difference of two cubes (DTC) and modified versions for odd period r.

Comparison of the probability of success of modified algorithms to identify the algorithm that provides the highest probability of success.

## 2. LITERATURE REVIEW

### 2.1. Shor's factoring algorithm (SFA)

Shor's factoring algorithm (abbreviated to SFA for the remainder of this paper) is comprised of 5 steps, starting at step 1 through to step 5, all of which can be performed on a classical computer except step 2, which requires the use of a quantum computer. The semiprime number that SFA will be applied to will be "N", the guess(s) that are used for N will be "a" and the period calculated for these numbers will be "r". The quantum computer step 2 will be explained in section 2.1.2.

#### 2.1.1. Traditional SFA step-by-step[1]

STEP 1. Choose a random positive integer $1 < a < N$, then use the Euclidean algorithm (polynomial time) to calculate $gcd(a, N)$. Resulting in the *greatest common divisor* of $a$ and $N$.

If the $gcd(a, N) \neq 1$, then a non-trivial factor of N has been found and we are done. If, however, the $gcd(a, N) = 1$, we then proceed to **STEP 2**.

**STEP 2**. Compute the unknown period[2] $r$ of the function:

$$N \xrightarrow{f_N} N \tag{1}$$

$$r \mapsto a^r mod\ N \tag{2}$$

STEP 3. If $r$ is an odd integer, revert[3] back to STEP 1, else proceed to STEP 4

STEP 4. Since $r$ is even

$$(a^{\frac{r}{2}} - 1)(a^{\frac{r}{2}} + 1) = a^r - 1 = 0\ modN \tag{3}$$

If $a^{\frac{r}{2}} + 1 = 0\ modN$, then revert back to STEP 1. Else if $a^{\frac{r}{2}} + 1 \neq 0\ modN$, proceed to STEP 5.

STEP 5. Using the Euclidean algorithm to calculate $p = gcd(a^{\frac{r}{2}} - 1, N)$, as $a^{\frac{r}{2}} + 1 \neq 0\ modN$, $p$ is a non-trivial factor of N. *End of steps*.

Thus, the algorithm has produced a non-trivial factor from an odd

---

[1] Steps as explained by S. J. Lomonaco[5]

[2] When determining the factors N from the period r of the function f(x) ≡ a^r (mod N), the period is the smallest value r = x such that a^x(modN)=1

[3] This is the step(s) we will be looking at in more detail later.

positive integer N.

### 2.1.2. Quantum step

To illustrate the period finding method in **STEP 2**, a worked example can be used to show the methodology step by step. The example below uses N = 91 as the composite number to be factored.

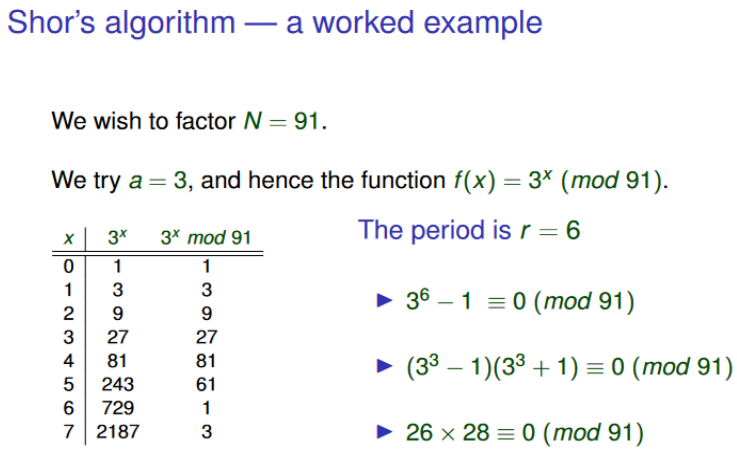



*Figure 1. Shor's period finding example from lecture slides [6]*

In the figure above, the guess chosen for $a$ is 3, and the exponent is referred to as $x$ as opposed to the assigned value I use in this project which is $r$. It should be noted that prior step in SFA has already been performed where the $gcd(a = 3, N = 91) = 1$.

To begin, $a^x$ and $a^x mod\ N$ are calculated; shown $3^x\ and\ 3^x mod 91, (x = 0) = 1$. $x$ is then incremented and as in the figure shown it can be seen at $x = 6$ that the periodic sequence has been found from $x = 0\ to\ 5$, thus the period r for this N is 6.

With this period now found we can carry this answer over into the next step(s) of the algorithm to determine a prime factor(s) of N. Steps performed after the period has been found will be referred to as post-processing steps.

The reason why this step has to be performed on a quantum computer is due to the fundamental differences in the way classical and quantum computer's function. In a classical computer, all the logic gates used are non-reversible[4], while quantum computers have all reversible logic gates no matter how many inputs are used. This allows for unique *quantum* circuits to be created which make quantum computers distinct from classical computers in the way they

[4] Except the NOT gate

perform calculations but is also the reason why quantum computers have many limitations in other types of operations when compared to classical computers. Using a classical computer to perform the modular arithmetic period for this algorithm leads to an exponential increase in the time for finding $r$ with increasing sizes of N, while a quantum computer can perform these calculations in polynomial O(logN). This is due to the efficiency of the quantum Fourier transform and the modular exponentiation by repeated squarings on quantum algorithms. The quantum Fourier transform, or QFT as it will now be referred, is a function that is applied to a superposition of states, which allows the algorithm to extract information about the period of the function being analysed. Specifically, SFA uses the fact the Fourier Transform (FT) of a periodic function is concentrated at certain frequencies, allowing it to estimate the period of the function by measuring the FT of the superposition of states.

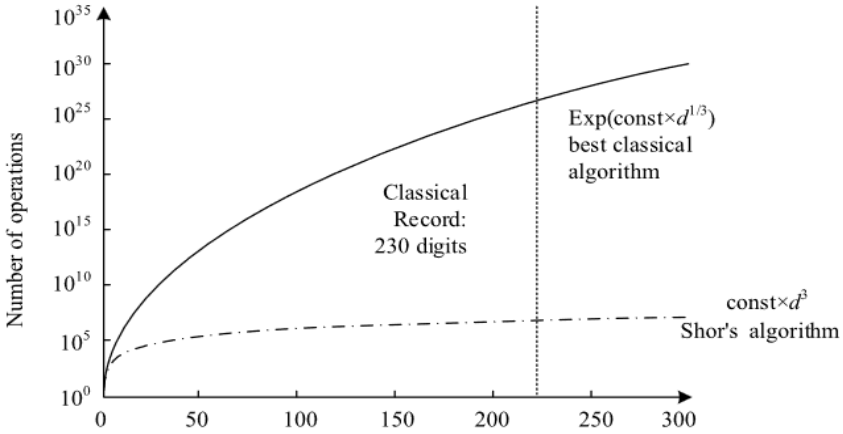



*Figure 2. Time complexity algorithm comparison [4]*

### 2.1.3. Standard SFA discard of odd $r$ values

The standard version of Shor's algorithm is designed to work with even values of the order r and explicitly states in step 3 that if r is an odd integer, it is discarded and reverts to step 1. The primary reason for this is that the Euclidean factorisation method uses r/2 as a exponent which rarely leads to an integer result. Whilst the standard SFA discards odd values of r, there are some special cases where odd orders can lead to an integer result. For example, if N is a composite number where p and q are two prime factors that are very close together, then an odd value of r can be used to find one of the prime factors by exploiting the fact that $a^{\frac{r}{2}} mod\ N = -1(mod\ N)$ when r is odd. However, this approach only works for a small set of composite numbers and it is not applicable to most large numbers.

Some odd values of r can be converted to even values by using a

technique called "adding multiples" but this can be computationally expensive and is not practical for large numbers.

Therefore, while there may be some special cases where odd values of the order r can be used by SFA for factoring certain numbers, it is generally more efficient to stick to even values of r due to the small pool of numbers that this applies to.

### 2.1.4. SFA failure

There are several instances where standard SFA can fail to find the nontrivial factors of a number.

- The number to be factored is not composite: SFA is designed to factor composite numbers, so if the number to be factored is prime, the algorithm will fail.[5]
- The number to be factored is too large: SFA has a computational cost that scales exponentially with the number of digits in the input number. As a result, the algorithm may become impractical for factoring large numbers beyond the capabilities of current quantum computers.
- The quantum computer used to run the algorithm is noisy: Quantum computers are highly susceptible to errors and noise can cause errors in the quantum gates used in SFA. If the errors are too large, the algorithm may fail to produce the correct result.
- The quantum computer used to run the algorithm does not have enough qubits: SFA requires a large number of qubits to be able to factor large numbers efficiently. If the quantum computer does not have enough qubits, the algorithm may not be able to produce the correct result.
- An odd r period rarely leads to an integer value of a in standard SFA[6]

## 2.2. Difference of two squares vs difference of two cubes

In the standard SFA steps, as detailed in 2.1.1, step 3 explicitly states that if an odd value period was obtained from the quantum step produced it is to be discarded and the algorithm will return to step 1 again. This results in the waste of the heavy quantum computation performed already and repeated looping back to step 1 until an even r value is obtained. An alternate post-processing approach is to use an alternative factorisation method to the

[5] The odd integer N chosen for the algorithm should always be an odd ‘Semiprime’.

[6] Certain specific circumstances that are usually related to an a value chosen being a square number can lead to the odd period r still producing results in standard SFA difference of two squares

difference of two squares that is able to use odd r values, allowing the quantum power that has already been performed to not be wasted and also to increase the efficiency of the factoring algorithm by reducing the amount of calls needed to the subroutine. The post-processing factorisation method identified for the scope of this project is the difference of two cubes. As opposed to the difference of two squares used in standard SFA, the difference of two cubes is a scalable way to factorise using any period found and so is not subject to the same limits of difference of two squares using even periods. A comparison of the factorisation methods is shown below:

Difference of two squares (squares + even incrementing):

$$(a^{2n} - b^{2n}) = (a^n - b^n)(a^n + b^n), \quad (4)$$

Difference of two cubes (cubes + odd incrementing):

$$(a^n - b^n) = (a - b)(a^{n-1} + a^{n-2} \cdot b^1 + a^{n-3} \cdot b^2 + \ldots + b^{n-1}) \quad (5)$$

As

$$f(x) \equiv a^r (mod\ N) \quad (6)$$

$$a^r \equiv 1 (mod\ N) \quad (7)$$

$$a^r - 1 \equiv m.N \quad (8)$$

With even period r obtained from step 2, the difference of two squares formula can be shown as:

$$a^r - 1 \equiv (a^{r/2} - 1)(a^{r/2} + 1) = m.N \quad (9)$$

With any period r obtained from step 2, the difference of two cubes formula can be shown as:

$$a^r - 1 \equiv (a - 1)(a^{r-1} + a^{r-2} + a^{r-3} + \ldots + a^1 + 1) = m.N \quad (10)$$

The difference between the two formulae is that there is no divisor of the period in the exponent in difference of two cubes formula therefore the r exponent will always remain an integer removing the limitation [7]of difference of two squares using odd periods. Thus, difference of two cubes formula allows us to use the odd period values found by the quantum step into this alternate post-processing equation, keeping whole integers which can be used to find the nontrivial factors of N[8].

Difference of two cubes worked example:

$$N = 115, a = 6$$

[7] Note there are certain circumstances where this may not be the case and “a” value to the exponent(s) of a float number still result in a whole number; further to this, this resulted value still may not produce non-trivial factors.

[8] It is worth noting that the prior steps are kept the same and it is only the post-processing where a change may occur depending on whether the period value produced is odd or even.

Using a quantum computer for **STEP 2**, we get $r = 11$

Substituting these values into the difference of two cubes method

$$\Rightarrow (6-1)(6^{10}+6^9+6^8+6^7+6^6+6^5+6^4+6^3+6^2+6+1)$$

$$\Rightarrow (5)(72559411) = m \cdot N$$

$$p.q = N, p = 5, q \Rightarrow gcd(72559411,115) = 23$$

This results in the difference of two cubes successfully finding the nontrivial factors of N as 5 and 23.

In comparison, substituting these values into the standard difference of two squares method

$$\Rightarrow (6^{11/2}-1)(6^{11/2}+1) = m \cdot N$$

$$\Rightarrow (19046.2322399)(19048.2322399)$$

Gcd fails and factors cannot be found through use of this odd order in standard SFA.

### 2.3. Relevant research into SFA

The primary implementation of quantum mechanics in current quantum computing development is through the use of qubits and qubit entanglement, however, as the number of qubits increases, there is a proportionate loss of accuracy due to decoherence and noise causing qubit entanglements to be lost too quickly before the required calculations can be completed. The highest number to date that SFA has computed faster than classical computing, when implemented on a quantum computer, is 21. A recent Google paper, 'Suppressing quantum errors by scaling a surface code logical qubit', published by "Google Quantum AI"[1] has brought forward a possible technique to attempt to overcome this limitation by combining many qubits into what the paper calls a "logical" qubit, maintaining all qubit entanglements and allowing for greater computational power. They spread the information being processed in the quantum computer across a number of qubits in a way that meant the system as a whole could retain enough information to complete its calculation; even when individual qubits fell out of their quantum states. So far, the research shows a reduction of only 4% in the error rate as Google scaled up its technique to run on larger quantum systems but researchers noted that this was the first time when increasing the size of the computer had not also led to a rise in the error rate. This "breakthrough" in error correction was the result of improvements Google made to all components of its quantum computer and shows that while current quantum computing is still limited in its application of finding the modular periods necessary for post-processing factorisation of large numbers using SFA, the research in this field is

moving closer to making quantum computing applications feasible.

Thomas Lawson considered odd value orders in the paper ‘Odd orders in Shor’s factoring algorithm’[3] with the conclusion that odd orders can be useful in factoring and should not be discarded but that while factors can sometimes be found from odd orders when a is square, improving the efficiency of SFA can be achieved more easily by avoiding square *a* coprimes, resulting in a small improvement to SFA by using the remaining odd order values. The paper noted that quantum subroutines are currently harder than classical to implement with error-prone young, imperfect technology which may require reconfiguration each time the calculation changes, which are costly in terms of SFA run time, which he expects to continue for some time. Lawson concludes that any small improvement in SFA efficiency by using a limited selection of odd orders can lead to substantial savings in runtime at the current stage of quantum computing development.

An extension to the application of standard SFA was proposed in research by Anna M. Johnston, ‘Don’t throw away odd orders’[2], where the standard factorisation step of SFA could be applied to any of the prime roots of N. While the standard function’s approach is to rerun the costly quantum step of the algorithm if the order was odd or an unusable even, this extension of the method not only avoids the quantum step rerun but, depending on the order factorisation, can allow for multiple attempts at factoring N.

In the paper ‘On completely factoring any integer efficiently in a single run of an order finding algorithm’[x], Martin Ekera demonstrated with a high probability of success that a single run of the quantum step of SFA is usually sufficient to efficiently find the complete factorisation of N in polynomial time using a classical algorithm attributed to Miller, which is the standard factorisation method for difference of two squares, with negligible computational cost in the post-processing step. Ekera noted that by multiplying on or dividing off small prime factors of r to provide a larger range of gcd comparisons led to a higher probability of success in the recovery of the complete factorisation of N. Grosshans and Lawsons et al.[8] also found that if a small prime factor q divides r, then gcd((g^(r/q) − 1) mod N, N) is likely to yield non-trivial factors of N in the paper ‘Factoring Safe Semiprimes with a Single Quantum Query’.

In the paper 'Improving the Success Rate of Quantum Algorithm Attacking RSA Encryption System' (Yumin Dong et al) [4], the authors proposed two new optimisation schemes based on the principle of SFA which could be applied to a small range of suitable odd value orders. The standard optimisation scheme of SFA was to discard odd orders at step 3 and restart at step 1. The authors found that an odd cycle obtained when *a* was a perfect square number can

randomly be effective. They also found that an odd order that was a multiple of 3 could be effective when using the new optimisation scheme. By modifying the decomposition method for these two schemes, the requirements for the period could be relaxed without affecting the algorithm complexity.

Yumin Dong et al paper, abbreviated to “Dong’s” for the remainder of this paper, provided a significant foundation for the development of this project. Dong’s research exploring multiples of 3 odd orders in SFA and the modification of the SFA algorithm influenced my own difference of two cubes research and provided a start point for modifying the difference of two cubes algorithm. Whilst Dong’s paper provided the probability of success for a suitable odd order for their modified algorithm being produced using a very small example size of 3-digit N values, it did not provide any probability of success data for any larger data selection and so was limited for comparison analysis. Due to this, as part of my project, Dong’s algorithm was implemented to gather data for my selected N values to compare the probability of success against my own algorithms[9].

[9] Note the same data Dong’s paper gathered was reproduced with my own implementation to ensure my results were correct.

## 3. DESIGN & ANALYSIS

This section explains the methods and techniques that will be used to implement SFA standard factorisation and the difference of two cubes factorisation method, in addition to modifications to the algorithms inspired by research explored in the literature review. Difference of two cubes will be referred to as DTC for the remainder of this paper.

The exploration of this project is focused on the post-processing probability of success in finding nontrivial factors of N for a provided period order r. Given that this project is purely theoretical in nature, without access to any quantum computing software like Qiskit, the methodology for comparing DTC and SFA will involve primarily theoretical analysis and results obtained through classical computation. Due to the time complexity of order finding on classical computers, a limit of 5-digit numbers were used for the main data selection as this provided an acceptable balance between computation time and sufficiently large sample size.

It is worth noting that all numbers used in this project are odd semiprimes, i.e. safe semiprimes, as these are the only numbers selected for cryptography and so these were used in this project as appropriate for SFA. A semiprime is a number that is the product of 2 prime numbers. The exact numbers used can be found in appendix [6.2].

To compare SFA and DTC, the probability of success of finding the nontrivial factors will be analysed using even orders for SFA and odd orders for DTC. Throughout the rest of this paper, the probability of success is determined as finding the nontrivial factors of any N. The mathematical functions used for both these factorisation methods have been outlined in section 2.2. and I created python functions for each. The set of composite integers in appendix [6.2] were executed and the probability of success was calculated for each N with a selection of *a* values (see section 3.1.). This probability was calculated by iterating through every *a* (i.e. guess) between $1 < a < (N-1)$ for each N and recording if each N, *a*, r combination found nontrivial factors for N using the selected factorisation method.

To ensure data validity, a random sample of factors was obtained using both methods to compare the results from paper calculations and third-party software to provide a high level of confidence in the accuracy of the python program. In addition, error analysis was implemented throughout the python program development to identify and eliminate any limitations or issues with inaccuracies that arose. Final probability values for each order of N were calculated using the average of the probability of success for each N, which were then arranged into order bands N1 to N4 (see section 3.1. for band details). These averages were rounded to the nearest 3 decimal places which will introduce rounding errors in further calculations, however, these rounding errors have been accepted as having negligible impact on any patterns that can be extrapolated.

Additional functions were created in python to calculate the probability of success results for Dong's algorithm and a modified version of DTC, which will be abbreviated to OptDTC for the remainder of this paper. For all factorisation methods, comparable data restrictions were used to limit bias in the datasets and results gained.

The data created by the algorithms will be analysed and evaluated in order to enable conclusions to be drawn on whether odd orders using DTC or modifications to DTC can provide an increase in the probability of success in post-processing when comparing to SFA.

### 3.1. Data selection criteria

Due to the time complexity of calculations on a classical computer, a limited selection of semiprimes was used for the input N values for this project. To ensure this selection was representative of all semiprimes, the selected numbers were subdivided into sets defined by their number of digits. These subsets will be referred to as bands for the remainder of this paper and have been defined as follows:

N1 band represents 10^1 numbers (2 digit numbers)
N2 band represents 10^2 numbers (3 digit numbers)
N3 band represents 10^3 numbers (4 digit numbers)
N4 band represents 10^4 numbers (5 digit numbers)

This band representation will continue for larger numbers, for example, N10 would represent 10^10 numbers (11 digit numbers).

Tests were performed to ensure the data was not skewed by limiting the selection. Excluding the first band, N1, which encompasses all its available 15 semiprime choices, the other bands have been limited to segments of 30 numbers using the first 10, middle 10 and final 10 semiprimes as a suitable compromise between time complexity and data accuracy.

| N order | N% encapsulation (Number of N) | a% | DTC (Success) | Total execution time (s) |
|---|---|---|---|---|
| N1 | 100% (15) | 100% | 0.981 | <1 |
| N2 | 16% (30) | 100% | 0.720 | <1 |
| N2 | 100% (185) | 100% | 0.702 | 5 |

*Table 1. Time taken to calculate P(Success) for different amounts of N where r(Odd)*

From table 1. analysing the data for N2 numbers, the probability of success for the limited 30 semiprimes is 0.720 when compared to 0.702 all 185 semiprimes in the N2 band. This has been accepted as an acceptable accuracy to demonstrate that a limited 30 semiprimes can represent the success rate for all Ns of that category whilst remaining at a practical execution time.

As the magnitude of N increases beyond N2, using all *a* values for each N became a time constraint, therefore an alternative approach was used where only a percentage of guesses (*a*) are selected for a given N. Various amounts of selected guesses were tested to identify the optimum balance between data accuracy vs time complexity (see table 2.).

| | a% of N | | | | | | |
|---|---|---|---|---|---|---|---|
| N Order | 0.1% | 0.5% | 1% | 5% | 10% | 50% | 100% |
| N1 | — | — | — | — | 0.857 | 0.913 | 0.918 (t<1) |
| N2 | — | — | 0.286 | 0.496 | 0.519 | 0.723 | 0.720 (t<1) |
| N3 | — | — | 0.552 | 0.580 | 0.631 (t=24) | 0.637 | 0.635 |
| N4 | 0.461 | 0.500 | 0.534 (t=1800) | — | 0.544 Time > 5hr | — | — |

*Table 2. DTC probability of success using percentage of total "a" guess for N order (t = seconds)*

The final percentages of a for each band were highlighted yellow in table 2 and identified as follows:

N1: a = 100% of N
N2: a = 100% of N

N3: a = 10% of N
N4: a = 1% of N

A selection of N1 numbers were evaluated to identify the feasibility of investigating the use of odd order r in factoring algorithms (see table 3). The probability of an odd order occurring was approximately 0.153 (approximately 15%), which was evaluated as a large enough initial probability to make continued research viable.

| N1 (10^1) | P(Even order) | P(Odd order) |
|---|---|---|
| 21 | 0.818 | 0.182 |
| 33 | 0.750 | 0.250 |
| 35 | 0.870 | 0.130 |
| 39 | 0.875 | 0.125 |
| 51 | 0.968 | 0.032 |
| 55 | 0.875 | 0.125 |
| 57 | 0.771 | 0.229 |

*Table 3. Probability of even/odd orders for selected N*

This was extended to then calculate the average probability of even vs odd order for an entire band (see table 4).

| N order | Avg Prob(Even) | Avg Prob(Odd) |
|---|---|---|
| N1 | 0.868 | 0.132 |
| N2 | 0.851 | 0.149 |
| N3 | 0.871 | 0.129 |
| N4 | 0.851 | 0.149 |
| N5 | 0.861 | 0.139 |

*Table 4. Average probability of even/odd orders for increasing order of N (a = 100%)*

It can be seen from table 4 that as the magnitude increases, the trend of an odd order occurring stays approximately the same around 14%. This must be taken into consideration when evaluating the success in using odd orders versus the standard discard but for this project it was considered a sufficient quantity to make the project investigation viable. The individual N probabilities of even versus odd

orders for each band is shown in appendix [6.3].

Further on in the paper, multiples of small prime factors of r were explored for 3, 5, 7, 11, 13, 17, 19 and 23. This group will be referred to as “period multiples” and analysis will be limited to this list unless otherwise specified.

## 3.2. DTC algorithm

### 3.2.1. Design of DTC

To recap the function for difference of two cubes we have this formula:

$$a^r - 1 \equiv (a-1)(a^{r-1} + a^{r-2} + a^{r-3} + \ldots + a^1 + 1) \qquad (11)$$

The useful feature of this factorisation method capitalises on its versatile application to present opportunities to find non-trivial factors with r period values other than 2. Because the formula does not divide the exponent as occurs in standard SFA factorisation, all results are integers and there is no occurrence of fractional results which is a limitation of standard SFA.

A limitation with the design of DTC for larger r values is the second bracket becomes more computationally expensive as r increases.

The python code for DTC was run using the numbers in bands N1 to N4 outlined in 3.1 (see Appendix 6.2 for all N values).

### 3.2.2. Analysis of DTC

Each dot in figure 3 represents the average probability of success for each N. This same data is shown more clearly in figure 4 when the order bands are shown as a linear order spread. The dot colours represent the bands showing N1 (blue), N2 (red), N3 (green) and N4 (cyan).

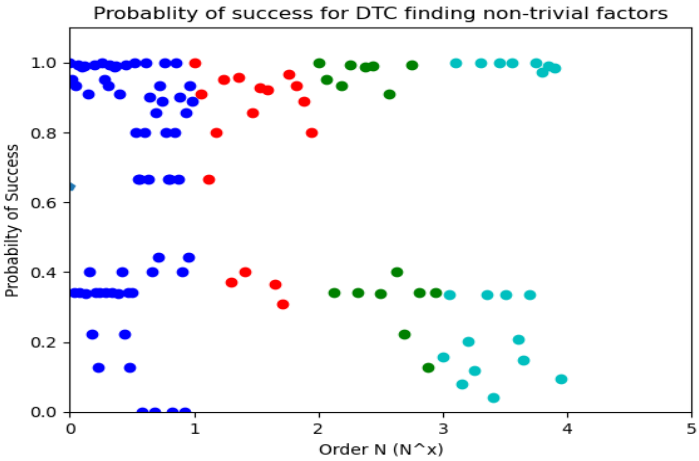



*Figure 3. Probability of success for DTC in finding nontrivial factors.*

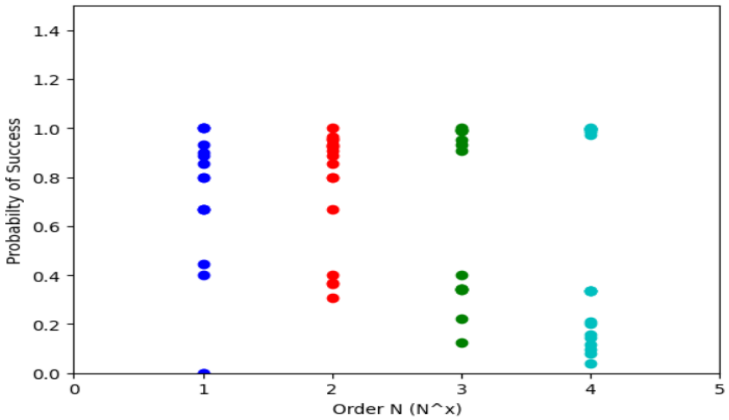



*Figure 4. Probability of success for DTC in finding nontrivial factors shown as a linear order spread.*

From figure 3 and 4, it can be seen that as the number magnitude increases, the success probability begins to trend closer to a very high or low success rate for individual Ns with little spread across the probability distribution. The percentage of highly successful Ns reduces as the magnitude increases so that while some individual Ns are highly successful in finding nontrivial factors, the average trend of success probability for each band decreases (see table 5). This information is also available as a chart (see appendix 6.4).

| N order(a%) | Prob(Odd) | Prob(success) | Prob(failure) |
|---|---|---|---|
| N1 (100) | 0.132 | 0.918 | 0.082 |
| N2 (100) | 0.149 | 0.720 | 0.280 |
| N3 (10) | 0.129 | 0.631 | 0.369 |
| N4 (1) | 0.149 | 0.534 | 0.466 |

*Table 5. Average probability(success) for DTC using r(odd) where a% is the percentage of guesses of N.*

Whilst the probability of success for N1 to N4 is greater than 0.5 and so is comparative to standard Shor's, if the downward trend were to continue in a similar pattern, the probability of success would become very small for very large N values such as those used by RSA encryption.

In order to identify any possible patterns for small prime factor success of r orders, the data was analysed for the period multiples specified in 3.1 (see table 6).

| N order (a%) | DTC (success) | r(3) | r(5) | r(7) | r(11) | r(13) | r(17) | r(19) | r(23) |
|---|---|---|---|---|---|---|---|---|---|
| N1 (100) | 0.981 | 0.856 | 0.833 | 1.0 | 1.0 | - | - | - | - |
| N2 (100) | 0.720 | 0.613 | 0.558 | 0.661 | 0.597 | 1.0 | 0.333 | 1.0 | 0.666 |
| N3 (10) | 0.631 | 0.482 | 0.538 | 0.833 | 0.355 | 0.611 | 0.286 | - | 0.304 |
| N4 (1) | 0.534 | 0.369 | 0.419 | 0.475 | 0.685 | 0.436 | 0.560 | - | 0.536 |
| N1-4 Avg | 0.717 | 0.580 | 0.587 | 0.742 | 0.659 | 0.682 | 0.394 | 1.0 | 0.502 |
| N1-N4 Amended Avg | - | 0.530 | 0.541 | 0.574 | 0.577 | 0.486 | 0.560 | - | 0.588 |

*Table 6. DTC P(success) when r(odd) for r(period multiples) where red indicates only 1 occurrence so is considered an excluded outlier or a dash which indicates no occurrence in that band.*

On analysis of table 6, there does not appear to be a noticeable improvement in the probability of success of DTC for any individual period multiple. The N1-N4 average does not accurately represent the dataset because of a skew where only one select multiple of r occurred and also does not take into consideration the quantity of select multiples of r versus their success probability for each order.

An amended average was calculated for each band by multiplying each probability by its number of occurrences, excluding where only one select multiple of r occurred as outliers. For example, N1-N4 average when r is a multiple of 7 was 0.742 (mean calculated from the total N1 to N4 probability divided by 4). The actual number of N that were successful when r is a multiple of 7 were N1 (1), N2 (7), N3 (3), N4 (14) with a total of 25 numbers. N1 was excluded as an outlier as only one N was successful. The amended average of N1-N4 was then calculated as the total of each band probability multiplied by its number of occurrences divided by 24.

Once these adjustments were made, the N1-N4 amended average for DTC success for the individual period multiples is similar from 0.486 to 0.588 resulting in no particular benefit for any individual multiple of r.

## 3.3. Dong's algorithm

### 3.3.1. Design of Dong's algorithm

From the literature review of Dong's research, his conclusions were that by modifying the standard SFA algorithm, the restriction on the period of the function when r is odd and a multiple of 3 can be relaxed to increase the success of finding nontrivial factors of any N, resulting in the use of some of the odd r periods that would otherwise

have been discarded when using standard SFA.

Dong's generalised formula:

$$\left(a^{\frac{r}{3}}\right)^3 - 1 = 0 \; mod \; N \tag{12}$$

$$\left(a^{\frac{r}{3}} - 1\right)\left(a^{\frac{2r}{3}} + a^{\frac{r}{3}} + 1\right) = 0 \; mod \; N \tag{13}$$

Dong table of results in his research paper only showed the probability of a suitable r period for his algorithm to occur (i.e. an odd period that is also a multiple of 3) for a limited number of N values.

I wrote a python program to implement Dong's algorithm specifically for when r is an odd period and multiple of 3. The probability of success was calculated for my selected N bands and analysed in comparison to DTC.

### 3.3.2. Analysis & comparison of Dong's algorithm and DTC

| N order (a%) | P(odd) | Prob(odd & multiple of 3) | DTC Prob(success) | Dong Prob(success) |
|---|---|---|---|---|
| N1(100) | 0.132 | 0.079 | 0.856 | 0.944 |
| N2(100) | 0.149 | 0.096 | 0.613 | 0.890 |
| N3(10) | 0.129 | 0.084 | 0.482 | 0.904 |
| N4(1) | 0.149 | 0.086 | 0.369 | 0.905 |

*Table 7. Probability of success comparing DTC and Dong's when r is odd and a multiple of 3.*

The data in table 7 shows that any probability that r is odd, the probability that it is also a multiple of 3 is approximately 50% of any odd r across N1 to N4 data. The probability of success for DTC versus Dong shows that Dong's algorithm was more successful than DTC for all N. Whilst DTC is successful for smaller order of N, there is a significant decline as N order increases whereas Dong's modified algorithm remains significantly high as can been better seen from figure 5 below.

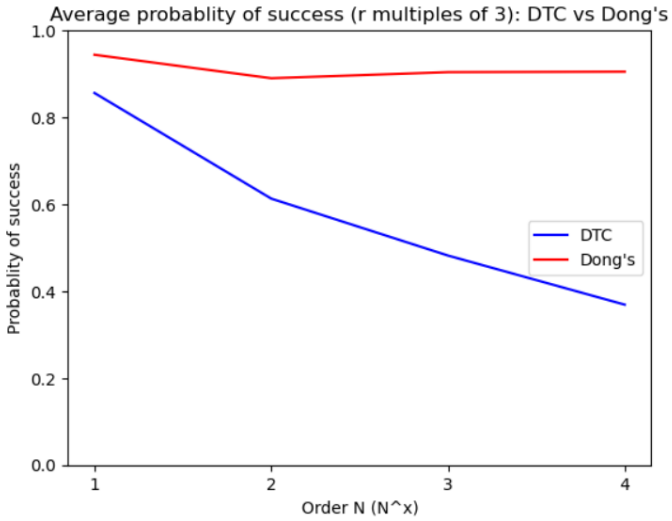



*Figure 5. Average probability of success comparison for DTC vs Dong when r is odd and a multiple of 3.*

The advantage of Dong's modified algorithm over DTC for all orders of N led to further development of my research where I built upon the method to create my own DTC modified algorithm, OptDTC, which will be discussed in the next section.

## 3.4. OptDTC algorithm

### 3.4.1. Design of OptDTC

My hypothesis is that Dong's algorithm has an improved probability of success over DTC because it has been specifically modified to divide the r period by 3 for multiples of 3 and so this could be successful if the r period is divided by a prime factor multiple of r.

I modified the DTC algorithm to represent this for alternative multiples of different prime factors.

Example when r multiple of 5:

$$N = 43, a = 3, r = 15$$

$$(a^{\frac{r}{5}})^5 - 1 \equiv 0 mod N \tag{14}$$

$$= (3^{\frac{15}{5}} - 1)(3^{\frac{4\times15}{5}} + 3^{\frac{3\times15}{5}} + 3^{\frac{2\times15}{5}} + 3^{\frac{1\times15}{5}} + 1)$$

$$= (26)(551881)$$

So we can get

$$P1 = \gcd(26,43) = 26$$
$$P2 = \gcd(551881,43) = 19$$

It can be seen from the above derivation example that P2 can find a

factor for N using this multiple of 5 formula.

The same modification was attempted for the period multiples to create probability of success for select multiples of r (see table 8).

| N order (a%) | OptDTC (Success) | r(3) | r(5) | r(7) | r(11) | r(13) | r(17) | r(19) | r(23) |
|---|---|---|---|---|---|---|---|---|---|
| N1 (100) | 0.944 | 0.944 | 1.0 | 1.0 | 1.0 | - | - | - | - |
| N2 (100) | 0.981 | 0.890 | 0.936 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| N3 (10) | 0.971 | 0.904 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | - | 1.0 |
| N4 (1) | 0.999 | 0.905 | 0.997 | 1.0 | 0.999 | 1.0 | 1.0 | - | 1.0 |
| N1-4 Avg | 0.974 | 0.974 | 0.983 | 1.0 | 0.999 | 1.0 | 1.0 | - | 1.0 |
| N1-N4 Amended Avg | - | 0.979 | 0.979 | 1.0 | 0.990 | 1.0 | 1.0 | - | 1.0 |

*Table 8. Probability of success for OptDTC when r(odd) & multiple of r(period multiples) where red indicated only one occurrence for the multiple and a dash indicates no occurrence for the multiple.*

To note, the probability of success r(3) column represents Dong's algorithm.

On analysis of table 8 results, it can be seen that by modifying DTC exponent to divide r by a prime factor multiple of r, in a similar way to SFA divides an even period r by 2 and Dong's divides an odd period r multiple of 3 by 3, the probability of success for all period multiples is extremely high in the range of 0.979 to 1.0 in the bands N1 to N4. There is not the same trend in decline of success as the magnitude increases as can be seen with DTC period multiples (see table 6).

A generalised formula for this optimisation of DTC can be specified as follows.

$$\left(a^{\frac{r}{x}}\right)^{x} - 1 \equiv 0 \; mod \; N \qquad (15)$$

Where x is a prime multiple of r

$$\left(a^{\frac{r}{x}} - 1\right)\left(a^{\frac{(x-1)r}{x}} + a^{\frac{(x-2)r}{x}} + \cdots + a^{\frac{1\times r}{x}} + 1\right) = 0 \; mod \; N \qquad (16)$$

This generalised formula of OptDTC becomes equivalent to SFA for periods that are a multiple of 2 and equivalent to Dong's for a period that is odd and a multiple of 3 but now extends to apply for all multiples of x/r numbers.

The OptDTC algorithm provides an additional benefit in its post-processing application when a period of r has multiple prime factors which gives additional opportunity for the other prime factors to

produce nontrivial factors of N when previous prime factors failed, thus giving one call to the period finding function additional opportunities for success.

An example is the case when N is 469, *a* is 39 and the r order finding for N and *a* calculates to 33. Prime factors of r that can be used are 3 and 11.

$$N = 469, a = 39, r = 33$$

When prime factor is 3.

$$(39^{\frac{33}{3}})^3 - 1 \equiv 0 mod N \quad (17)$$

$$= (39^{\frac{33}{3}} - 1)(39^{\frac{2\times33}{3}} + 39^{\frac{1\times33}{3}} + 1)$$

$$= (317475837322472438)(1007909072836049843833771972370811 61)$$

So we can get

$$P1 = \gcd(317475837322472438,469) = 162$$
$$P2 = \gcd(1007909072836049843833771972370811 61,469) = 0$$

So P1 and P2 fail to find nontrivial factors.

When prime factor is 11.

$$\left(39^{\frac{33}{11}} - 1\right)\left(39^{\frac{10\times33}{11}} + 39^{\frac{9\times33}{11}} + \cdots + 39^{\frac{1\times33}{11}} + 1\right) = 0 \; mod \; N$$

$$= (59318)(5394429630863174482359597647023837703910311034 01)$$

So we can get

$$P1 = \gcd(59318,469) = 224$$
$$P2 = \gcd(5394429630863174482359597647023837703910311034 01,469) = 67$$

So P2 finds a prime factor of N as 67 and so succeeds to find a nontrivial factor for a prime factor multiple of 11 of r where prime factor of 3 failed.

### 3.4.2. Analysis & comparison of OptDTC, Dong's and DTC

| N order(a%) | DTC Prob (success) | Dong Prob (success) | OptDTC Prob (success) |
|---|---|---|---|
| N1(100) | 0.856 | 0.944 | 0.944 |
| N2(100) | 0.613 | 0.890 | 0.981 |
| N3(10) | 0.482 | 0.904 | 0.971 |
| N4(1) | 0.369 | 0.905 | 0.999 |

*Table 9. Average probability of success when r(odd & multiple of 3) for DTC, Dong and OptDTC*

To enable a comparison, the probability of success for OptDTC, Dong's and DTC was analysed when r was odd and a multiple of 3. It can be seen from table 9 above that the highest level of success is achieved through the use of OptDTC with a probability of success of 0.999 which has a significantly improved probability of success over DTC and Dong's. The increased success of OptDTC can be attributed to the ability of OptDTC to provide additional algorithm attempts when r has other prime factors which lead to success when Dong's fails. Figure 6 shows the comparative success of DTC, Dong and OptDTC when r is odd and a multiple of 3. It should be noted that a multiple of 3 represents approximately 50% of the odd orders available and so Dong's algorithm probability of success when considering all available odd order of r would be approximately half of the values in table 9.

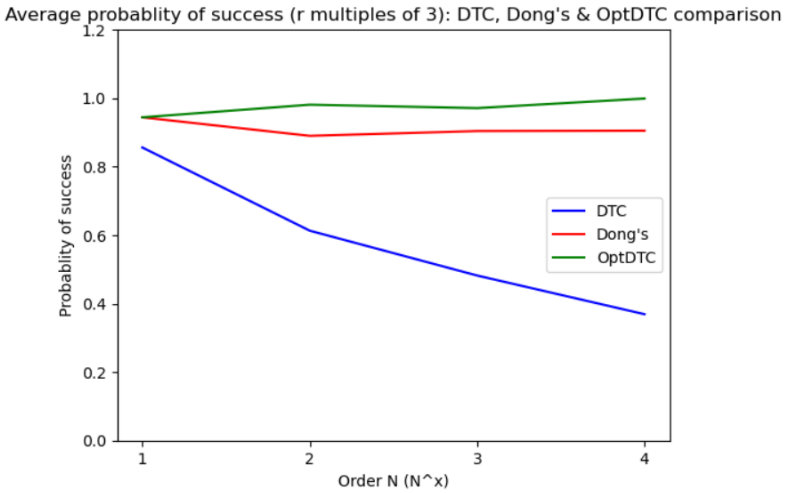



*Figure 6. Average probability of success comparison for DTC, Dong and OptDTC when r is odd and a multiple of 3.*

### 3.4.3. Analysis & comparison of OptDTC and SFA

From the comparison of OptDTC and Dong's, it was shown that OptDTC had an improved probability of success over Dong's due to additional prime factors of r leading to success. This led to the consideration of applying OptDTC when standard SFA fails for even orders where the order also contains at least one other prime factor. The results in table 10 below show that OptDTC did improve the probability of success over standard SFA.

| N order (a%) | SFA prob(success) | OptDTC prob(success) |
|---|---|---|
| N1(100) | 0.780 | 0.950 |
| N2(100) | 0.763 | 0.985 |
| N3(10) | 0.833 | 0.998 |
| N4(1) | 0.807 | 0.999 |

*Table 10. Probability of success for SFA and OptDTC when r is even.*

From the analysis of table 10 results, it shows an improved probability of success for OptDTC over standard SFA with even orders when r also has other odd prime factors, with the additional performance attributed to the alternative attempts available to be used to calculate nontrivial factors using the additional prime factors of r.

An example case is when N is 9989, *a* is 517 and the order finding for N and *a* calculates to an even r of 1426. Prime factors of r that can be used are 2 and 23. SFA and OptDTC fail for prime factor 2, but OptDTC succeeds in its second attempt with prime factor 23, finding the factors for N as 7 and 1427.

When analysing these results further, OptDTC will only produce a greater probability of success over SFA when the r period is even and contains other odd prime factors, otherwise if the r period only has 2 as a prime factor, the general formula for OptDTC becomes identical to SFA and will result in an identical outcome. Therefore, orders that are only the product of the formula $2^x$, where x is some multiple > 1, will not produce any increased probability of success for OptDTC as opposed to SFA. To see a probability of failure comparison between SFA and OptDTC with only periods that are even and also contain an odd prime factor, see table 11 below.

| N order (a%) | SFA prob(failure) | OptDTC prob(failure) |
|---|---|---|
| N1(100) | 0.245 | 0.005 |
| N2(100) | 0.237 | 0.007 |
| N3(10) | 0.167 | 0.001 |
| N4(1) | 0.193 | 0.000 |

*Table 11. Probability of failure for SFA when r is even and has other odd prime factors & the probability of failure for OptDTC when both SFA fails and r is even and has other odd prime factors.*

Note that the probability of failure produced for OptDTC in this table is tied to the failure of SFA, i.e. when SFA failed to find nontrivial factors for any N and the order r calculated was even and also had other odd prime factors, OptDTC was then executed with the same input values to calculate the failure rate. The results in table 11 clearly show that OptDTC has a much smaller failure rate than SFA for even orders when there is an additional prime factor of r that can be used to increase the opportunities to find nontrivial factors.

As shown in the results, the probability that OptDTC also failed where SFA did was extremely low, reaching a 0% probability of failure for band N4, meaning that for any number that SFA failed to find nontrivial factors, OptDTC had enough alternative prime factor attempts so that at least one of these alternatives found the nontrivial factors.

When considering all odd and even order periods available for the data range used in this project, the overall probability of success for OptDTC can be seen in table 12 and for SFA in table 13.

| OptDTC | N1 | N2 | N3 | N4 |
|---|---|---|---|---|
| P(odd) | 0.132 | 0.149 | 0.129 | 0.149 |
| P(success, r odd) | 0.944 | 0.981 | 0.971 | 0.999 |
| % (of all odd & even r) P(success, r odd) | 0.125 | 0.146 | 0.125 | 0.149 |
| P(even) | 0.868 | 0.851 | 0.871 | 0.851 |
| P(success, r even) | 0.950 | 0.985 | 0.998 | 0.999 |
| % (of all odd & even r) P(success, r even) | 0.825 | 0.838 | 0.869 | 0.850 |
| **Overall P(success all r)** | **0.950** | **0.984** | **0.994** | **0.999** |

*Table 12. OptDTC overall probability of success for odd and even r. Values for P(success, r odd) from table 8 and P(success r even) from table 10.*

| SFA | N1 | N2 | N3 | N4 |
|---|---|---|---|---|
| P(odd) | 0.132 | 0.149 | 0.129 | 0.149 |
| P(success, r odd) | - | - | - | - |
| % (of all odd & even r) P(success, r odd) | - | - | - | - |
| P(even) | 0.868 | 0.851 | 0.871 | 0.851 |
| P(success, r even) | 0.780 | 0.763 | 0.833 | 0.807 |
| % (of all odd & even r) P(success, r even) | 0.663 | 0.649 | 0.725 | 0.686 |
| **Overall P(success all r)** | **0.663** | **0.649** | **0.725** | **0.686** |

*Table 13. SFA overall probability of success for even r (odd periods discarded). Values for P(success, r even) from table 10.*

Comparing the results from table 12 and 13, it can be clearly seen that OptDTC provides a significant improvement in the success probability when compared to standard SFA for all available odd and even period r values. The improvement is high across all magnitude numbers in the data (see figure 7). The improvement is 43.3% over SFA for N1, 51.6% for N2, 37.1% for N3 and 45.6% for N4 with an overall average improvement of OptDTC over SFA of 44.4% across the whole data set.

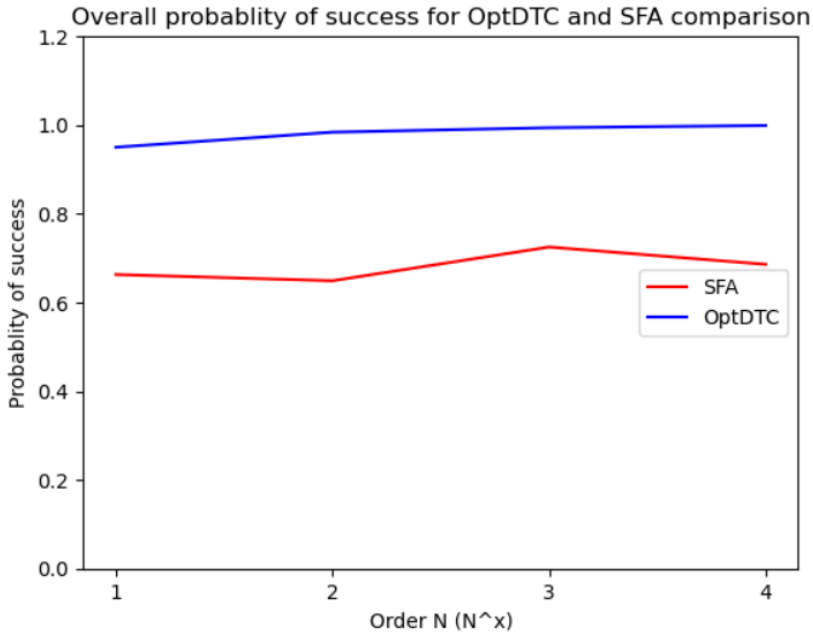



*Figure 7. Overall probability of success for OptDTC and SFA for all orders*

A flowchart to represent the new process using the OptDTC generalised algorithm is shown in figure 8 below. The flowchart modifies the current SFA and introduces a new step into the traditional step-by-step method for the generalised OptDTC algorithm. The additional step occurs after the quantum step 2 to calculate r, where the additional step “2b” then creates a prime factorisation list for the calculated r. The list will then be used to provide the alternative x values in the generalised formula (see section 3.4.1.). The list is used incrementally until either this new prime factor of r successfully finds the nontrivial factors of N or the end of the list is reached, at which point it is deemed to have failed and returns to step 1 to begin the process again using a new random guess.

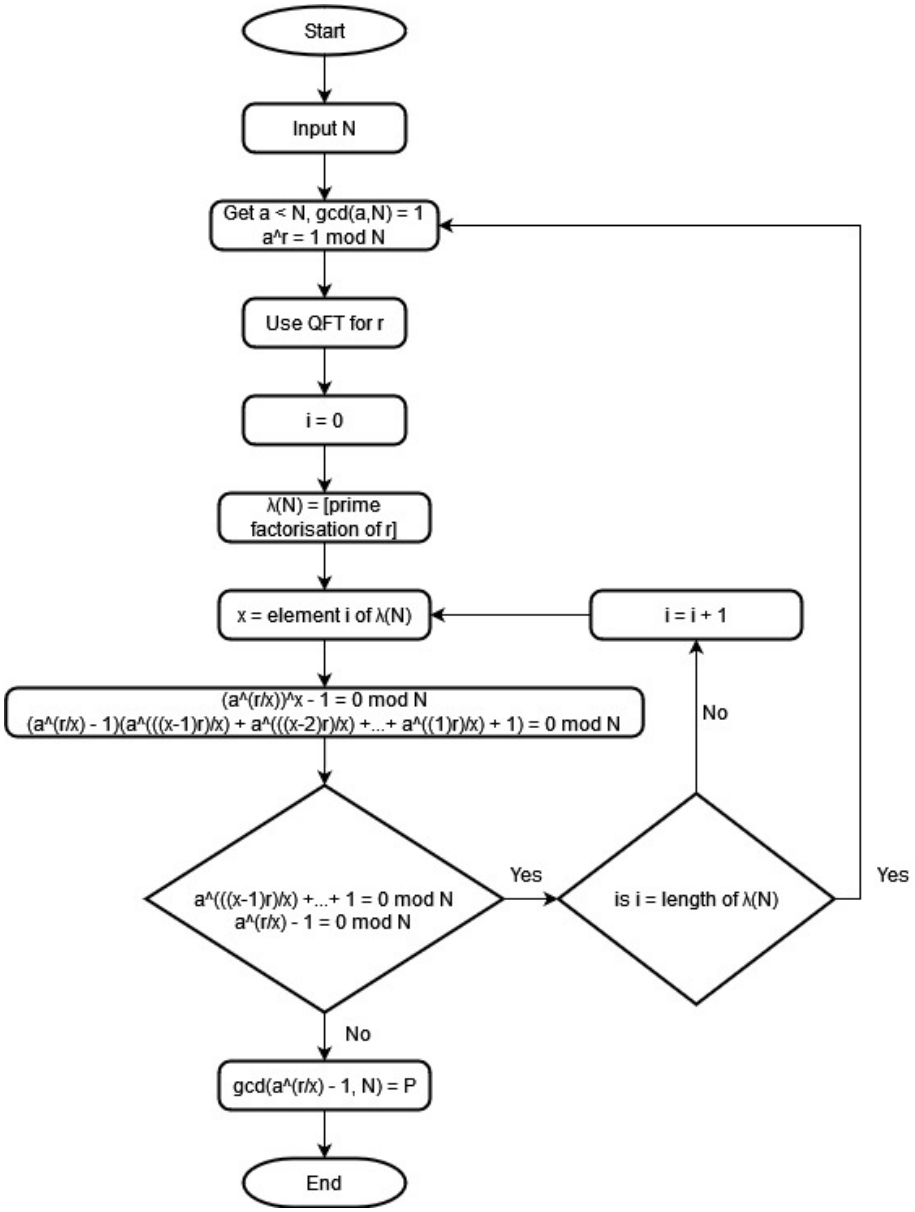



*Figure 8.Flowchart of the proposed OptDTC generalised algorithm process.*

## 4. EVALUATION

When evaluating the effectiveness of DTC, it can be stated that DTC does provide some effectiveness in its usage of odd orders to find nontrivial factors of N but the probability of success trends downwards significantly as the magnitude of N increases. As order N increased for the 4 bands evaluated, the probability of success significantly reduced from approximately 92% for N1 to 53% for N4. Assuming the trend shown in table 5 continues as a steady decline then for numbers of a much higher magnitude, for example those used in cryptography, it would be expected to become a very small probability of success, limiting the effectiveness of using DTC as the magnitude increases. Experimentally, the probability of an odd order occurring across the 4 bands was calculated as approximately 14% of all orders and so, whilst DTC had some success, it may only result in a very small overall increase in the success rate for finding nontrivial factors due to its applicability to a small quantity of numbers. This small increase in the probability of success can still be argued as providing an overall increase in the success of factor finding for Shor's algorithm as opposed to the current standard of discarding all odd orders and repeating the computationally expensive quantum period finding step until a suitable even order r is achieved, however, the trend downwards as N magnitude increases leads to the conclusion that DTC may not provide sufficient benefit when used for large numbers as used in cryptography and the recommendation would be to utilise the most successful algorithm in this project, which is identified as OptDTC.

Dong's modified algorithm produced an improved probability of success over DTC in all bands, but only for a limited range of r when r was odd and a multiple of 3, with approximately 91% success with N4 order numbers compared to the DTC algorithm's success of 37% at N4. Whilst this is a much larger improvement compared to DTC, multiples of 3 r values only encapsulate approximately 53% of the total number of available odd value orders, meaning that the remaining values, representing approximately 47% of all odd orders are unsuitable for Dong's algorithm and so must still discarded. With consideration to Dong's algorithm, it can be concluded that even though approximately half of all available odd orders are still discarded, the high level of success when used with suitable r values provides a significant improvement over the standard SFA, however, it was found to be less effective overall than OptDTC.

On evaluation of all the results, it is clearly shown that the probability of success is much greater for OptDTC for all bands of N than any other algorithm considered. This is due to the versatile adaption of the generalised algorithm to be able to use any prime factor of r and the possibility that multiple prime factors of r can be used to attempt

to calculate nontrivial factors. This reached an approximate 99% probability of success with OptDTC for all odd orders in the N4 band, while DTC success was approximately 53% and Dong was approximately 52% when adjusted as a percentage of all odd order numbers available (multiples of 3 represent approximately 58% of all odd r with Dong's success of 0.905 at N4). Further research using OptDTC showed not only a high probability of success for odd orders compared to DTC and Dong but also an increased probability of success for each band with even orders in comparison to standard SFA, outperforming it in each band as well. OptDTC reached an approximate 99% probability of success for all even orders in the N4 band while SFA was approximately 81%. The project results show that the expensive order finding step no longer requires discarding when an odd order r occurs and can actually provide an incredibly high probability of success for any N in a single call to the order finding routine when using the OptDTC algorithm. In addition, OptDTC also was able to increase the probability of success when standard SFA failed using an even order due to its ability to attempt alternative prime factor calculations, again resulting in an increase in the probability of success in a single call to the order finding routine. With consideration to the whole dataset for all periods of r, when considering all odd and even values of r in the dataset, OptDTC provided approximately 46% increased success at N4 over SFA ((OptDTC 0.999-SFA 0.686)/SFA 0.686). This remains similar to the increased success over SFA for N1 (43%), N2 (52%), N3 (37%). If this trend continues for larger magnitudes of N then it can be assumed that OptDTC will continue to provide the highest probability of success in finding nontrivial factors.

# 5. CONCLUSION

## 5.1. Project conclusion

This research has shown that OptDTC provides the greatest success when compared to all other algorithms explored. The initial project aim was to identify whether odd orders could be utilised to try and find nontrivial factors and use the computationally expensive period finding step that would otherwise have been discarded by standard SFA. The OptDTC algorithm succeeded in this aim by providing a high probability of success of approximately 98% using odd order r values. With the occurrence of odd order r representing approximately 14% of the dataset, OptDTC provides a significant advantage over SFA by utilising these odd orders. Whilst Dong's algorithm obtained a high probability of success for odd order multiples of 3, these only represent approximately 53% of odd numbers and so Dong's probability of success is reduced by approximately half when consideration is also given to the discarded numbers.

The project exceeded the initial aim by creating a generalised algorithm that also increased the probability of success over standard SFA when using even order r values. This enables OptDTC to provide the highest probability of success for all orders of r over all other algorithms researched while efficiently using one call to the quantum finding subroutine.

In conclusion, I propose the OptDTC algorithm replaces the standard SFA algorithm for factor finding (see figure 8 for the flowchart of the proposed OptDTC algorithm).

## 5.2. Implications & further research

This project provides optimistic evidence for an increased probability of success in finding nontrivial factors for all orders using OptDTC. It is limited in its application currently to smaller numbers due to the technological limitations of quantum computers and the time complexity of classical computers. Further research is required using computers with a significantly increased processing power to identify whether the trends are similar as the magnitude increase beyond the current project data limit of 5 digit numbers 10^4(N4). If the trends continue, the implementation of OptDTC could have significant implications in the future for cryptographic methods based on the difficulty of factoring a large composite number. Whilst this currently is not an achievable aim due to the infancy of quantum computing and its inability to operate with qubits sufficiently large enough to process large numbers, the continued developments in quantum computing may one day soon overcome these limitations. It may be

more of a case of when, rather than if, the factorisation of large composites can occur and so the long-term implications need to shift research focus away from scaling current cryptographic methods (e.g. 128 bit to 256 bit to 512 bit and beyond) and instead to research into creating new cryptographic methods that are not based on the difficulty of factorising a large composite. Any new cryptographic schemes developed would need to be tested to be robust from both classical and quantum computing cryptography attacks.

Whilst the recommendation made in this project is to develop new cryptographic schemes to lead to better protection of future data, the possibility in the near future of factorising large composite numbers continues to be a significant ongoing threat for many organisations. There is evidence that many organisations, including national governments, have already experienced data breaches and compromises where sensitive data has been harvested (known as “harvest now, decrypt later”) and while the data currently cannot be decrypted, the threat still remains that once quantum computing progresses far enough to overcome current limitations and can find the multiplicative order r for large numbers and if OptDTC or similar algorithms scale up to larger magnitude numbers, then current day encryption will be cracked and the historic stolen data from companies can be decrypted with serious implications and threads as a result of this.

# 6. APPENDIX

## 6.1. Programming code and raw data

The Python code used to produce the data and the raw data is accessible electronically in a ZIP file uploaded with this project.

## 6.2. N values used for all data

N1(2 Digits) = [21,33,35,39,51,55,57,65,69,77,85,87,91,93,95]

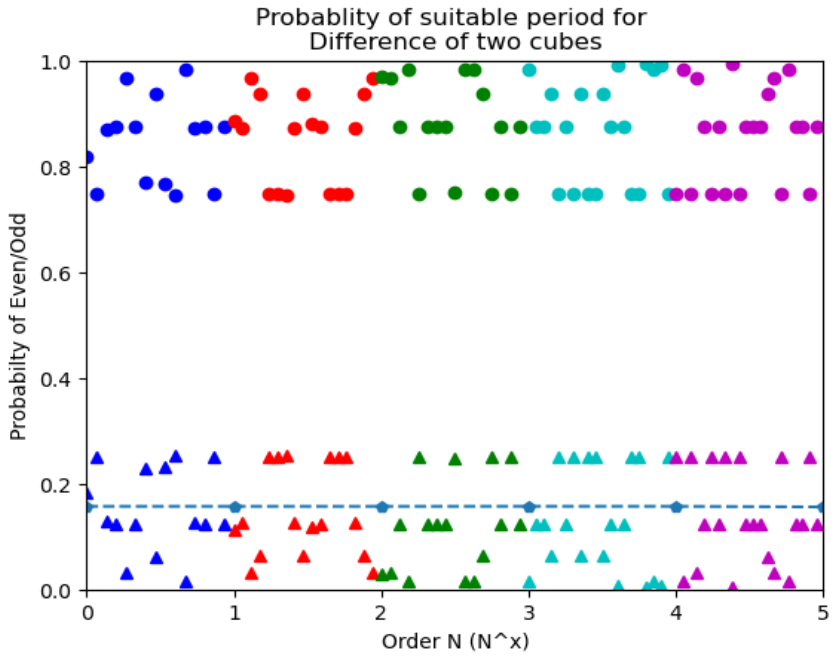
N2(3 Digits) = [111,115,119,123,129,133,141,143,145,155,451,453, 469,471,473,481,485, 489,493,497,955,959,961,965,973,979,985, 989,993,995]

N3(4 Digits) = [1003,1011,1027,1037,1041,1043,1047,1055,1057, 1059,5561,5567,5579,5583,5585,5587,5597,5599,5601,5603,9977, 9979,9983,9985, 9987,9989,9991,9993,9995, 9997]

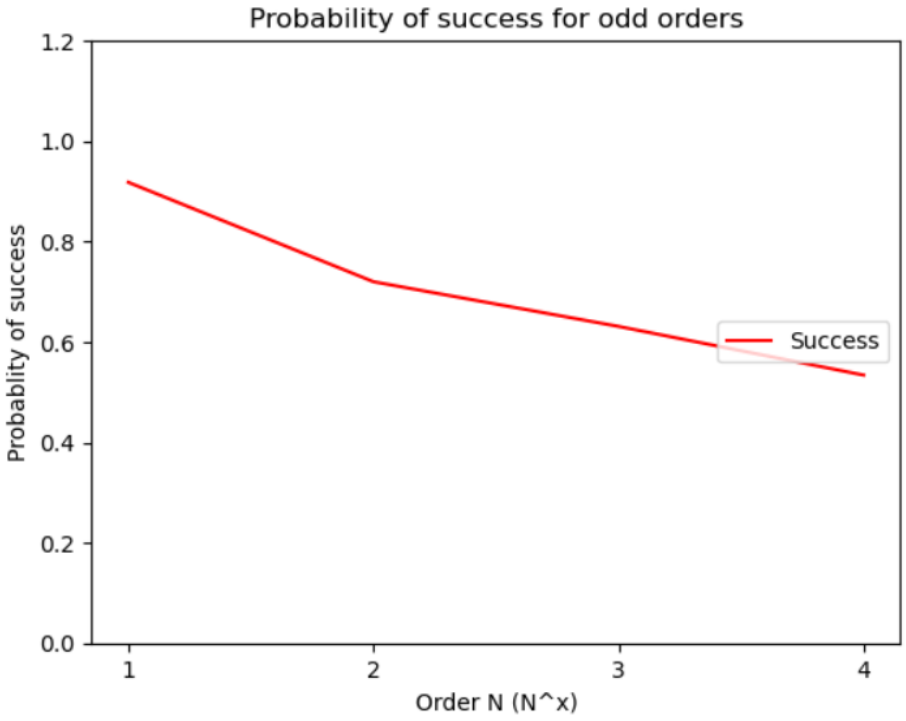
N4(5 Digits) = [10001,10003,10015,10019,10021,10027,10029, 10031,10033,10041,53121,53123,53135,53137,53139,53141,53143, 53153,53167,99955,99959,99965,99967,99973,99977,99983,99987, 99993]

## 6.3. Figure showing individual probability of even vs odd

Individual probability of even vs odd orders for each N value (Triangles represent odd orders, circles represent even orders)



## 6.4. Probability of success for DTC for odd orders



## 7. BIBLIOGRAPHY

[1] Google Quantum AI. Suppressing quantum errors by scaling a surface code logical qubit. Nature 614, 676–681 (2023). https://doi.org/10.1038/s41586-022-05434-1

[2] Johnston, Anna M.. “Shor's Algorithm and Factoring: Don't Throw Away the Odd Orders.” IACR Cryptol. ePrint Arch. 2017 (2017): 83.

[3] Lawson, T. Odd orders in Shor’s factoring algorithm. Quantum Inf Process 14, 831–838 (2015). https://doi.org/10.1007/s11128-014-0910-z

[4] Improving the success rate of quantum algorithm attacking RSA encryption system (2022) Home. Available at: https://www.researchsquare.com/article/rs-1501203/v1

[5] Lomonaco, J. (2000) Shor's quantum factoring algorithm, arXiv.org. Available at: https://arxiv.org/abs/quant-ph/0010034

[6] S. L. Braunstein, “Quantum Computation Lecture 10”, University of York, 2019

[7] Ekerå, M. On completely factoring any integer efficiently in a single run of an order-finding algorithm. Quantum Inf Process 20, 205 (2021). https://doi.org/10.1007/s11128-021-03069-1

[8] Grosshans, Frédéric & Lawson, Thomas & Morain, François & Smith, Benjamin. (2015). Factoring Safe Semiprimes with a Single Quantum Query.