

Research Report: RSA Encryption Fixed Point Analysis

Aron Schwartz | Portland State University | Fall 2019

Introduction

The mathematical foundations of RSA encryption were first proposed in 1977 in a paper by Ron Rivest, Adi Shamir, and Leonard Adleman^[1]. Shortly after this initial paper was released, another paper was published describing a peculiar flaw in the RSA algorithm—the so called occurrence of “Fixed Points”^[2]. In this paper by Blakely and Borosh, a phenomenon is described in which RSA encryption can sometimes fail to “hide” an encrypted integer for certain choices of parameters. In other words, the mathematical result of encrypting an integer (which should by all means be different than the original integer) is identical to the original integer...as if encryption did not occur at all. This so called phenomenon of “fixed points” (also referred to as “holes” in this report) has been well known since this paper was released in 1979. However, while the existence of this phenomenon has been well known for some time, methods correlating the relationship between the 7 input parameters (also referred to as a “septuple”) and the resulting likelihood of fixed point occurrence has been lacking. Current methods rely on manually checking for fixed point occurrence by encrypting a set of integers and observing the results...however this method is computationally expensive and inefficient, particularly if exhaustive hole analysis is desired (“exhaustive” referring to an explicit check for fixed point occurrence across the set of encryptable integers, i.e. 0 to $N-1$ where $N = P*Q$).

The motivations of the research described in this report are rooted in the search for a computationally cheap and mathematically reliable check for fixed point occurrence. This is done by analyzing the relationship of initial parameter choice compared against the resulting “transparency” for a given set of encryption parameters, whilst utilizing python scripting as a data collection and analysis tool. A detailed overview of the work performed, conclusions reached, and questions raised can be found in the following sections.

A link to the repository containing all code and data that is referenced in this report can be found at the link below. The .README file of the repository contains a brief description of all data folders and python files.

Link to Repository: https://github.com/aronjschwartz/RSA_Encryption.git

Phase 1—RSA Algorithm Study and Python Implementation

The first phase of research involved a general overview and study of the RSA Encryption algorithm, followed by an implementation of the algorithm in Python. Initially, this process was qualitative in nature beginning with an analysis of the Master's Thesis by Behnaz^[3]: "Finding Cases of Ciphertext equal to Plaintext in the RSA Algorithm". A brief overview of the required parameters to perform RSA encryption and their role in the RSA algorithm can be seen below:

(NOTE: This report does not cover the mathematics behind the RSA algorithm in detail. For a concise mathematical overview, see Behnaz Thesis)

- **P and Q:** Two initially chosen prime numbers which govern other parameter choices
- **N:** Product of P and Q
- **Totient (Also called Phi):** Product of (P-1)*(Q-1)
- **E:** Encryption exponent, part of the public key (E, N)
- **D, K:** Parameters related to decryption. Not needed for fixed point analysis, since one only needs to encrypt to check for existence of a fixed point/hole.

Phase 1 Goal: Encrypt a string to ciphertext, and de-encrypt it back to plaintext, thus demonstrating proper setup of the RSA mathematics and python code structure.

The design decision to implement the algorithm using Python's OOP (Object Oriented Programming) framework was chosen for code readability and scalability. All functionality needed to encrypt and decrypt integers is comprised in the file **encrypt.py**. A short snippet of this code can be seen below:

```
class encryption_set():
    #Need the original primes to initialize the object
    def __init__(self, p, q, custom_e=None, custom_d=None, custom_k=None):
        #Assign p, q, n, and totient
        self.p = p
        self.q = q
        self.n = p*q
        self.totient = self.generate_totient(p, q)

        self.valid_e_list = []
        #Debug mode default to false
        self.debug = False

        #Make randomly valid keys if none are passed in, otherwise set the the specified value
        if (custom_e == None):
            self.e = self.generate_random_valid_e()
        else:
            self.e = custom_e
        if (custom_d == None):
            self.d = self.generate_random_valid_d()
        else:
            self.d = custom_d

        #Set the k if a custom one is passed, otherwise it will be generated when we create a valid d
        if custom_k != None:
            self.k = custom_k
```

Figure 1: Encrypt.py implements an OOP approach to RSA algorithm

As shown in the screenshot, initialization of the object 'encryption_set' accepts at a minimum the initial P and Q values. If no other parameters are provided, the code will find valid E, D, and K values to complete the set, as well as deriving N and the Totient from the P and Q values.

However, more pragmatically it allows the user to choose all parameters of the set if they wish upon object creation. This flexibility allows creation of specific “encryption objects” to exist as unique python objects with their own unique initialization parameters, methods, handles, etc. More importantly, this flexibility allows easy creation and distinction of an arbitrary numbers of encryption objects making large data analysis easier.

Once the code was implemented, various septuples were created and tested against basic strings. A screenshot demonstrating the output of **encryption_test.py** can be seen below:

```
C:\Users\aronj\Grad_School_Classwork\Fall_2019\RSA_Encryption_Research\RSA_Encryption>python encryption_test.py
P: 47
Q: 59
N: 2773
T: 2668
E: 1543
D: 2151
K: 1244

Plain text: But soft, what light from yonder window breaks? It is the East...and Juliet is the Sun
Cipher text: >°bB~BpbeBİğbBByçbBj0B[B~BTeBÖBİyTeBİBÖBÖğt~BbBbBç~BbBbBç~B.....ğTeBÖBÖğBç~BbBbBç~B
Decrypted cipher text: But soft, what light from yonder window breaks? It is the East...and Juliet is the Sun
C:\Users\aronj\Grad_School_Classwork\Fall_2019\RSA_Encryption_Research\RSA_Encryption>
```

Figure 2: Output of encryption_test.py demonstrating algorithm functionality on a simple string

Phase 2: Focus shift to “holes”- Recreating the Tiny Key Encryption Table to verify hole detection algorithm

After functionality of the RSA algorithm was verified, the focus was shifted toward fixed points and the ability to detect them. Ensuring proper “hole searching” functionality on a pre-known solution was critical to expanding the work to further phases, thus the motivation for recreating the chart from the Behnaz thesis.

Phase 2 Goal: Recreate the “Tiny Key Encryption Table” from Behnaz thesis thus demonstrating proper functionality of the “hole searcher” algorithm

Table 3.1 Tiny Key Encryption Table

$e = 11$		$n = 15$		$d = 3$
cleartext		ciphertext		decrypted text
m	$m^2 \bmod(n)$	c	$c^2 \bmod(n)$	$m \times g$
2	4	8	4	2
3	9	12	9	3
4	1	4	1	4
5	10	5	10	5
6	6	6	6	6
7	4	13	4	7
8	4	2	4	8
9	6	9	6	9
10	10	10	10	10
11	1	11	1	11
12	9	3	9	12
13	4	7	4	13
14	1	14	1	14

Figure 3: The “Tiny Key Encryption Table” showing hole occurrences highlighted in orange

The chart shown in the screenshot demonstrates the concept of holes visually. With a given choice of parameters, all values from 2 to $N-1$ are encrypted (in this chart, 0 and 1 are skipped as they are trivial cases due to the modular math involved^[3]). The rows highlighted in orange represent a “hole”, as evidenced by the fact that the cipher text is identical to the plaintext. In summary, this chart was recreated with simple print statements validating the ability to calculate and analyze all holes for a given set of initial parameters.

Phase 3: Large scale data collection mapping input parameter choice to relative hole occurrence

Once the ability to find holes via code was confirmed, open ended data collection was initiated without regards to a particular pattern being searched for. The goal was to generate statistically significant numbers of septuples and analyze the resulting occurrence of holes, with the hopes of an explorable relationship emerging.

Phase 3 Goal: Systematically generate hundreds of parameter combinations searching for discernible or explorable relationships between input parameter choice and the resulting relative occurrence of fixed points

To accomplish this, code was written to generate various septuple combinations and encrypt all possible values (0 to $N-1$) to exhaustively check for holes. Since this process is computationally expensive for large values of N , lower initial primes (P and Q) were chosen for easier computation. An overview of how the input parameters were chosen and generated can be seen in the following sections.

Choosing P and Q

As mentioned, the higher the value of N , the more computation time is needed to analyze all values from 0 to $N-1$. To accommodate this fact, P and Q were chosen to be all possible combinations of prime numbers in the first 100 primes, **approximately 25,000 different combinations**. This was accomplished by downloading a text file of the first 1,000,000 primes and doing some simple parsing, as well as some additional logic to prevent repeated combinations of P and Q . A screenshot of this code illustrating the logic flow can be seen below.

```

p_list = prime_list[start_choice-1:end_choice-1]
q_list = prime_list[start_choice-1:end_choice-1]
print("Start val: ", prime_list[start_choice-1], " End val: ", prime_list[end_choice -1])
#e_list = [3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89]
pub_keys = [3, 5, 7, 17, 257, 65537]
#Loop through all p/q combinations trying each e-value for all combinations
start_time = time.time()
for p_val in p_list:
    for q_val in q_list:
        for e_val in pub_keys:
            #Ignore repeat cases and cases where p==q
            if q_val > p_val:
                try:
                    #Make a temp object for the current septuple
                    temp_object = encrypt.encryption_set(p=p_val, q=q_val, custom_e=e_val)
                    #temp_object.enable_debug_mode()

                    sept = temp_object.get_septuple();

                    #Analyze the holes in the septuple
                    holes_num = search_septuple(temp_object)

```

Figure 4: Code allows choice of starting and ending index in first million primes

As shown, the code utilizes for-loops to generate all possible prime combinations, with the lowest prime being 3 and the largest prime being 541 (the 100th prime). For example, the first few combinations of P and Q and 3/5, 3/7, 3/11...ending with 521/523, 523/541.

Choosing E (and by extension, the public keys (E, N)):

The choice of E values for this analysis was chosen to represent the six most commonly recommend values of E following independent research on stack overflow and encryption forums—specifically, the values {3, 5, 17, 257, 65537}. The public recommendation of these E-values is rooted in the fact that these values only have two hot bits, thus making modular exponentiation faster as well as easing the computational burden on the end-user (since huge exponential calculations is often required). The following online forum posts help illuminate the common use of these keys in the security community:

<https://security.stackexchange.com/questions/2335/should-rsa-public-exponent-be-only-in-3-5-17-257-or-65537-due-to-security-c>

https://www.di-mgt.com.au/rsa_alg.html

For the purposes of this research, these E values were used to provide consistency and alignment with commonly used values in the real world. All P/Q combinations as described above were combined with each of these values of E **to ultimately generate 24256 unique encryption sets.**

Phase 4: Analysis of obtained data

After large amounts of data were generated as described in the previous section, the data was analyzed and reviewed.

Phase 4 Goal: Determine any discernible, repeatable, or describable relationship between the choice of input parameters (P, Q, N, Totient, E) chosen to create the encryption object, and the resulting occurrence of fixed points

For this analysis, the term “transparency” is often used to describe the relative strength of a septuple. This refers to the degree by which an encryption set has (or does not have) holes. For example, a set that has 100% transparency means that all possible values that can be encrypted for that set are holes (the worst possible scenario). Likewise, 0% transparency would indicate no holes, although this is not possible in reality due to the existence of the six principal holes^[3].

In summary, 0% *opacity* is impossible, however 100% *transparency* is certainly possible^[2] and was in fact observed in many cases.

Observation 1: The highest transparency septuples are associated with totients that are a power-of-two, or near a power-of-two

Upon analysis, a mathematical commonality among the highest transparency septuples was found in the totient parameter (Totient = $(P-1)*(Q-1)$). The following screenshots and charts further illustrate the peculiar pattern. The screenshot below is taken from the file “primes_1_to_100_holes.csv”, and can be found in its entirety in the provided repository.

p, q, n, Phi, e, k, d	n	totient	e	# holes	Transparency Percentage
[17, 257, 4369, 4096, 257, 241, 3841]	4369	4096	257	4366	99.95
[17, 257, 4369, 4096, 65537, 16, 1]	4369	4096	65537	4366	99.95
[5, 257, 1285, 1024, 257, 193, 769]	1285	1024	257	1282	99.84
[5, 257, 1285, 1024, 65537, 64, 1]	1285	1024	65537	1282	99.84
[3, 257, 771, 512, 257, 129, 257]	771	512	257	768	99.74
[3, 257, 771, 512, 65537, 128, 1]	771	512	65537	768	99.74
[5, 17, 85, 64, 17, 13, 49]	85	64	17	82	97.62
[5, 17, 85, 64, 257, 4, 1]	85	64	257	82	97.62
[5, 17, 85, 64, 65537, 1024, 1]	85	64	65537	82	97.62
[3, 17, 51, 32, 17, 9, 17]	51	32	17	48	96
[3, 17, 51, 32, 257, 8, 1]	51	32	257	48	96
[3, 17, 51, 32, 65537, 2048, 1]	51	32	65537	48	96
[3, 5, 15, 8, 5, 3, 5]	15	8	5	12	85.71
[3, 5, 15, 8, 17, 2, 1]	15	8	17	12	85.71
[3, 5, 15, 8, 257, 32, 1]	15	8	257	12	85.71
[3, 5, 15, 8, 65537, 8192, 1]	15	8	65537	12	85.71
[3, 5, 15, 8, 3, 1, 3]	15	8	3	6	42.86
[7, 257, 1799, 1536, 257, 43, 257]	1799	1536	257	768	42.71
[7, 257, 1799, 1536, 65537, 43734, 1024]	1799	1536	65537	768	42.71
[7, 17, 119, 96, 17, 3, 17]	119	96	17	48	40.68
[7, 17, 119, 96, 257, 174, 65]	119	96	257	48	40.68
[7, 17, 119, 96, 65537, 44374, 65]	119	96	65537	48	40.68
[13, 257, 3341, 3072, 257, 150, 1793]	3341	3072	257	1282	38.38
[3, 257, 3341, 3072, 65537, 21867, 1024]	3341	3072	65537	1282	38.38
[13, 17, 221, 192, 17, 10, 113]	221	192	17	82	37.27
[13, 17, 221, 192, 257, 87, 65]	221	192	257	82	37.27
[13, 17, 221, 192, 65537, 22187, 65]	221	192	65537	82	37.27

Figure 5: The highest transparency septuples had totients equal to a power-of-two

The screenshot shows the highest transparency septuples after sorting the data by transparency value. One can see the commonality of the totients all being a direct power-of-two for the highest transparency sets. Upon this discovery, some charts were generated in an attempt to further understand the pattern. The code was re-spun to output the binary representation of the totient as an additional parameter. Screenshots of this data re-visualization and a representative graph illustrating the trend across the data set can be seen below.

A	B	C	D	E
p, q, n, Phi, e, k, d	Totient	Binary Totient	# Hot Bits	Transparency Percentage
[17, 257, 4369, 4096, 257, 241, 3841]	4096	0b10000000000000	1	99.95
[17, 257, 4369, 4096, 65537, 16, 1]	4096	0b10000000000000	1	99.95
[5, 257, 1285, 1024, 257, 193, 769]	1024	0b100000000000	1	99.84
[5, 257, 1285, 1024, 65537, 64, 1]	1024	0b100000000000	1	99.84
[3, 257, 771, 512, 257, 129, 257]	512	0b10000000000	1	99.74
[3, 257, 771, 512, 65537, 128, 1]	512	0b10000000000	1	99.74
[5, 17, 85, 64, 17, 13, 49]	64	0b10000000	1	97.62
[5, 17, 85, 64, 257, 4, 1]	64	0b10000000	1	97.62
[5, 17, 85, 64, 65537, 1024, 1]	64	0b10000000	1	97.62
[3, 17, 51, 32, 17, 9, 17]	32	0b1000000	1	96
[3, 17, 51, 32, 257, 8, 1]	32	0b1000000	1	96
[3, 17, 51, 32, 65537, 2048, 1]	32	0b1000000	1	96
[3, 5, 15, 8, 5, 3, 5]	8	0b1000	1	85.71
[3, 5, 15, 8, 17, 2, 1]	8	0b1000	1	85.71
[3, 5, 15, 8, 257, 32, 1]	8	0b1000	1	85.71
[3, 5, 15, 8, 65537, 8192, 1]	8	0b1000	1	85.71
[3, 5, 15, 8, 3, 1, 3]	8	0b1000	1	42.86
[5, 17, 85, 64, 5, 1, 13]	64	0b1000000	1	26.19
[3, 17, 51, 32, 5, 2, 13]	32	0b1000000	1	24
[3, 17, 51, 32, 3, 1, 11]	32	0b1000000	1	12

Figure 6: Data from figure 4 re-spun to show bit pattern and totient hot bit quantity

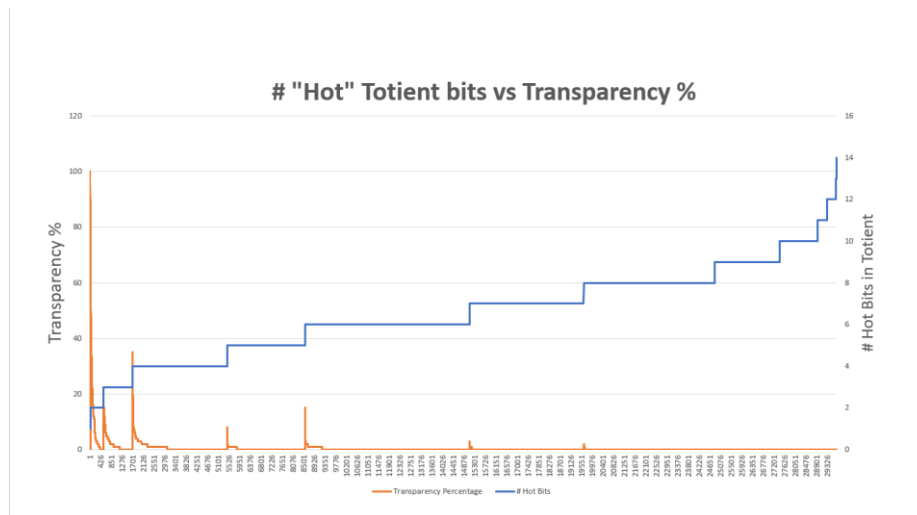


Figure 7: Graph showing relationship between hot bits in the totient and exponentially decreasing transparency

The blue line in the graph demonstrates increasing number of hot bits of the totient vs the relative transparency percentage in orange. As evident by the graph, the peak transparency values occur when the totients are direct powers of two. As the number of hot bits in the totient increases (as by extension, as the totient trends away from a direct power of two), the peak transparency decreases. Each subsequent “bit zone” has a lower and lower peak transparency. By the time we reach about 8-9 hot bits in the totient, transparency percentages are nearly negligible (transparency < 0.1%).

A few specific septuples and their associated data can be seen below to further visualize these findings.

1 Hot Bit Examples:

1. Totient = 512

- Binary = 0010 0000 0000
- Septuple = [3, 257, 771, 512, 257, 129, 257]
- Transparency = 99.74%

2. Totient = 4096

- Binary = 0001 0000 0000 0000
- Septuple = [17, 257, 4369, 4096, 257, 241, 3841]
- Transparency = 99.95%

3. Totient = 64

- Binary = 0100 0000
- Septuple = [5, 17, 85, 64, 65537, 1024, 1]
- Transparency = 97.62%

2 Hot Bit Examples:

1. Totient = 192

- Binary = 1100 0000
- Septuple = [13, 17, 221, 192, 17, 10, 113]
- Transparency = 37.27%

2. Totient = 2560

- Binary = 1010 0000 0000
- Septuple = [11, 257, 2827, 2560, 257, 180, 1793]
- Transparency = 27.18%

3. Totient = 1152

- Binary (0100 1000 0000)
- Septuple = [7, 193, 1351, 1152, 257, 143, 641]
- Transparency = 14.22%

3 hot Bit Examples:

1. Totient = 448

- Binary = 0001 1100 0000
- Septuple = [17, 29, 493, 448, 65537, 37596, 257]
- Transparency = 16.67%

2. Totient = 1792

- Binary = 0111 0000 0000
- Septuple = [17, 113, 1921, 1792, 257, 147, 1025]
- Transparency = 14.90%

3. Totient = 114688

- Binary = 0001 1100 0000 0000 0000
- Septuple = [257, 449, 115393, 114688, 257, 183, 81665]
- Transparency = 14.47%

...

13 Hot Bit Examples:

1. Totient = 131028

- Binary = 0001 1111 1111 1101 0100
- Septuple = [359, 367, 131753, 131028, 17, 15, 115613]
- Transparency < 0.01%

2. Totient = 259956

- Binary = 0011 1111 0111 0111 0100
- Septuple = [499, 523, 260977, 259956, 3, 1, 346608]
- Transparency < 0.01%

14 Hot Bit Examples:

1. Totient = 253692

- Binary = 0011 1101 1110 1111 1100
- Septuple = [487, 523, 254701, 253692, 17, 16, 238769]
- Transparency < 0.01%

2. Totient = 262044

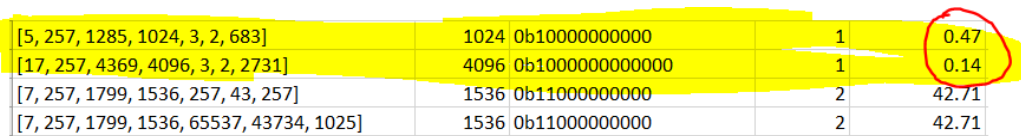
- Binary = (0011 1111 1111 1001 1100)
- Septuple = [503, 523, 263069, 262044, 65537, 34659, 138581]
- Transparency < 0.01%

Questions raised from the totient-pattern observation

Upon the discovery of the demonstrated totient relationship, a series of questions were postulated regarding the nature, repeatability, and significance of the described pattern. Some fundamental questions that arose from this discovery and the progress toward answering those questions can be seen below.

Question 1: Does the “pattern” always hold? In other words, does the hot-bit to totient relationship demonstrated in the previous section always hold?

Answer: No, the pattern is not deterministic and does not hold in all cases. Cases in which power-of-two totients and/or nearly power-of-two totients result in acceptable encryption (no holes other than principle) were found. A data screenshot can be seen below demonstrating this:



[5, 257, 1285, 1024, 3, 2, 683]	1024	0b100000000000	1	0.47
[17, 257, 4369, 4096, 3, 2, 2731]	4096	0b10000000000000	1	0.14
[7, 257, 1799, 1536, 257, 43, 257]	1536	0b110000000000	2	42.71
[7, 257, 1799, 1536, 65537, 43734, 1025]	1536	0b110000000000	2	42.71

Figure 8: Low hot-bits in the totient does not always mean poor encryption

The screenshot shows two septuples with “1 hot bit” totients that have acceptable transparency percentages of 0.47% and 0.14%. Other occurrences of “low hot bit” totients having acceptable transparency values were also found in the data alongside this example.

Question 2: It can be seen that the totient alone is not enough to make a reliable prediction with regards to transparency. **If the totient alone is not enough to predict transparency, what additional parameters play a role?**

It is from this question that the final phase of the work was initiated, which can be found described in detail in the next section.

Phase 5: Targeted septuple analysis and the emergence of common transparency profiles

In order to further tease out and understand the observed pattern in the totient and its relationship to transparency, various septuples from the original data set (“primes_1_to_100_holes.csv”) were isolated and tested against different choices of E. As a reminder, the original data was analyzed only for the common public keys of 3, 5, 17, 257, and 65537. However the following analysis involves using many different values of E with the goal being to observe how the transparency percentages change while only varying this parameter. The values of E were chosen to be all prime numbers between 3 and 65537, such that coprimality conditions are satisfied^[3].

The first septuples analyzed were taken from the highest transparency sets in the original data, specifically with totients that were “1 hot bit”, or a direct power of two. A screenshot demonstrating the output of a septuple from this set [17, 257, 4369, 4096, 257, 241, 3841] sorted by largest-to-smallest transparency can be seen below. Data of this format is referred to in this report as a “transparency profile”, showing the value of E used and the associated transparency for all values of E.

p, q, n, Phi, e, k, d	N	Totient	E	Transparency Percentage
[17, 257, 4369, 4096, 257, 241, 3841]	4369	4096	257	99.95
[17, 257, 4369, 4096, 769, 625, 3329]	4369	4096	769	99.95
[17, 257, 4369, 4096, 3329, 625, 769]	4369	4096	3329	99.95
[17, 257, 4369, 4096, 7681, 962, 513]	4369	4096	7681	99.95
[17, 257, 4369, 4096, 7937, 498, 257]	4369	4096	7937	99.95
[17, 257, 4369, 4096, 9473, 6515, 2817]	4369	4096	9473	99.95
[17, 257, 4369, 4096, 10753, 4035, 1537]	4369	4096	10753	99.95
[17, 257, 4369, 4096, 11777, 1475, 513]	4369	4096	11777	99.95
[17, 257, 4369, 4096, 12289, 3, 1]	4369	4096	12289	99.95
[17, 257, 4369, 4096, 13313, 9988, 3073]	4369	4096	13313	99.95
[17, 257, 4369, 4096, 14081, 7924, 2305]	4369	4096	14081	99.95
[17, 257, 4369, 4096, 14593, 6388, 1793]	4369	4096	14593	99.95
[17, 257, 4369, 4096, 15361, 3844, 1025]	4369	4096	15361	99.95
[17, 257, 4369, 4096, 17921, 11205, 2561]	4369	4096	17921	99.95
[17, 257, 4369, 4096, 18433, 9221, 2049]	4369	4096	18433	99.95
[17, 257, 4369, 4096, 19457, 4869, 1025]	4369	4096	19457	99.95
[17, 257, 4369, 4096, 22273, 12534, 2305]	4369	4096	22273	99.95
[17, 257, 4369, 4096, 23041, 8646, 1537]	4369	4096	23041	99.95
[17, 257, 4369, 4096, 23297, 7286, 1281]	4369	4096	23297	99.95
[17, 257, 4369, 4096, 25601, 19207, 3073]	4369	4096	25601	99.95
[17, 257, 4369, 4096, 26113, 16327, 2561]	4369	4096	26113	99.95
[17, 257, 4369, 4096, 26881, 11767, 1793]	4369	4096	26881	99.95
[17, 257, 4369, 4096, 30977, 13560, 1793]	4369	4096	30977	99.95
[17, 257, 4369, 4096, 31489, 9848, 1281]	4369	4096	31489	99.95
[17, 257, 4369, 4096, 32257, 4040, 513]	4369	4096	32257	99.95
[17, 257, 4369, 4096, 36097, 6777, 769]	4369	4096	36097	99.95
[17, 257, 4369, 4096, 36353, 4553, 513]	4369	4096	36353	99.95

Figure 9: Snippet of transparency profile for septuple with totient = 4096

As shown in the screenshot, the different transparency values with regards to varying values of E is seen. As data in this manner was generated, commonalties were observed for certain septuples, leading to observation 2:

Observation 2: Certain septuples have identical “transparency profiles” when their totient is a multiple of the same value

It was found that some septuples had identical results with regards to the value of E chosen and the resulting transparency percentages when certain conditions were met. For example, consider the following three septuples taken from the “1 hot bit” set:

Septuple 1: [3, 257, 771, 512, E, D, K]

Septuple 2: [5, 257, 1285, 1024, E, D, K]

Septuple 3: [17, 257, 4369, 4096, E, D, K]

The E, D, and K values have been left generic, however the important observation to note is the fact that all the Totients are a direct power of two (512, 1024, 4096), as evidenced by the fact that all three sets share 257 as one of the P/Q values (meaning the totient is a multiple of 256 and therefore always a multiple of a power-of-two, regardless of the other initial prime value).

The transparency profiles for these three septuples can be seen below:

A	B	C	D	E
p, q, n, Phi, e, k, d	N	Totient	E	Transparency Percentage
[3, 257, 771, 512, 257, 129, 257]	771	512	257	99.74
[3, 257, 771, 512, 769, 386, 257]	771	512	769	99.74
[3, 257, 771, 512, 3329, 1671, 257]	771	512	3329	99.74
[3, 257, 771, 512, 7681, 15, 1]	771	512	7681	99.74
[3, 257, 771, 512, 7937, 3984, 257]	771	512	7937	99.74
[3, 257, 771, 512, 9473, 4755, 257]	771	512	9473	99.74
[3, 257, 771, 512, 10753, 21, 1]	771	512	10753	99.74
[3, 257, 771, 512, 11777, 23, 1]	771	512	11777	99.74
[3, 257, 771, 512, 12289, 24, 1]	771	512	12289	99.74
[3, 257, 771, 512, 13313, 26, 1]	771	512	13313	99.74
[3, 257, 771, 512, 14081, 7068, 257]	771	512	14081	99.74
[3, 257, 771, 512, 14593, 7325, 257]	771	512	14593	99.74
[3, 257, 771, 512, 15361, 30, 1]	771	512	15361	99.74
[3, 257, 771, 512, 17921, 35, 1]	771	512	17921	99.74
[3, 257, 771, 512, 18433, 36, 1]	771	512	18433	99.74
[3, 257, 771, 512, 19457, 38, 1]	771	512	19457	99.74
[3, 257, 771, 512, 22273, 11180, 257]	771	512	22273	99.74
[3, 257, 771, 512, 23041, 45, 1]	771	512	23041	99.74
[3, 257, 771, 512, 23297, 11694, 257]	771	512	23297	99.74
[3, 257, 771, 512, 25601, 50, 1]	771	512	25601	99.74
[3, 257, 771, 512, 26113, 51, 1]	771	512	26113	99.74
[3, 257, 771, 512, 26881, 13493, 257]	771	512	26881	99.74
[3, 257, 771, 512, 30977, 15549, 257]	771	512	30977	99.74
[3, 257, 771, 512, 31489, 15806, 257]	771	512	31489	99.74

Figure 10: Transparency profile for septuple P = 3, Q = 257 (Totient = 512)

A	B	C	D	E
p, q, n, Phi, e, k, d	N	Totient	E	Transparency Percentage
[5, 257, 1285, 1024, 257, 193, 769]	1285	1024	257	99.84
[5, 257, 1285, 1024, 769, 193, 257]	1285	1024	769	99.84
[5, 257, 1285, 1024, 3329, 2500, 769]	1285	1024	3329	99.84
[5, 257, 1285, 1024, 7681, 3848, 513]	1285	1024	7681	99.84
[5, 257, 1285, 1024, 7937, 1992, 257]	1285	1024	7937	99.84
[5, 257, 1285, 1024, 9473, 7114, 769]	1285	1024	9473	99.84
[5, 257, 1285, 1024, 10753, 5387, 513]	1285	1024	10753	99.84
[5, 257, 1285, 1024, 11777, 5900, 513]	1285	1024	11777	99.84
[5, 257, 1285, 1024, 12289, 12, 1]	1285	1024	12289	99.84
[5, 257, 1285, 1024, 13313, 13, 1]	1285	1024	13313	99.84
[5, 257, 1285, 1024, 14081, 3534, 257]	1285	1024	14081	99.84
[5, 257, 1285, 1024, 14593, 10959, 769]	1285	1024	14593	99.84
[5, 257, 1285, 1024, 15361, 15, 1]	1285	1024	15361	99.84
[5, 257, 1285, 1024, 17921, 8978, 513]	1285	1024	17921	99.84
[5, 257, 1285, 1024, 18433, 18, 1]	1285	1024	18433	99.84
[5, 257, 1285, 1024, 19457, 19, 1]	1285	1024	19457	99.84
[5, 257, 1285, 1024, 22273, 5590, 257]	1285	1024	22273	99.84
[5, 257, 1285, 1024, 23041, 11543, 513]	1285	1024	23041	99.84
[5, 257, 1285, 1024, 23297, 5847, 257]	1285	1024	23297	99.84
[5, 257, 1285, 1024, 25601, 25, 1]	1285	1024	25601	99.84
[5, 257, 1285, 1024, 26113, 13082, 513]	1285	1024	26113	99.84
[5, 257, 1285, 1024, 26881, 20187, 769]	1285	1024	26881	99.84
[5, 257, 1285, 1024, 30977, 23263, 769]	1285	1024	30977	99.84
[5, 257, 1285, 1024, 31489, 7903, 257]	1285	1024	31489	99.84

Figure 11: Transparency profile for septuple P = 5, Q = 257 (Totient = 1024)

A	B	C	D	E
p, q, n, Phi, e, k, d	N	Totient	E	Transparency Percentage
[17, 257, 4369, 4096, 257, 241, 3841]	4369	4096	257	99.95
[17, 257, 4369, 4096, 769, 625, 3329]	4369	4096	769	99.95
[17, 257, 4369, 4096, 3329, 625, 769]	4369	4096	3329	99.95
[17, 257, 4369, 4096, 7681, 962, 513]	4369	4096	7681	99.95
[17, 257, 4369, 4096, 7937, 498, 257]	4369	4096	7937	99.95
[17, 257, 4369, 4096, 9473, 6515, 2817]	4369	4096	9473	99.95
[17, 257, 4369, 4096, 10753, 4035, 1537]	4369	4096	10753	99.95
[17, 257, 4369, 4096, 11777, 1475, 513]	4369	4096	11777	99.95
[17, 257, 4369, 4096, 12289, 3, 1]	4369	4096	12289	99.95
[17, 257, 4369, 4096, 13313, 9988, 3073]	4369	4096	13313	99.95
[17, 257, 4369, 4096, 14081, 7924, 2305]	4369	4096	14081	99.95
[17, 257, 4369, 4096, 14593, 6388, 1793]	4369	4096	14593	99.95
[17, 257, 4369, 4096, 15361, 3844, 1025]	4369	4096	15361	99.95
[17, 257, 4369, 4096, 17921, 11205, 2561]	4369	4096	17921	99.95
[17, 257, 4369, 4096, 18433, 9221, 2049]	4369	4096	18433	99.95
[17, 257, 4369, 4096, 19457, 4869, 1025]	4369	4096	19457	99.95
[17, 257, 4369, 4096, 22273, 12534, 2305]	4369	4096	22273	99.95
[17, 257, 4369, 4096, 23041, 8646, 1537]	4369	4096	23041	99.95
[17, 257, 4369, 4096, 23297, 7286, 1281]	4369	4096	23297	99.95
[17, 257, 4369, 4096, 25601, 19207, 3073]	4369	4096	25601	99.95
[17, 257, 4369, 4096, 26113, 16327, 2561]	4369	4096	26113	99.95
[17, 257, 4369, 4096, 26881, 11767, 1793]	4369	4096	26881	99.95
[17, 257, 4369, 4096, 30977, 13560, 1793]	4369	4096	30977	99.95
[17, 257, 4369, 4096, 31489, 9848, 1281]	4369	4096	31489	99.95

Figure 12: Transparency profile for septuple P = 17, Q = 257 (Totient = 4096)

As one can see from the screenshots, **there is a 1-to-1 correspondence regarding the “key strength” for every single value of E used** (screenshot depicts only highest transparency entries, however full data can be seen in repository under the folder “Excel_Data/transparency_profile_data/1_hot_bit_totients/pattern_1”).

It is worth noting that the percentages differ slightly due to the fact that the range of encryptable integers is different for each set (different N values), however they are considered to have identical profiles since the E values match verbatim with regards to their relative transparency strength. Large amounts of data were collected for a sampling of septuples from the 1-hot-bit, 2-hot-bit, 3-hot-bit, 4-hot-bit, and 5-hot-bit sets. A total of 13 patterns were identified that were able to be matched with another septuple. One may note that the three profiles shown above shared 257 as a common initial prime. Some screenshots illustrating observation two from other sets can be seen below:

« Excel_Data > transparency_profile_data > 2_hot_bit_totients > pattern_5		
<input type="checkbox"/>	Name	Date modified
	prime_19_and_457_specific_analysis.csv	12/9/2019 2:16 PM
	prime_37_and_457_specific_analysis.csv	12/9/2019 2:16 PM
	prime_73_and_457_specific_analysis.csv	12/9/2019 2:16 PM

Figure 13: Pattern 5 in the 2_hot_bit_totient set all share 457 as an initial prime

« Excel_Data > transparency_profile_data > 3_hot_bit_totients > pattern_3




<input type="checkbox"/> Name	Date modified
 prime_3_and_353_specific_analysis.csv	11/20/2019 5:40 PM
 prime_5_and_353_specific_analysis.csv	11/21/2019 1:29 PM
 prime_17_and_353_specific_analysis.csv	11/20/2019 7:00 PM

Figure 14: Pattern 3 in the 3_hot_bit_totient set all share 353 as an initial prime

« Excel_Data > transparency_profile_data > 5_hot_bit_totients > pattern_1



<input type="checkbox"/> Name	Date modified
 prime_7_and_271_specific_analysis.csv	11/21/2019 1:16 PM
 prime_11_and_271_specific_analysis.csv	11/21/2019 1:17 PM

Figure 15: Pattern 1 in the 5_hot_bit_totient set all share 271 as an initial prime

It can be seen that commonly patterned sets share this commonality, leading to the question:

Question 3: Do septuples with a common initial prime and hot-bit-value in the totient mean they will have the same transparency profile?

Answer: No. Septuples that have the commonalities described were found with different transparency profiles. The file “prime_41_and_257_specific_analysis.csv” can be found as evidence of this inside the “2_hot_bit_patterns” folder. While this totient is also a multiple of 256 and a “2_hot_bit” value as in example shown above, it has an entirely different transparency profile:

A	B	C	D	E
p, q, n, Phi, e, k, d	N	Totient	E	Transparency Percentage
[41, 257, 10537, 10240, 7681, 1921, 2561]	10537	10240	7681	99.98
[41, 257, 10537, 10240, 14081, 8802, 6401]	10537	10240	14081	99.98
[41, 257, 10537, 10240, 15361, 7682, 5121]	10537	10240	15361	99.98
[41, 257, 10537, 10240, 17921, 4482, 2561]	10537	10240	17921	99.98
[41, 257, 10537, 10240, 23041, 17283, 7681]	10537	10240	23041	99.98
[41, 257, 10537, 10240, 25601, 12803, 5121]	10537	10240	25601	99.98
[41, 257, 10537, 10240, 26881, 10083, 3841]	10537	10240	26881	99.98
[41, 257, 10537, 10240, 40961, 4, 1]	10537	10240	40961	99.98
[41, 257, 10537, 10240, 49921, 6245, 1281]	10537	10240	49921	99.98
[41, 257, 10537, 10240, 57601, 21606, 3841]	10537	10240	57601	99.98
[41, 257, 10537, 10240, 60161, 7526, 1281]	10537	10240	60161	99.98
[41, 257, 10537, 10240, 61441, 6, 1]	10537	10240	61441	99.98
[41, 257, 10537, 10240, 641, 601, 9601]	10537	10240	641	50.17
[41, 257, 10537, 10240, 4481, 2521, 5761]	10537	10240	4481	50.17
[41, 257, 10537, 10240, 9601, 601, 641]	10537	10240	9601	50.17
[41, 257, 10537, 10240, 12161, 9882, 8321]	10537	10240	12161	50.17
[41, 257, 10537, 10240, 13441, 9242, 7041]	10537	10240	13441	50.17
[41, 257, 10537, 10240, 16001, 7002, 4481]	10537	10240	16001	50.17
[41, 257, 10537, 10240, 19841, 1242, 641]	10537	10240	19841	50.17
[41, 257, 10537, 10240, 21121, 19803, 9601]	10537	10240	21121	50.17
[41, 257, 10537, 10240, 35201, 19804, 5761]	10537	10240	35201	50.17
[41, 257, 10537, 10240, 39041, 7324, 1921]	10537	10240	39041	50.17
[41, 257, 10537, 10240, 54401, 37406, 7041]	10537	10240	54401	50.17
[41, 257, 10537, 10240, 55681, 31326, 5761]	10537	10240	55681	50.17
[41, 257, 10537, 10240, 62081, 58207, 9601]	10537	10240	62081	50.17
[41, 257, 10537, 10240, 63361, 51487, 8321]	10537	10240	63361	50.17
[41, 257, 10537, 10240, 1601, 1351, 8641]	10537	10240	1601	25.27

Figure 16: Septuple with alternate transparency profile, despite the commonalities described (2 hot bit totient, multiple of 256)

It can be seen that more variables and/or unseen factors are at play with regards to a septuples transparency profile. Many sets were analyzed that had no matching pattern, and can be found in the “no_matching_pattern” folder inside the transparency profile data folders.

Possible Future Work

The data and results described in phase 4 and 5 of this research are certainly intriguing. A transparency analysis of approximately 25,000 septuples found a mathematical commonality in values of the totient with regards to the highest transparency sets. Further analysis showed that high transparency was not solely reliant on the totient, but was governed by the choice of E value as well. Septuples with totients that shared common factors were sometimes found to have identical transparency profiles for over 6500 different values of E. Based on these conclusions, some questions and hypothesis to verify can be examined:

Hypothesis 1.0 (Unverified): Data collected in this report indicates that while certain totient patterns do not mean poor encryption, poor encryption appears to be associated with particularly patterned totients. Understanding this relationship could lead to the ability to statistically “throw out” bad keys by totient analysis alone. **More data needs to be collected and analyzed to observe the re-emergence or non re-emergence of the pattern on larger data sets to further gain confidence.**

With regards to hypothesis one, the statistical significance of the totient pattern emergence is hard to ignore. However it was shown that the relationship is not completely flushed out, and exceptions exist. Understanding this in even more detail would be beneficial and useful information.

Future work: Generate larger quantities of data using larger initial primes and more computing power. Observe if the same trend occurs for larger, more realistic data sets (512, 1024, 2048 bit N).

Hypothesis 2.0 (Unverified): All septuples have a “transparency profile” that matches the transparency profile of other septuples when certain conditions are met.

It can be seen that common transparency profiles exists between septuples with a common initial prime (and therefore a common multiple totient). However the exact relationship between septuples that causes the transparency profiles to be identical requires more research. Understanding this relationship could lead to the ability to derive a septuples transparency profile mathematically, and by extension, derive which keys are good and or bad for a given set.

Future work: Further explore the nature of the relationship between septuples that have common transparency profiles. More specifically, explore mathematical patterns and/or relationships that could be used to predict whether a septuple will (or will not) have a given transparency profile. In addition, finding a matching profile for all data that has not yet been matched with other septuples would be illuminating.

Conclusions

In summary, a concentrated effort to further flush out the findings of this research could potentially lead to newer and computationally cheaper methods of fixed point prediction in RSA encryption. Even though fixed point occurrence is statistically unlikely for modern implementations, the possibility of accidental poor key generation still exists. The ability to confidently avoid any fixed point, or ensure the generation of strong keys for given initial parameters, has strong value in any serious security implementation that takes advantage of RSA encryption.

References

1. R. L. Rivest, A. Shamir, and L. Adleman. 1978. *A Method for Obtaining Digital Signatures and Public-key Cryptosystems*, Commun. ACM, Vol. 21, no. 2, pp. 120–126.
DOI: [10.1145/359340.359342](https://doi.org/10.1145/359340.359342)
2. G. Blakley and I. Borosh. 1979. *Rivest-Shamir-Adleman Public Key Cryptosystems do not Always Conceal Messages*, Computers and Mathematics with Applications, Vol. 5, Issue 3, pp.169-178. DOI: [https://doi.org/10.1016/0898-1221\(79\)90039-7](https://doi.org/10.1016/0898-1221(79)90039-7)
3. Sadr, Behnaz. 1992. *Finding Cases of Ciphertext equal to Plaintext in the RSA Algorithm*, School of Electrical and Computer Engineering, OSU. DOI: <https://hdl.handle.net/11244/10269>