Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter known, including display on the world wide web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Richard Michael Oh	April 12, 2010

Cryptanalysis of Small-Valued Secret Exponents in RSA Cryptosystems

by

Richard Michael Oh

Advisor Skip Garibaldi

Department of Mathematics and Computer Science

Skip Garibaldi

Advisor

Eric Brussel

Committee Member

Jose Soria

Committee Member

April 12, 2010

Cryptanalysis of Small-Valued Secret Exponents in RSA Cryptosystems

by

Richard Michael Oh

Advisor Skip Garibaldi

Abstract of

A thesis submitted to the Faculty of Emory College

of Emory University in partial fulfillment

of the requirements of the degree of

Bachelor of Sciences with Honors

Department of Mathematics and Computer Science

2010

Abstract

Cryptanalysis of Small-Valued Secret Exponents in RSA Cryptosystems

by

Richard M. Oh

A vulnerability of the RSA encryption system that uses small-valued secret exponents is written here. When small-valued secret exponents are used, the encryption system is exposed to an attack via a continued fractions algorithm. The original algorithm was discovered by Michael J. Weiner. The algorithm is able to use public key information to find the private key information. The attack is valuable since it is able to discover the factors of the modulus and the secret exponent in polynomial time. This paper expands the analysis of Wiener's original paper and implements the algorithm through Python programming.

Cryptanalysis of Small-Valued Secret Exponents in RSA Cryptosystems

by

Richard Michael Oh

Advisor Skip Garibaldi

A thesis submitted to the Faculty of Emory College

of Emory University in partial fulfillment

of the requirements of the degree of

Bachelors of Science with Honors

Department of Mathematics and Computer Science

2010

Acknowledgements

I would like to express the deepest appreciation to my committee chair and advisor, Professor Skip Garibaldi, who has the attitude and substance of a genius. With his continual guidance and his spirit of adventure in regard to research and scholarship, he made this honor thesis possible.

I would like to thank my committee members, Professor Eric Brussel and Dr. Jose Soria, for teaching me and guiding me through my college career. With their positive reinforcements along with their brilliance in teaching, they have developed my character academically.

I would also like to thank my academic advisor, Professor Emily Hamilton, for supporting and guiding me through my college career. Without her continual support and encouragement, I would not be where I am today.

I would also like to thank Ashlyn Winkler for her comments on my paper. I would like to thank Shahein Tajmir for his insight on my program.

Finally, I would like to thank my parents, Thomas and Yung Oh, my brother, Jonathan Oh, and Melissa Steele for their continual support and encouragement throughout the time period of my honors thesis.

Contents

1 Introduction

The RSA cryptosystem is one of the most well-known public key cryptosystems in the world today. Most people unknowingly surf the internet without realizing that their sessions are being protected by this encryption system. Each public key consists of two numbers: the public number, or modulus, and the public exponent, or the encrypting exponent. A common form of weakness that arises within RSA is the poor choice of keys. Further readings on these weaknesses can be found in [1]. The attack within this paper is from Michael Wiener's paper, *Cryptananlysis of Short RSA Secret Exponents* [5]. His paper exploits a weakness and breakdown of the RSA cryptosystem when the secret exponent is small relative to the modulus. Through the use of a continued fractions algorithm, the attacker is able to produce the secret exponent and the prime factors of the modulus using only public information. Furthermore, this attack is especially useful since the attack is done in polynomial time.

There are variety of situations where the use of short secret exponents is desirable. A smaller secret exponent allows for faster decryption execution time since RSA decryption time is roughly proportional to the number of bits in the exponent. Therefore, it is sometimes convienent to use smaller exponents, especially if there is a vast difference in computing speeds between the encryption and decryption devices. One example of this situation is the use of smart cards with computers. It would be convienent for the smart card to possess a short secret exponent since the computing powers of a smart card are vastly inferior to the computer. The ease of use and convenience of small secret exponents creates a security flaw, which we shall discuss in this

paper.

This paper is an expanded version of Weiner's original paper. It breaks down the original argument and builds it back up with some variation to the verfication step. Furthermore, the algorithm has been implemented through the use of Python programming, where the user can test various keys.

2 Continued Fractions Background

This section will cover the necessary background on continued fractions allowing the reader to effectively understand and use the continued fractions algorithm in the next section.

Definition 2.1. *A continued fraction is an expression with the form*

$$\begin{aligned} &\cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{\cdots \cfrac{}{b_{m-1} + \cfrac{a_m}{b_m}}}}} \\ &= a_1/(b_1 + a_2/(b_2 + a_3/(.../(b_{m-1} + a_m/b_m)...))). \end{aligned} \tag{2.1}$$

We are specifically interested in the particular form of continued fractions where the numerators, or a_i's in (2.1) are equal to one. For convience, we shall denote this as

$$\langle q_0, q_1, ..., q_m \rangle = q_0 + 1/(q_1 + 1/(q_2 + 1/(.../(q_{m-1} + 1/q_m)...))). \tag{2.2}$$

Example 2.2.

$$\langle 2, 1, 3, 5 \rangle = 2 + 1/(1 + 1/(3 + 1/5)) = \frac{58}{21}.$$

For vocabulary purposes, we will call $\frac{58}{21}$ the *convergent fraction* while $\langle 2, 1, 3, 5\rangle$ is called the *continued fraction expansion*. Also, continued fractions do not need to be finite, but we will not address this issue since our numerators and denominators are positive integers, thus resulting in finite expansions [3].

We will now cover how to get a continued fractions expansion from a rational number f. The first step is to subtract the integer part away from f. The following steps are then done in repetition until the remainder is zero: Invert the remainder, subtract away the integer part away from the inverted remainder, and repeat. We shall call q_i the integer quotient and r_i the remainder at step i. We will call m the total number of inversion steps:

$$\begin{aligned} &q_0 = \lfloor f \rfloor, \quad r_0 = f - q_0, \qquad \text{and} \\ &q_i = \left\lfloor \frac{1}{r_{i-1}} \right\rfloor, \quad r_i = \frac{1}{r_{i-1}} - q_i, \qquad \text{for } i = 1, 2, ..., m. \end{aligned} \tag{2.3}$$

When $r_j = 0$, we set $j = m$, and the algorithm for generating the continued fractions expansion for f terminates. Since the algorithm generating the expansion results in the remainder at step m being zero, we have that $f = \langle q_0, q_1, ..., q_m\rangle$. There are two immediate results that emerge at this point. First,

$$q_m \geq 2. \tag{2.4}$$

We get this from the fact that if $q_m = 1$, then $0 = r_m = \frac{1}{r_{m-1}} - q_m$, from (2.3). We get from here that $r_{m-1} = 1$, which is impossible since $0 < r_i < 1$, by construction.

Second, for any $x > 0$,

$$\begin{aligned} \langle q_0, q_1, ..., q_m \rangle &< \langle q_0, q_1, ..., q_{m-1}, q_m + x \rangle, \quad \text{if } m \text{ is even,} \\ \langle q_0, q_1, ..., q_m \rangle &> \langle q_0, q_1, ..., q_{m-1}, q_m + x \rangle, \quad \text{if } m \text{ is odd.} \end{aligned} \tag{2.5}$$

This result comes directly from the number of nestings and inversions that occur from (2.2).

We will now begin to discuss the relationship between numerators and denominators at step i of the continued fraction expansion and its relationship with the quotients. We will define n_i and d_i inductively.

Definition 2.3. *Let*

$$n_{-2} = 0, \quad n_{-1} = 1, \quad d_{-2} = 1, \quad d_{-1} = 0.$$

Then we define n_i and d_i as follows:

$$n_0 = q_0, \quad d_0 = 1,$$

$$n_1 = q_0 q_1 + 1, \quad d_1 = q_1,$$

$$n_i = q_i n_{i-1} + n_{i-2}, \quad d_i = q_i d_{i-1} + d_{i-2}, \ for\ i = 2, 3, ..., m.$$

Lemma 2.4. *Let n_i, d_i be defined as above. Then*

$$\frac{n_i}{d_i} = \langle q_0, q_1, ..., q_i \rangle.$$

Proof. First, we will show for $i = 0$. We have from Definition 2.3 that

$$\frac{n_0}{d_0} = \frac{q_0}{1} = q_0 = \langle q_0 \rangle$$

Next, we will show for $i = 1$. We have that

$$\frac{n_1}{d_1} = \frac{q_0 q_1 + 1}{q_1} = q_0 + \frac{1}{q_1} = \langle q_0, q_1 \rangle.$$

We will use induction for the rest of the proof. Suppose true for the i case. We have that

$$c_i = \frac{n_i}{d_i} = \frac{q_i n_{i-1} + n_{i-2}}{q_i d_{i-1} + d_{i-2}} = \langle q_0, q_1, ..., q_i \rangle. \tag{2.6}$$

We wish to prove that the $i+1$ case holds true. To do this we use equation (2.6) to help us supply a proof that

$$c_{i+1} = \frac{n_{i+1}}{d_{i+1}} = \frac{q_{i+1} n_i + n_{i-1}}{q_{i+1} d_i + d_{i-1}} = \langle q_0, q_1, ..., q_i, q_{i+1} \rangle.$$

From 2.3, we substitute in n_i, d_i to get

$$\frac{n_{i+1}}{d_{i+1}} = \frac{q_{i+1}(q_i n_{i-1} + n_{i-2}) + n_{i-1}}{q_{i+1}(q_i d_{i-1} + d_{i-2}) + d_{i-1}}.$$

By multiplying out and rearranging, we are able to get

$$\frac{q_{i+1}(q_i n_{i-1} + n_{i-2}) + n_{i-1}}{q_{i+1}(q_i d_{i-1} + d_{i-2}) + d_{i-1}} = \frac{(q_i q_{i+1} + 1) n_{i-1} + q_{i+1} n_{i-2}}{(q_i q_{i+1} + 1) d_{i-1} + q_{i+1} d_{i-2}}.$$

We then divide the numerator and denominator by q_{i+1} to get

$$\frac{n_{i+1}}{d_{i+1}} = \frac{\left(q_i + \dfrac{1}{q_{i+1}}\right) n_{i-1} + n_{i-2}}{\left(q_i + \dfrac{1}{q_{i+1}}\right) d_{i-1} + d_{i-2}}.$$

In order to use 2.6, we are hoping that we can replace q_i with $q_i + \frac{1}{q_{i+1}}$ in the expansion, since there is a similiarity with

$$\frac{n_i}{d_i} = \frac{q_i n_{i-1} + n_{i-2}}{q_i d_{i-1} + d_{i-2}} = \langle q_0, q_1, ..., q_i \rangle$$

and

$$\frac{n_{i+1}}{d_{i+1}} = \frac{\left(q_i + \dfrac{1}{q_{i+1}}\right) n_{i-1} + n_{i-2}}{\left(q_i + \dfrac{1}{q_{i+1}}\right) d_{i-1} + d_{i-2}}.$$

This is possible only if we are able to prove that the numbers $n_{i-2}, d_{i-2}, n_{i-1}, d_{i-1}$ do not change their values when we tamper with q_i.

To see that these values do not change, we must look at the manner in which they are calculated. In equation (2.6), we first replace i by $i-2$ and then by $i-1$. We observe the following succession:

$$\frac{n_{i-2}}{d_{i-2}} = \frac{q_{i-2}n_{i-3} + n_{i-4}}{q_{i-2}d_{i-3} + d_{i-4}}$$

and

$$\frac{n_{i-1}}{d_{i-1}} = \frac{q_{i-1}n_{i-2} + n_{i-3}}{q_{i-1}d_{i-2} + d_{i-3}}.$$

We can observe from the above that the numbers n_{i-1} and d_{i-1} are dependent on the number q_{i-1} and the numbers $n_{i-2}, n_{i-3}, d_{i-2}, d_{i-3}$ for $i > 3$, which are in turn dependent on the preceding q's, n's, and d's. Therefore, the numbers $n_{i-1}, n_{i-2}, d_{i-1}, d_{i-2}$ are dependent on the first $i-1$ quotients and are independent of q_i. Hence, they will not change when we replace q_i with $q_i + \frac{1}{q_{i+1}}$.

Therefore, we are able to substitute q_i with $q_i + \frac{1}{q_{i+1}}$, thus getting us

$$\frac{n_{i+1}}{d_{i+1}} = \frac{\left(q_i + \frac{1}{q_{i+1}}\right) n_{i-1} + n_{i-2}}{\left(q_i + \frac{1}{q_{i+1}}\right) d_{i-1} + d_{i-2}} = \langle q_0, q_1, ..., q_i + \frac{1}{q_{i+1}} \rangle = \langle q_0, q_1, ..., q_i, q_{i+1} \rangle.$$

□

This result leads us to a direct corollary for the above Lemma.

Corollary 2.5. *Let the n_i's, d_i's, and q_i's be as above. Then we have that:*

$$n_i d_{i-1} - n_{i-1} d_i = -(-1)^i \; for \; i = 1, 2, ..., m.$$

Proof. We will prove this by induction on i.

Base Case $i = 1$

From Definition 2.3, we have

$$n_1 d_0 - n_0 d_1 = (q_0 q_1 + 1)(1) - (q_0 q_1) = 1 = -(-1)^1.$$

Induction Step

Let us assume to be true up to the $i - 1$ case, which is as followed:

$$n_{i-1} d_{i-2} - n_{i-2} d_{i-1} = -(-1)^{i-1}.$$

We must show true for the i case:

$$n_i d_{i-1} - n_{i-1} d_i = -(-1)^i.$$

Using the relationship in Lemma 2.4, we substitute in for n_i and d_i to obtain

$$n_i d_{i-1} - n_{i-1} d_i = (q_i n_{i-1} + n_{i-2}) d_{i-1} - n_{i-1}(q_i d_{i-1} + d_{i-2}).$$

Then, by simplification using algebra, we get

$$-(n_{i-1} d_{i-2} - n_{i-2} d_{i-1}) = (-1) \cdot -(-1)^{i-1} = -(-1)^i.$$

□

Corollary 2.6. *Let n_i, d_i be as above. Then n_i and d_i are relatively prime for $i = 0, 1, ..., m$.*

Proof. This is obvious for $i = 0$ because $d_0 = 1$. For $i = 1, 2, ..., m$, if a prime p divided n_i and d_i, then p would divide -1, by Corollary 2.5, which is a contradiction. □

We now have this equation as a direct consequence of Lemma 2.4 and Corollary 2.6:

$$\begin{gathered}\frac{n_i}{d_i} = \langle q_0, q_1, ..., q_i \rangle, \quad \gcd(n_i, d_i) = 1, \quad \text{for } i = 0, 1, ..., m \\ n_i = q_i n_{i-1} + n_{i-2}, d_i = q_i d_{i-1} + d_{i-2}.\end{gathered} \tag{2.7}$$

Lemma 2.7. *Let f be a fraction and $f > 0$ and $m \geq 1$ where $f = \langle q_0, q_1, ..., q_m \rangle$. Then*

$$q_i \geq \begin{cases} 0 & \text{if } i = 0 \\ 1 & \text{if } 1 \leq i < m \\ 2 & \text{if } i = m \end{cases}.$$

Proof. We will first show that $q_0 \geq 0$. From the construction of q_i from (2.3), we have $q_0 \geq 0$ when $f > 0$. Next, we will show that $q_i \geq 1$ when $1 \leq i < m$. Here, again by construction of q_i from (2.3), we have that

$$q_i = \left\lfloor \frac{1}{r_{i-1}} \right\rfloor,$$

which implies that when $f > 0$, $0 < r_i < 1$ for $1 \leq i < m$. This gives us that

$$q_i \geq 0 \text{ for } 1 \leq i < m.$$

It will suffice to show that $q_i \neq 0$. We will do this by contradiction.

Suppose $q_i = 0$. By construction of q_i from (2.3),

$$q_i = \left\lfloor \frac{1}{r_{i-1}} \right\rfloor = 0.$$

From the above, we are able to get the fact that

$$0 < \frac{1}{r_{i-1}} < 1,$$

which directly leads us to

$$r_{i-1} > 1,$$

which is a direct contradiction of (2.3). Therefore, $q_i \geq 1$ for $1 \leq i < m$.

Finally, by (2.4), we have $q_m \geq 2$, which verifies our last case. □

Lemma 2.8. *Suppose* $f = \langle q_0, q_1, ..., q_m \rangle > 0$ *and* $\frac{n_i}{d_i} = \langle q_0, q_1, ..., q_i \rangle$. *Then* $d_i > d_{i-1} > 0$ *for* $2 \leq i \leq m$.

We claim that the Lemma fails for the $i = 1$ case because d_1 can equal 1. We can see from Example 2.2, with verification using Definition 2.3, that $d_1 = 1$, and since $d_0 = 1$, the inequality fails.

Proof. We will use induction on i to prove this.

Base Case $i = 2$

From Definition 2.3, we have $d_1 = q_1$ and $d_2 = q_2 d_1 + d_0$. Since $d_0 = 1$, $d_2 > q_2 d_1$, and $q_2 \geq 1$, we have that

$$d_2 > d_1 > 0.$$

Induction Step

Assume true for the $i - 1$ case. We will prove the claim for the general i case. From equation (2.7), we have $d_i = q_i d_{i-1} + d_{i-2}$ which gives us that $d_i > q_i d_{i-1}$. Since $q_i \geq 1$ for $i \geq 1$ and $d_{i-1} > d_{i-2} > 0$, we have

$$d_i > d_{i-1} > 0.$$

Therefore, $d_i > d_{i-1} > 0$ for $2 \leq i \leq m$. □

Corollary 2.9. *Let n_i's, d_i's, and q_i's be as above and $m \geq 2$. Then*

$$d_{m-1} \leq \frac{1}{2} d_m.$$

Proof. Using Definition 2.3 and Lemma 2.8, we have that

$$d_m > q_m d_{m-1}.$$

Since $q_m \geq 2$ from (2.4), we have $q_m d_{m-1} \geq 2d_{m-1}$. Since $d_m > 2d_{m-1}$, we get $d_{m-1} < \frac{1}{2} d_m$. □

Lemma 2.10. *Suppose that*

$$\langle q_0, q_1, q_2, ... \rangle < x < \langle q_0', q_1', q_2', ... \rangle.$$

If $q_0 = q_0'$, then

$$\langle q_1, q_2, q_3, ... \rangle > \frac{1}{x - q_0} > \langle q_1', q_2', q_3', ... \rangle.$$

Proof. Since $q_0 = q_0'$, we have through properties of inequalities that

$$\langle q_0, q_1, q_2, ... \rangle - q_0 < (x - q_0) < \langle q_0', q_1', q_2', ... \rangle - q_0.$$

Through simplification, we have

$$\langle 0, q_1, q_2, ... \rangle < (x - q_0) < \langle 0, q_1', q_2', ... \rangle.$$

Since $0 < \langle 0, q_1, q_2, ... \rangle < 1$ and $0 < \langle 0, q_1', q_2', ... \rangle < 1$ by construction of continued fractions, and the fact for $0 < a < b$, we have

$$\frac{1}{a} > \frac{1}{b}$$

since $f(x) = \frac{1}{x}$ is a strictly decreasing function, we get

$$\langle q_1, q_2, q_3, ... \rangle > \frac{1}{x - q_0} > \langle q_1', q_2', q_3', ... \rangle.$$

□

We have covered sufficient background to proceed on to the Continued Fractions Algorithm.

3 Continued Fractions Algorithm

Keeping the above background in mind, we shall discuss a continued fractions algorithm that will allow us to produce a reasonable candidate for a rational number f. We begin this discussion letting f' be a underestimate for a fraction f, given by the relationship from the equation below.

$$f' = f(1 - \delta), \text{ for some } \delta \geq 0. \tag{3.1}$$

We can easily rearrange this equation to

$$\delta = 1 - \frac{f'}{f}, \tag{3.2}$$

which will be a useful relationship later on. The following is Wiener's original continued fractions algorithm[5]. We are given a continued fraction f', where we assume that f' is a proper underestimate, and we are given an oracle that will tell us whether or not a given number is the desired fraction, f. The following step will be repeated until the oracle reveals to us that the generated fraction is the desired one, or the algorithm terminates. If δ is not of proper size, then the algorithm will terminate by itself since

all rational fractions have a finite continued fractions expansion[3], and thus leading to a finite number of reiterations of the algorithm. We shall discuss in detail what our oracle is in a following section.

Let q'_i be the i^{th} quotient of f', generated as written in (2.3). Let j be the length of the continued fractions expansion for f', since the expansion for f' is finite. Here are the steps of the algorithm:

1. Give f' to the oracle. If f' is the correct number, then terminate. Otherwise, proceed onto Step 2.

2. Generate the next quotient q'_i of the continued fraction expansion for f'.

3. Using equation (2.7), we construct the fraction equal to

$$\langle q'_0, q'_1, ..., q'_{i-1}, q'_i + 1\rangle, \quad \text{if } i \text{ is even}$$

$$\langle q'_0, q'_1, ..., q'_{i-1}, q'_i\rangle, \quad \text{if } i \text{ is odd.}$$

4. We give the generated number to the oracle. If the number is incorrect, we go back to Step 2, unless we have generated q'_j, which in this case, we quit. If the number is correct, the algorithm terminates.

We must understand that although we are assuming that f' is an underestimate of some number f, we do not know what f is. Therefore, we must evaluate what our δ from (3.1) should be in order for us to find f. We claim that if

$$\delta < \frac{1}{\frac{3}{2}n_m d_m}, \tag{3.3}$$

where n_m and d_m are the numerator and denominator of f, then the algorithm will be able to produce f from f'. We will examine the implications of δ from (3.3) and see

how it leads us to the production of f from f' from the algorithm. We will examine the relationship on how the size of δ affects the outcome of the algorithm starting with f'. Studying this relationship will lead us to the theorem that defines the requirement for the algorithm to succeed.

Proposition 3.1. *Suppose that* $0 \leq \delta < \dfrac{1}{\frac{3}{2}n_m d_m}$. *Then*

$$\begin{aligned} \langle q_0, q_1, ..., q_{m-1}, q_m - 1\rangle < f' \leq \langle q_0, q_1, ..., q_m\rangle, \qquad &\textit{if } m \textit{ is even,} \\ \langle q_0, q_1, ..., q_{m-1}, q_m + 1\rangle < f' \leq \langle q_0, q_1, ..., q_m\rangle, \qquad &\textit{if } m \textit{ is odd.} \end{aligned} \tag{3.4}$$

Proof. From (3.1), we have that if $\delta = 0$, $f' = f$. Thus, our upper bound holds. Therefore, we only have to justify our lower bounds. We perform a separate analysis for the following cases: $m = 0$, $m = 1$, $m \geq 2$ and even, and $m \geq 3$ and odd.

Case 1 $m = 0$

Suppose $\delta < \dfrac{1}{\frac{3}{2}n_0 d_0}$. Then

$$\delta < \frac{1}{\frac{3}{2}n_0 d_0} < \frac{1}{n_0 d_0}.$$

By Definition 2.3, we know that $d_0 = 1$ and that $n_0 = q_0$. Thus,

$$\delta < \frac{1}{q_0}.$$

From (3.2), we have $\delta = 1 - \dfrac{f'}{q_0} < \dfrac{1}{q_0}$, which can be rearranged to

$$f' > q_0 - 1.$$

Since $m = 0$ is even, we get $\langle q_0 - 1\rangle < f'$ as needed.

Case 2 $m = 1$

Suppose that $\delta < \dfrac{1}{\frac{3}{2}n_1 d_1}$. By Definition 2.3, we can substitute $n_1 = q_0 q_1 + 1$ and

$d_1 = q_1$ to have

$$\delta < \frac{1}{\frac{3}{2}(q_0q_1 + 1)q_1}.$$

For our next step, we need the following claim.

$$\text{Claim:} \quad \frac{3}{2}q_1 \geq q_1 + 1.$$

By Definition 2.3 and the fact that $m = 1$, we have that $q_1 \geq 2$. Since $\frac{1}{2}q_1 \geq 1$, we get

$$q_1 + 1 \leq q_1 + \frac{1}{2}q_1 \leq \frac{3}{2}q_1.$$

Therefore, we have $\frac{3}{2}q_1 \geq q_1 + 1$, as required.

With this claim, we have that

$$\delta < \frac{1}{\frac{3}{2}(q_0q_1 + 1)q_1} \leq \frac{1}{(q_0q_1 + 1)(q_1 + 1)}.$$

Then we will rearrange the fraction, resulting in

$$\delta < \frac{\frac{1}{q_1 + 1}}{q_0q_1 + 1} = \frac{1 - \frac{q_1}{q_1 + 1}}{q_0q_1 + 1} = \frac{\frac{1}{q_1} - \frac{1}{q_1 + 1}}{q_0 + \frac{1}{q_1}} = 1 - \left(\frac{q_0 + \frac{1}{q_1 + 1}}{q_0 + \frac{1}{q_1}} \right).$$

Using (3.2),

$$\delta = \left(1 - \frac{f'}{q_0 + \frac{1}{q_1}} \right) < \left(1 - \frac{q_0 + \frac{1}{q_1 + 1}}{q_0 + \frac{1}{q_1}} \right).$$

Solving for f', we have

$$f' > \left(q_0 + \frac{1}{q_1 + 1} \right).$$

Which gives us $\langle q_0, q_1 + 1 \rangle < f'$, as required.

Case 3 $m \geq 2$ *and even*

Suppose $\delta < \frac{1}{\frac{3}{2}n_md_m}$. Then by algebra, we have that

$$\delta < \frac{1}{\frac{3}{2}n_md_m} < \frac{1}{n_md_m}.$$

By Lemma 2.8, we know that $d_m - d_{m-1} < d_m$, therefore we get

$$\delta < \frac{1}{n_m(d_m - d_{m-1})}.$$

Since m is even

$$n_{m-1}d_{m-2} - n_{m-2}d_{m-1} = 1, \text{ by (2.5)}$$

and

$$n_m = q_m n_{m-1} + n_{m-2} \text{ and } d_m = q_m d_{m-1} + d_{m-2}, \text{ by (2.7)},$$

By substitution we get

$$\delta < \frac{n_{m-1}d_{m-2} - n_{m-2}d_{m-1}}{(q_m n_{m-1} + n_{m-2})(q_m d_{m-1} + d_{m-2} - d_{m-1})}.$$

We add

$$q_m d_{m-1} q_m n_{m-1} - q_m d_{m-1} q_m n_{m-1} + q_m d_{m-1} n_{m-2} - q_m d_{m-1} n_{m-2} + d_{m-1} q_m n_{m-1} -$$

$$d_{m-1} q_m n_{m-1} + d_{m-2} q_m n_{m-1} - d_{m-2} q_m n_{m-1} + d_{m-2} n_{m-2} - d_{m-2} n_{m-2} = 0$$

into the numerator of the above inequality to get

$$\delta < \frac{\begin{pmatrix} q_m d_{m-1} q_m n_{m-1} & - & q_m d_{m-1} q_m n_{m-1} & + & q_m d_{m-1} n_{m-2} & - \\ q_m d_{m-1} n_{m-2} + d_{m-1} q_m n_{m-1} - d_{m-1} q_m n_{m-1} + d_{m-2} q_m n_{m-1} & - \\ d_{m-2} q_m n_{m-1} + d_{m-2} n_{m-2} - d_{m-2} n_{m-2} + n_{m-1} d_{m-2} - n_{m-2} d_{m-1} \end{pmatrix}}{(q_m n_{m-1} + n_{m-2})(q_m d_{m-1} + d_{m-2} - d_{m-1})}.$$

Then by factoring, we have that

$$\delta < \frac{(q_m d_{m-1} - d_{m-1} + d_{m-2})(q_m n_{m-1} + n_{m-2}) - (q_m n_{m-1} - n_{m-1} + n_{m-2})(q_m d_{m-1} + d_{m-2})}{(q_m n_{m-1} + n_{m-2})(q_m d_{m-1} + d_{m-2} - d_{m-1})}.$$

Since

$$\frac{(q_m d_{m-1} - d_{m-1} + d_{m-2})(q_m n_{m-1} + n_{m-2}) - (q_m n_{m-1} - n_{m-1} + n_{m-2})(q_m d_{m-1} + d_{m-2})}{(q_m n_{m-1} + n_{m-2})(q_m d_{m-1} + d_{m-2} - d_{m-1})} =$$

$$1 - \frac{(q_m n_{m-1} - n_{m-1} + n_{m-2})(q_m d_{m-1} + d_{m-2})}{(q_m d_{m-1} - d_{m-1} + d_{m-2})(q_m n_{m-1} + n_{m-2})},$$

we can further simplify, and substitute from equation (2.7), we have

$$\delta < 1 - \frac{\frac{(q_m - 1)n_{m-1} + n_{m-2}}{(q_m - 1)d_{m-1} + d_{m-2}}}{\frac{q_m n_{m-1} + n_{m-2}}{q_m d_{m-1} + d_{m-2}}}.$$

By Equation (3.2) and equation (2.7), we have

$$\delta = 1 - \frac{f'}{f} = 1 - \frac{f'}{\langle q_0, q_1, ..., q_m \rangle} = 1 - \frac{f'}{\frac{n_m}{d_m}} = 1 - \frac{f'}{\frac{q_m n_{m-1} + n_{m-2}}{q_m d_{m-1} + d_{m-2}}}.$$

Thus, we are able to get that

$$\langle q_0, q_1, ..., q_m - 1 \rangle < f'.$$

Case 4 $m \geq 3$ *and odd*

Suppose $\delta < \frac{1}{\frac{3}{2} n_m d_m}$. By (2.9), we have that since

$$d_m + d_{m-1} < \frac{3}{2} d_m,$$

we can simplify the above δ inequality to

$$\delta < \frac{1}{n_m(d_m + d_{m-1})}.$$

Since $\frac{1}{\frac{3}{2} d_m} < \frac{1}{d_m + d_{m-1}}$, we can use a similar analysis as in *Case 3*, resulting in

$$\delta < \frac{n_{m-2} d_{m-1} - n_{m-1} d_{m-2}}{(q_m n_{m-1} + n_{m-2})(q_m d_{m-1} + d_{m-2} + d_{m-1})},$$

Since m is odd, we get from Corollary 2.5 that

$$n_{m-2} d_{m-1} - n_{m-1} d_{m-2} = -(-1)^m = 1$$

and from Definition 2.3

$$n_m = q_m n_{m-1} + n_{m-2}, \text{ and}$$
$$d_m = q_m d_{m-1} + d_{m-2}$$

.

After a algebraic simplification similar to *Case 3*, we will achieve

$$\delta < 1 - \frac{\frac{(q_m+1)n_{m-1}+n_{m-2}}{(q_m+1)d_{m-1}+d_{m-2}}}{\frac{q_m n_{m-1}+d_{m-2}}{q_m d_{m-1}+d_{m-2}}}.$$

By using (3.2), we will get

$$\delta = 1 - \frac{f'}{\langle q_0, q_1, ..., q_m \rangle} < 1 - \frac{\frac{(q_m+1)n_{m-1}+m_{m-2}}{(q_m+1)d_{m-1}+d_{m-2}}}{\langle q_0, q_1, ..., q_m \rangle}.$$

With simplification, it leads to

$$\langle q_0, q_1, ..., q_m + 1 \rangle < f'.$$

After evaluating all four cases, we have sufficiently proven the proposition. □

Proposition 3.2. *Let m be even. Assume that $f' \neq f$, and that*

$$\langle q_0, q_1, ..., q_m - 1 \rangle < f' < \langle q_0, q_1, ..., q_m \rangle.$$

Then $q'_i = q_i$, for $0 \leq i < m$.

Proof. We will use induction on i. We will use $i = 0, 1$ for our base cases.

Base case $i = 0$

By the construction in (2.3), $q_0 = \lfloor f \rfloor$. Since $q_0 < \lfloor f' \rfloor < q_0$, we get that $q'_0 = \lfloor f' \rfloor = q_0$.

Base case 2 $i = 1$

By the previous case, since $q'_0 = q_0$, by 2.10 and $r'_0 = f' - q'_0$, we have that

$$\langle q_1, q_2, ..., q_m \rangle < \frac{1}{r'_0} < \langle q_1, q_2, ..., q_m - 1 \rangle.$$

Therefore, $q'_1 = \left\lfloor \frac{1}{r'_0} \right\rfloor = q_1$.

Case 1 i *odd,* $i > 2$

Assume true for $i-2$. Therefore, we have

$$q'_{i-2} = \left\lfloor \frac{1}{r'_{i-3}} \right\rfloor = q_{i-2}$$

and

$$\langle q_{i-2}, q_{i-1}, ..., q_m \rangle < \frac{1}{r'_{i-3}} < \langle q_{i-2}, q_{i-1}, ..., q_m - 1 \rangle.$$

We will show that $q'_i = q_i$. Since $q_{i-2} = q'_{i-2}$, we use Lemma 2.10 and (2.3) to obtain

$$\langle q_{i-1}, q_i, ..., q_m - 1 \rangle < \frac{1}{r'_{i-2}} < \langle q_{i-1}, q_i, ..., q_m \rangle.$$

By the above inequality and (2.3), we have that

$$q'_{i-1} = \left\lfloor \frac{1}{r'_{i-2}} \right\rfloor = q_{i-1},$$

which leads us to $q_{i-1} = q'_{i-1}$. Repeating the same argument using Lemma 2.10 and (2.3), we obtain

$$\langle q_i, q_{i+1}, ..., q_m \rangle < \frac{1}{r'_{i-1}} < \langle q_i, q_{i+1}, ..., q_m - 1 \rangle.$$

Again, by the above inequality and (2.3), we have that

$$q'_i = \left\lfloor \frac{1}{r'_{i-1}} \right\rfloor = q_i,$$

which satisifies this case.

Case 2 i *even,* $i > 2$, $i \neq m$

Assume true for $i-2$. Therefore, we have

$$q'_{i-2} = \left\lfloor \frac{1}{r'_{i-3}} \right\rfloor = q_{i-2}$$

and

$$\langle q_{i-2}, q_{i-1}, ..., q_m - 1\rangle < \frac{1}{r'_{i-3}} < \langle q_{i-2}, q_{i-1}, ..., q_m\rangle.$$

Using the same argument as the above case, we obtain

$$q'_{i-1} = \left\lfloor \frac{1}{r'_{i-2}} \right\rfloor = q_{i-1},$$

which satifies this case.

Through case by case analysis by induction, we have shown that for m even, $q_i = q'_i$ for $0 \leq i < m$. □

Now, we will prove that the algorithm works for the m^{th} case. Since f' and f are rational numbers, we know that the continued fractions expansion for them are unique [3]. We claim that the algorithm will find f in at the m^{th} iteration. First, we must prove that $q'_i = q_i$ for all $0 \leq i < m$.

Proposition 3.3. *Let m be odd. Assume that $f' \neq f$, and that*

$$\langle q_0, q_1, ..., q_m + 1\rangle < f' < \langle q_0, q_1, ..., q_m\rangle.$$

Then $q'_i = q_i$ for $0 \leq i < m$ where the q'_i are quotients for f'.

The proof for this proposition is similar to the proof of Proposition 3.2.

Theorem 3.4. *Let the following inequalities hold true:*

$$\langle q_0, q_1, ..., q_{m-1}, q_m - 1\rangle < f' < \langle q_0, q_1, ..., q_m\rangle, \qquad \text{if } m \text{ is even,}$$

$$\langle q_0, q_1, ..., q_{m-1}, q_m + 1\rangle < f' < \langle q_0, q_1, ..., q_m\rangle, \qquad \text{if } m \text{ is odd.}$$

Then the algorithm will find f at the m^{th} iteration.

Proof. We shall prove this by case analysis using induction.

Case 1 $m = 0$

From our hypothesis, we have

$$\langle q_0 - 1\rangle < f' < \langle q_0\rangle.$$

From (2.3), we generate q_0' for our first step in the algorithm. From the above inequality, we get that

$$q_0' = \lfloor f' \rfloor = q_0 - 1.$$

Since $i = 0$ is even, we add one to q_0', resulting in $q_0' + 1 = (q_0 - 1) + 1 = q_0$. The final step is to verify if the generated fraction is correct, which is immediate.

Case 2 $m = 1$

From our hypothesis, we assume

$$\langle q_0, q_1 + 1\rangle < f' < \langle q_0, q_1\rangle.$$

From Proposition 3.2 and Lemma 2.10, we obtain

$$\langle q_1\rangle < \frac{1}{r_0'} < \langle q_1 + 1\rangle.$$

By the inequality above, we get that

$$q_1' = \left\lfloor \frac{1}{r_0'} \right\rfloor = q_1.$$

Next, we generate an expansion, and since $i = 1$ is odd, we have $\langle q_0', q_1'\rangle = \langle q_0, q_1\rangle$, which is our desired fraction f.

Case 3 $m \geq 2$, m *even*

From our hypothesis, we assume

$$\langle q_0, q_1, ..., q_m - 1\rangle < f' < \langle q_0, q_1, ..., q_m\rangle.$$

From Proposition 3.2 and Proposition 3.3, we have that $q'_i = q_i$ for $0 \leq i < m$. By this fact, we may assume that we can skip to the m^{th} iteration of the algorithm. Assume that we have

$$\langle q_0, q_1, ..., q_{m-1} \rangle.$$

Our next step is to generate q'_m. Using Proposition 3.2 and Proposition 3.3 repeatedly, we have

$$\langle q_{m-1}, q_m \rangle < \frac{1}{r'_{m-2}} < \langle q_{m-1}, q_m - 1 \rangle.$$

By using Lemma 2.10, we obtain

$$\langle q_m \rangle > \frac{1}{r'_{m-2}} - q_m > \langle q_m - 1 \rangle.$$

From (2.3), we have that $\frac{1}{r'_{m-2}} - q_m = \frac{1}{r'_{m-1}}$, giving us

$$\langle q_m - 1 \rangle < \frac{1}{r'_{m-1}} < \langle q_m \rangle.$$

Therefore, we get

$$q'_m = \left\lfloor \frac{1}{r'_{m-1}} \right\rfloor = q_m - 1.$$

Proceeding to the next step of the algorithm, since m is even, we add one to q'_m, resulting in the expansion

$$\langle q'_0, q'_1, ..., q'_m + 1 \rangle = \langle q_1, q_2, ..., q_m \rangle = f.$$

Hence, we have found f for m even on the m^{th} iteration of the algorithm.

Case 4 $m \geq 3$, m *odd*

By a similar analysis from Case 3, we can obtain

$$q'_m = \left\lfloor \frac{1}{r'_{m-1}} \right\rfloor = q_m.$$

Proceeding to the next step of the algorithm, since m is odd, we get the expansion

$$\langle q'_0, q'_1, ..., q'_m \rangle = \langle q_1, q_2, ..., q_m \rangle = f.$$

Hence, we have found f for m odd. □

Theorem 3.5. *Suppose that we have f' and an oracle. If*

$$f' = f(1 - \delta), \quad 0 \leq \delta < \frac{1}{\frac{3}{2} n_m d_m},$$

where n_m and d_m is the numerator and denominator of f, then the continued fractions algorithm will find f from f'.

Proof. We begin by following the first step of the algorithm. We give f' to the oracle. If the oracle says that f' is not correct, then we can apply the results of Propositions 3.1, 3.2, 3.3, and Theorem 3.4 to find f from f'. □

4 RSA Encryption System Background

Below here lies a simplified version of the RSA encryption system. Suppose that Alice wants to receive a message from Bob, but wants to prevent Eve, a third party, from intercepting it. From R. L. Rivest, A. Shamir, and I. Adleman's RSA paper [4], Alice is able to do this by this encryption scheme. First, she picks two prime numbers, p, q, of approximately the same size. Ideally she would want these two primes to be large. The *modulus* $n = pq$ is the product of the two primes. Next, she needs to calculate $\phi(n)$, Euclid's Phi Function. Since $n = pq$, the product of two primes,

$$\phi(n) = (p - 1)(q - 1). \tag{4.1}$$

Alice would then need to pick a number e where $1 \leq e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$. The number e is called the *public exponent*. She would then need to calculate the *private* or *secret exponent* d where

$$ed \equiv 1 \pmod{\phi(n)}. \tag{4.2}$$

The numbers e, d are called modular inverses of each other. The number d can be calculated using modular arithmetic. From (4.2), we know that there exists an integer k such that

$$ed = k \cdot (p-1)(q-1) + 1. \tag{4.3}$$

From here, Alice will send Bob the numbers (n, e), which is called the *public key*. Alice will not send or reveal to anybody the following numbers: $p, q, \phi(n), d$. Theoretically, since p, q are large, $\phi(n)$ will be difficult to calculate, resulting in the prevention of d being calculated by third parties.

Encryption and decryption using RSA is described in the following simple model. Alice sends Bob the public key (n, e). Bob encrypts his message $M \in \mathbb{Z}_+$ into a ciphertext C by $C = M^e \pmod{n}$. Bob will send C back to Alice, where she will decrypt by

$$C^d = M^{ed} = M \pmod{n}.$$

We have this equality from the fact that n is square free, and e, d are modular inverses mod $\phi(n)$. Since Eve is unable to calculate d, she will be unable to decrypt the message that she intercepts between Alice and Bob. This is a simple, yet necessary background of the standard encryption and decryption using the RSA cryptosystem.

5 Continued Fractions Algorithm Applied to RSA

In this section, we will apply the continued fractions algorithm to RSA, showing that if our secret exponent, d, is a certain size, then the algorithm will find the numbers d, p, q. In order for us to be able to apply the continued fractions algorithm to RSA, we must first see the relationship between RSA and the algorithm. We see this by manipulations from the original RSA paper.

Let us start with (4.3). We will divide this equation through by dpq, resulting in

$$\frac{e}{pq} = \frac{k\left[(p-1)(q-1) + \frac{1}{k}\right]}{dpq}. \tag{5.1}$$

With some algebra, we get

$$\frac{e}{pq} = \frac{k}{d}\left(1 - \frac{p+q-1-\frac{1}{k}}{pq}\right). \tag{5.2}$$

If we set $\delta = \dfrac{p+q-1-\frac{1}{k}}{pq}$, we can rewrite (5.2) to

$$\frac{e}{pq} = \frac{k}{d}(1-\delta). \tag{5.3}$$

We must first justify that $\delta \geq 0$. Since k is a nonzero integer, $\frac{1}{k} \leq 1$. Also, since p, q are big, $p + q \geq 2 \geq 1 + \frac{1}{k}$. Finally, since p, q are positive integers, $pq > 0$, we have

$$\delta = \frac{p+q-1-\frac{1}{k}}{pq} \geq 0.$$

Second, we know that $\gcd(k, d) = 1$ since we get this directly from equation (4.3). Notice the similarities between (5.3) and (3.1). Let us define our $f' = \frac{e}{pq}$. Notice that f' is made entirely up with public information. Now that we have developed an f'

that is translated from RSA terms, we must prove that our δ is of proper size. That is, we will prove that if d is small, we will have $\delta < \frac{1}{\frac{3}{2}n_m d_m}$.

Lemma 5.1. *Suppose p, q are primes, and p, q are approximately the same size. Let n, p, q, e, d be as above. Let $d < \frac{1}{3}\sqrt[4]{n}$. Then $\delta < \frac{1}{\frac{3}{2}n_m d_m}$.*

Proof. First, we must clarify that we want the end result of the algorithm find the fraction where $n_m = k$ and $d_m = d$. We get this from the second observation above. We will prove this Lemma by contradiction. Suppose that $\delta > \frac{1}{\frac{3}{2}kd}$. From (5.3), we get

$$\frac{p + q - 1 - \frac{1}{k}}{pq} > \frac{1}{\frac{3}{2}kd}.$$

We know that

$$\frac{p+q}{pq} > \frac{p + q - 1 - \frac{1}{k}}{pq}.$$

Therefore, we have that

$$\frac{p+q}{pq} > \frac{2}{3kd}.$$

With further algebraic rearrangement, we have that

$$d > \frac{2pq}{3(p+q)k}.$$

Since p, q are approximately the same size, $n = pq \approx p^2$ and $p + q \approx 2p$. Therefore, $\frac{pq}{p+q} \approx \frac{p}{2} \approx \frac{\sqrt{n}}{2}$. Therefore, we have

$$d \gtrapprox \frac{\sqrt{n}}{3k}. \tag{5.4}$$

We have from (4.3) and the fact that $k \geq 1$ that $\phi(n)k < ed$. Since $e < \phi(n)$, we get

$$\phi(n)k < ed < \frac{1}{3}\phi(n)\sqrt[4]{n}.$$

Therefore, we get the result that $k < \frac{1}{3}\sqrt[4]{n}$. We can combine this result with (5.4) to get

$$d \gtrapprox \frac{\sqrt{n}}{\sqrt[4]{n}} = \sqrt[4]{n} \gg \frac{1}{3}\sqrt[4]{n},$$

contradicting the hypothesis that $d < \frac{1}{3}\sqrt[4]{n}$. Therefore, we have that if $d < \frac{1}{3}\sqrt[4]{n}$, then $\delta < \frac{1}{\frac{3}{2}kd}$. □

Now that we know what size d must be in order for the continued fractions algorithm to generate $\frac{k}{d}$, we must now reveal our oracle so that we are able to check whether our generated fractions is correct or not. From (5.3), we know that our desired product from the algorithm to be $\frac{k}{d}$. Therefore, we will denote our candidates that are generated from the algorithm to be $\frac{k'}{d'}$. Since $ed = k(p-1)(q-1)+1$ from (4.3), we can extract a guess for $\phi(n)$ from this equation. We will denote our guess as $\phi(n)'$, where

$$\phi(n)' = \frac{ed'-1}{k'}.$$

This is possible since we already know e and have generated d', k' from the algorithm. Since

$$(x-p)(x-q) = x^2 - px - qx + pq = x^2 - (n - \phi(n) + 1)x + n,$$

we can use our candidate $\phi(n)'$ in this equation. Substituting our candidate into the equation, resulting in

$$x^2 - (n - \phi(n)' + 1)x + n = 0.$$

We are able to find the roots to this equation since it is a quadratic polynomial. Suppose we get

$$x = p', \quad x = q',$$

as our roots. If $p', q' \notin \mathbb{Z}$, then we have not found our $\frac{k}{d}$, since our p, q are positive integers. Therefore, we move on to the next iteration of the algorithm. If we have $p', q' \in \mathbb{Z}$ and $p'q' = pq$, then we have found our $\frac{k}{d}$ since p, q are unique roots to the polynomial.

Theorem 5.2. *Suppose p, q are primes, and p, q are approximately the same size. Let $n = pq$ and let $1 \leq d, e < \phi(n)$ satisfy $ed \equiv 1 \pmod{\phi(n)}$. If $d < \frac{1}{3}\sqrt[4]{n}$, then d can be calculated from the continued fractions algorithm, thus p, q.*

Proof. We start off with the public key (n, e). Since $d < \frac{1}{3}\sqrt[4]{n}$, by Lemma 5.1, we have that $\delta < \frac{1}{\frac{3}{2}kd}$. Therefore, the fraction $\frac{k}{d}$ can be generated from the continued fractions algorithm. Using the oracle above that we have defined, and Theorem 3.5, we have shown that we can produce p, q from only public key information. □

6 Polynomial Time

We have sufficiently proven that our algorithm will discover a fraction f from f' when f' is an underestimate, from (3.1), and δ is of proper size. When this algorithm is applied to RSA where the secret exponent d is small, the algorithm leads to the discovery of d, p, q. In application to breaking codes, it is important to note that this discovery should take place in a length of time that is reasonable. For example, if RSA is used to protect credit card information, you would want to obtain the information before the credit card expires. Using a brute force attack against RSA to discover $\phi(n)$ might take a very long time. Fortunately, this algorithm runs in polynomial time, thus

allowing us to break an RSA key in polynomial time. The following proof is taken from [2].

Proposition 6.1. *The Euclidean Algorithm finds the continued fraction expansion for* $\frac{a}{b}$ *for* $a \geq b$.

Proof. Let

$$(c_{-1}, c_0) = (a, b)$$

$$(p_{-1}, p_0) = (1, 0)$$

$$(q_{-1}, q_0) = (0, 1).$$

We will set the counter $t = 1$. We will repeat the following steps until $c_t = 0$.

$$d_t = \left\lfloor \frac{c_{t-2}}{c_{t-1}} \right\rfloor$$

$$c_t = c_{t-2} - d_t c_{t-1}$$

$$p_t = p_{t-2} - d_t p_{t-2}$$

$$q_t = q_{t-2} - d_t q_{t-1}.$$

$$t = t + 1$$

When $c_t = 0$, the iteration stops. Then set $T = t$. The previous is the set up for the proof. We shall use induction on t.

Base case $\quad t = 0$

$$\beta = \frac{a}{b} = \frac{c_{-1}}{c_0}.$$

Induction step

Assume $\beta = \langle d_1, d_2, ..., d_{t-1}; \frac{c_{t-2}}{c_{t-1}} \rangle$. We will show that

$$\beta = \langle d_1, d_2, ..., d_{t-1}, d_r; \frac{c_{t-1}}{c_t} \rangle.$$

We will expand $\frac{1}{x}$ where $x = \frac{c_{t-2}}{c_{t-1}}$. Therefore, we have

$$\frac{1}{x} = \frac{1}{\left(\frac{c_{t-2}}{c_{t-1}}\right)} = \left(\frac{c_{t-2}}{c_{t-1}}\right)^{-1} = \left(\left\lfloor \frac{c_{t-2}}{c_{t-1}} \right\rfloor + \frac{c_{t-2}}{c_{t-1}} - \left\lfloor \frac{c_{t-2}}{c_{t-1}} \right\rfloor\right)^{-1}.$$

With rearrangement by common denominator, we have

$$\left(\left\lfloor \frac{c_{t-2}}{c_{t-1}} \right\rfloor + \frac{c_{t-2} - \left\lfloor \frac{c_{t-2}}{c_{t-1}} \right\rfloor c_{t-1}}{c_{t-1}}\right)^{-1} = \left(d_r + \frac{c_t}{c_{t-1}}\right)^{-1} = \left(d_r + \frac{1}{\left(\frac{c_{t-1}}{c_t}\right)}\right)^{-1}.$$

Therefore, we have that

$$\beta = \langle d_1, d_2, ..., d_{t-1}, d_r; \frac{c_{t-1}}{c_t} \rangle.$$

When $t = T$, then $c_T = 0$, which leaves

$$\left(d_T + \frac{c_T}{c_{T-1}}\right)^{-1} = {d_T}^{-1}.$$

This implies that

$$\beta = \langle d_1, d_2, ..., d_{T-1}, d_T \rangle = \frac{a}{b}.$$

Therefore, the Euclidean Algorithm is done in polynomial time. □

This results leaves us directly to our desired result.

Corollary 6.2. *The continued fraction expansion is found in polynomial time.*

Proof. Since the continued fraction expansion is found by the same algorithm as Euclidean Algorithm, it is found in the same time as the Euclidean Algorithm. By 6.1, since the Euclidean Algorithm takes polynomial time, implying that the continued fraction expansion is found in polynomial time. □

Hence, we are able to break short secret exponents RSA encryption using continued fractions in polynomial time.

7 Appendix

7.1 Examples

In this section, the continued fractions algorithm is going to be applied to a small RSA key pair to illustrate the operation. For this example, we give

$$pq = 41626477 \text{ and } e = 9247217.$$

We will show each step of the continued fractions algorithm applied to RSA for the public key, starting with

$$\frac{e}{pq} = \frac{9247217}{41626477} = \langle 0, 4, 1, 1, 164, 1, 1, 1, 1, 1, 7, 9, 1, 5, 1, 1, 3 \rangle$$

in Table 1. The attack on the public key (n, e) produces the following result:

$$d = 9, \quad p = 9719, \quad q = 4283, \quad k = 2, \quad g = 1.$$

For further verfication, Appendix A contains an RSA key generator that produces keys that can be placed into a python program in Appendix B.

Table 1: Example

Calculated Quantity	How it is Derived	$i = 0$	$i = 1$	$i = 2$
q'_i	See (2.3)	0	4	1
r'_i	See (2.3)	$\frac{9247217}{41626477}$	$\frac{9247217}{4637609}$	$\frac{4609608}{4637609}$
$\frac{n'_i}{d'_i} = \langle q'_0, q'_1, ..., q'_i \rangle$	See (2.7)	$\frac{0}{1}$	$\frac{1}{4}$	$\frac{1}{5}$
Guess of $\frac{k}{dg}$	From Section 3	$\frac{1}{1}$	$\frac{1}{4}$	$\frac{2}{9}$
Guess of $\phi(pq)$	From Section 3	9247217	36988868	41612476
Guess of p	From Section 3	21309240.778...	4637601.024...	9719
Guess of q	From Section 3	(quit)	(quit)	4283
d	$\frac{dg}{g}$			9

7.2 Appendix A

We have included a RSA key generator that can produce primes of length 10^n digits and give you the modulus, along with the public and secret exponents. This is written in Python programming language, using Sage version 4.3.1.

```
import random

def extEuclideanAlg(a, b) :

    """Computes a solution  to a x + b y = gcd(a,b), as well as gcd(a,b)

    """

    if b == 0 :

        return 1,0,a

    else :
```

```
        x, y, gcd = extEuclideanAlg(b, a % b)
        return y, x - y * (a // b),gcd

def modInvEuclid(a,m) :
    """Computes the modular multiplicative inverse of a modulo m,
    using the extended Euclidean algorithm
    """
    x,y,gcd = extEuclideanAlg(a,m)
    if gcd == 1 :
        return x % m
    else :
        return None

def RSAKeyGen(n):
    loopend = 0
    evaltest = 0
    while evaltest==0:
        while loopend ==0:
            x = random.randint(10^n, 10^(n+1))
            p = next_prime(x)
            y = random.randint(10^n, 10^(n+1))
            q = next_prime(y)
            m = p*q
```

```
        phi = (p-1)*(q-1)
        d_length = int(n/4)
        d = random.randint(10^(d_length), 10^(d_length + 1))
        if d < 1/3 * (m)^(1/4):
            if gcd(d,phi) == 1:
                loopend =1
    e = modInvEuclid(d, phi)
    if e== None:
        evaltest = 0
        loopend = 0
    if e != None:
        evaltest = 1
print "The primes are", n, "digits in length"
print "This is your p value:", p
print "This is your q value:", q
print "This is your modulus:", m
print "This is your secret exponent:", d
print "This is your public exponent:", e
```

7.3 Appendix B

We have included a program of the algorithm applied to RSA, written in Python programming language, using Sage version 4.3.1.

```
def SmallExponentAttack(n, e):
    """This algorithm is a modified version of Michael Wiener, Wade
    Trappe, and Lawrence Washington's attack on short secret exponents
    of RSA encryption system.  Michael Wiener's version of his attack
    is found in his paper "Cryptanalysis of Short RSA Secret Exponents.
    The paper can be found in the "IEEE Transaction on Information
    Theory", Volume 36, Number 3, May 1990, starting on page 553.
    Trappe and Washington's version can be found in their book
    "Introduction to Cryptography with Coding Theory", Second Edition,
    starting on page 170.

    In order for this program to find the private decryption key from
    the public encryption key, the private decryption key, d, must be
    below the specified size that this paper has discussed.  The input
    for this program is the standard form of the public key, (n, e),
    where n is the modulus and e is the public encryption exponent.

    This program will use a continued fractions algorithm starting off
    with the continued fractions expansion for the public key, where
```

```
'e' will be the numerator and 'n' will be the denominator.  We will
call this expansion 'x' in this program.  The program will set the
checker to 0.  This will allow us to check to see whether the
algorithm will actually find a plausible solution.

In accordance with Wiener's paper, this program will perform his
algorithm to attempt to find a suitable candidate for inspection to
determine whether the algorithm has found the private key.  The
program has already generated 'x' from the public key.  Next, it
will truncate the fraction 'x', starting from having only one
quotient value, 'q_0', to the entire fraction 'x'.  If the last
quotient in the expansion is on an even slot, then the program will
add one to its value.  Otherwise, it will keep the quotient value
as it is.  Then the program will proceed to determine whether the
fraction generated can lead to the private key.  Using Trappe and
Washington's method to verify the candidate for the factors of the
modulus, the program proposes a candidate.  If the candidate fails,
it will continue the loop with the next quotient slot.  If the
candidate succeeds though, it will print the results.

If the program progresses through the entire continued fractions
expansion 'x', and does not have a suitable candidate, then the
program will print 'No solution for this public key.' """
```

```
x = continued_fraction_list(e / n)

print "Our continued fraction is:"

print x

for i in range(len(x)):

    if i%2==0:

        y_1 = x[i] + 1

        y = x[0:i] + [y_1]

        a = convergent(y,i)

    else:

        y = x[0:i+1]

        a = convergent(y,i)

    k_1 = a.numerator()

    d = a.denominator()

    phi_1 = ((e*d)-1)/k_1

    p = quad1(n, phi_1)

    q = quad2(n, phi_1)

    if p * q == n:

        print "We have found the solution."

        print "This is our p value:", p

        print "This is our q value:", q

        print "This is our d value (private exponent):", d

        return

print "There is no solution for this public key."
```

```
def quad1(n, phi):
    """This is a quadratic equation solver that will solve for a
    root of a given candidate phi."""
    b = -(n - phi + 1)
    c = n
    sol1 = (-b + sqrt(b^2 - 4*c))/2
    if type(sol1) == sage.symbolic.expression.Expression:
        sol1 = 0
        return sol1
    else:
        return sol1
def quad2(n, phi):
    """This is a quadratic equation solver that will solve for a
    root of a given candidate phi."""
    b = -(n - phi + 1)
    c = n
    sol2 = (-b - sqrt(b^2 - 4*c))/2
    if type(sol2) == sage.symbolic.expression.Expression:
        sol2 = 0
        return sol2
    else:
        return sol2
```

7.4 Appendix C

In this Appendix, we have listed the keys and the lengths of primes that were generated through the key generator program in Appendix A. We will use these keys to run through our program to measure the average time to complete the algorithm. These results are covered in Appendix D.

The primes are 10 digits in length.

This is your p value: 46109347829

This is your q value: 78894873221

This is your modulus: 3637791151271946587209

This is your secret exponent: 433

This is your public exponent: 1024966559907452583537

The primes are 20 digits in length.

This is your p value: 934302731369511230053

This is your q value: 107725948742424571621

This is your modulus: 100648648149419240655445128876247306125913

This is your secret exponent: 386141

This is your public exponent: 631248419847318769612223068917832190446 61

The primes are 30 digits in length.

This is your p value: 988715887014265871804785617 6023

This is your q value: 688866299411294395388677334 3059

This is your modulus: 681093054256672794272883412349820782442073101584789356692749274357

This is your secret exponent: 85694311

This is your public exponent: 527911352083580554989147393232517383935996298750764793554704459

The primes are 40 digits in length.

This is your p value: 1320519173507050353352211532805690263164 3

This is your q value: 7948689433959410096741946540853036809698 3

This is your modulus: 1049639680179630412269363440709346936377581585696977095062265598901225822548633069

This is your secret exponent: 93140865639

This is your public exponent: 5070891686479206507055371772501236586319941319504641076195065253944263797583875 59

The primes are 50 digits in length.

This is your p value: 84445566740714532022466795046892908779931685787210 3

This is your q value: 9713710524872584726624752934963650793115835626921 63

This is your modulus: 8202797904281090407554894301225273942189644956487256484390759200931979778572444294320131474662144287 89

This is your secret exponent: 9288018680195

This is your public exponent: 6555602504738429484837506767693078356606995416345279715176118769422917742498584158570913353907882334 95

The primes are 60 digits in length.

This is your p value: 961956726675992223377761852340696309402617485043855921764196 1

This is your q value: 253340559673991279790596788079233163263344268888770932205808 1

This is your modulus: 243702655518256527065700371450551115382239281049266304695107973724496541443468136862103748306810518646692855959803047368419510797372449654144346813686210374830681051864669285595980304736841

This is your secret exponent: 7544190177959629

This is your public exponent: 239213501957777598969435227296158644504202933894537060977162145099578820411656166777385685291774252565751689328059670832691

The primes are 70 digits in length.

This is your p value: 3894464044500350985685201530602403451012222912974785298943333489961075 9

This is your q value: 9768155619333218166519620030190351449467702360424848708172507591529204 7

This is your modulus: 3804173084057727569677808076248609342224052854541750114949436170166832714603017552802578072687522212536345058364909473395946220383196108333673

This is your secret exponent: 328912203630965819

This is your public exponent: 1521552551702992527495266329483716994687439002027633768029214345378073164880622283877536278623939931646219284033044951855488

75735320058015427

The primes are 80 digits in length.

This is your p value: 731504297766575890837163585969362997688513115017022018813432830869302769384572709

This is your q value: 505899803623545108592382564093673871734616550110316882457458165938529893648493903

This is your modulus: 370067880589890009945724677419436222670052028605023623276336927105390242750787562710300040952654864779531586753476473622123292006974459222098721235400601046693227

This is your secret exponent: 300070249244721159241

This is your public exponent: 306915486932639720907352564125370593831473674435122781002211261458250332747007481236340983837270813915591405935336736147423511498506544920619320405128548025557377

The primes are 90 digits in length.

This is your p value: 231990689166172703932876356774097314014043327776569742810800713747458537339539914572150189 9

This is your q value: 156070942579247553259525515034561971718302230751615462060451167798299074367891355818931342 7

This is your modulus: 362070055277738075159877699888475021157853529605186960402279233193283220396911790112044152303589898289187373789855292136184018148136059013545055471992190306742392666325655408669787 3

This is your secret exponent: 41075147631741792008819

This is your public exponent: 360246745641354207826249187615109770636490137769651099812346409462823197240346627418694741635139580741601863364362356437911708261055333322652392524623028390646186911322174103464333108261055333322652392524623028390646186911322174103464333

The primes are 100 digits in length.

This is your p value: 74213394744938727254106462031659198278301080189827176834371974416507473490047085704054823204670209509

This is your q value: 72653620388150074022408737100172857883354381920328470240223597812542716198113865385647449531516471197

This is your modulus: 53918718095147098769057008932905172676610790019847167265035964231429177604811063692445684361113372070364657249704723713753030179129832532695155589177941784208789496175212000785937241541318937435401227393249783838985422793219691273419053881229696159686114792847230980155744579074578387587639055669776591124891335037381045993804947755451018627786126155643828566276063212001826426458089811783771167955

The primes are 110 digits in length.

This is your p value: 81625379566176579642988609853817701576320122790007308985095720123231171981359348972834685955132973653740150499

This is your q value: 10845665897205187390543122306353720151145679826945936557

979705427374542820284623906312837264281204089522126153

This is your modulus: 885281595507310483149341670114040411033215530149797015567963816926687075229380159298738634467295147701237434568508623441388717832884140631071259917404009338055453989646246887140547760554370712782695651635625047954053819291

This is your secret exponent: 49399233031534785500977497 13

This is your public exponent: 28000332343626775581142773497022695557150287607654782417176348509476114295526604248821411018318042727318030349637436643462800156652598053082563487984081815973517960156454160852413565996042975105634595669737597960330161201

The primes are 120 digits in length.

This is your p value: 948486960217796347428690297066244461578007072003558765889783904205623477336464986251661303656656142711681102421809729903 3

This is your q value: 215438702759096176075523147595011595562712478227647199029621324617602720987743072161210428834704923868534240168308945135 1

This is your modulus: 204340800293240506949204130328304877140882954146602147761625108715625953359534277481903089139217227517429789729329314995294668577128320224371594036375559139979466262466173806572422893205037313194475837224560877860227537467796559945869528435 83

This is your secret exponent: 352074953384479729761249056134 1

This is your public exponent: 189319798942958662295566647537723468425518165836127589924943030889155761638972127600929284322949914330454385972574244196002

522148394053080639465103414058881710877052783707294153240644173185100291850251284855830867581699477997675216946773638610

The primes are 130 digits in length.

This is your p value: 902518748564973192681476891743907861539751453118196285424109190885881119699232736841772084957376442549415425479394634725956968010790

This is your q value: 48275181594904633785826239048131705860405956550432941832820614883542932814224724751768495355606339417126295453967301425596879566101This is your modulus: 435692564797801567367337399985061519241829574607383017102964135838987688868712597584659026632322266692690804970113439083179867369242972883779996120873679597414808329023724932105715101326870425426013581073322965890114292810980961619692610747631644840278722862297900

This is your secret exponent: 1275893872685979513151238061965670

This is your public exponent: 392796492792404598625645379497581060472668689252111387050830442212778652813805715225110950303126454512327147595672538055472409970972441409631178265619770265428516265804113842010472029861004778004523869788266096735844978921662665882492163770795487761063698162850300

The primes are 140 digits in length.

This is your p value: 269524175642152041718316313178781811745837431289721261882130363269113601825563070681699032578944469264934165441604097160410337132600651105390

This is your q value: 3703344228802432062976077861702514746723270970973216609190181927135774837904994858949891380330404515440581294547683817279909858087035362304291

This is your modulus: 9981408003870967974784007382803453112568591102279749149360096226250605154130243956065916202846218911303030537946179198108284780292217959838908417474964496650527167002815014011637561189160398441693228160670670277651147726484377663170713351207701656528836381066459091092370926039123

This is your secret exponent: 728971783812315646258642582386548539

This is your public exponent: 3247008936895502688948650241553489493737127293056839036338819039699391363826053945058168049002682730724286710145371448629281606935992980258017330606825210439100100669245713212728011150047910899717407130530566662977491478251255291816762313687966981568466198156285065523009349763645

The primes are 150 digits in length.

This is your p value: 4627799369605690488963715075213405137653165911202661942416279411591935094834122254874962237416350986431628503217752026727278481348234379832542667940257

This is your q value: 3453534427587690594843870519266576731181413666175104382255997350819418440643782234340908933075982325154106000255137185987142033768536765982658031612643

This is your modulus: 1598226444690186368308431081239831565857086899418846304

00307468812277821563487125316938614081832344545791392080231367996230014516985571957957143455954353875192910782049901011026388876350927979162761857018388571646228742334595495932365444157985810381123128374622780652240929271341710140132069688898692 51

This is your secret exponent: 746325024435713913178950005067994054333

This is your public exponent: 47177620228397800111320179445005744279004214428703717330702458123016945627335210706553527817545507549530818880157427811926023250377500045129141485914311650573500951432295186588681128576636385050781806017486312747307649951768524052358827361154209611234447997763098595112517084061380115992754642300353 37

7.5 Appendix D

In this Appendix, we cover the results of time test of the keys from Appendix C. These tests were run on an iMac, Mac OS X 10.6.2, 3.06 GHz Intel Core 2 Duo, 4 GB 1067 MHz Ram, on Sage v.4.3.1. Each key was run through 5 loops.

Table 2: Time Chart for Algorithm Completion

Primes	Modulus	Public Exponent	Average Time(s)
10	22	22	0.120
20	42	41	0.193
30	62	62	0.351
40	82	81	0.572
50	102	102	0.847
60	122	122	0.925
70	142	141	1.14
80	162	162	1.56
90	181	181	2.02
100	202	202	2.08
110	221	221	2.58
120	242	242	2.90
130	262	262	3.27
140	281	281	3.66
150	302	301	4.27

Putting the length of the primes versus the average in seconds in the graph, we

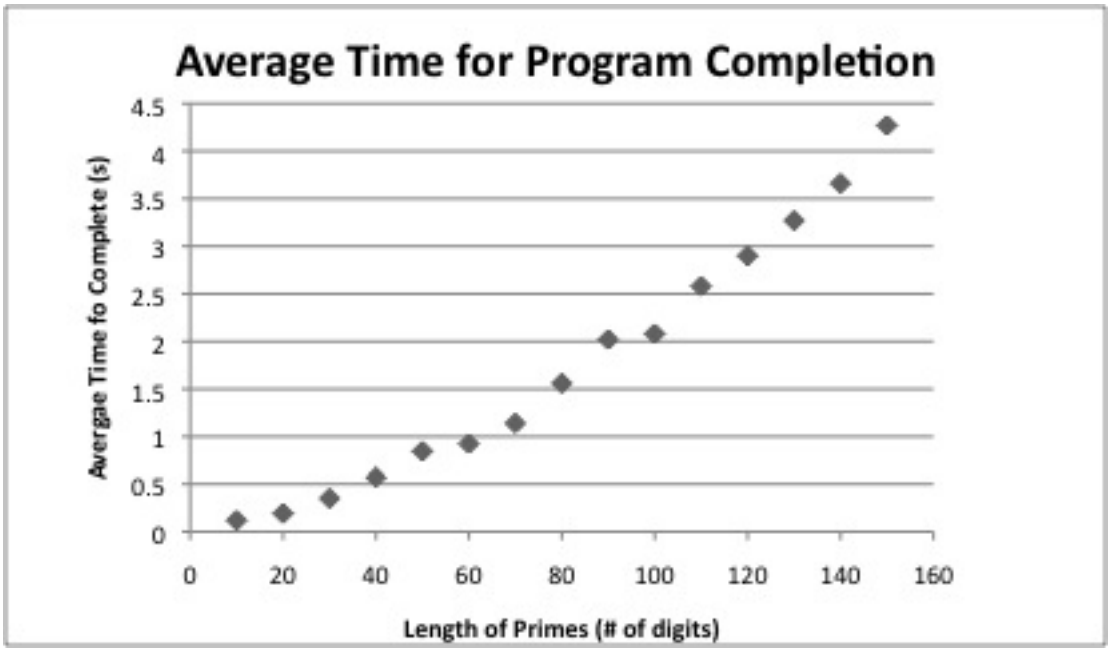

Figure 1: Length of Primes versus Average Time of Completion in Seconds

can see visual evidence that the algorithm runs in polynomial time.

References

[1] Dan Boneh, *Twenty years of attacks on the rsa cryptosystem*, Notices of the AMS **46** (1999), no. 2, 203–213.

[2] George Nemhauser and Laurence Wolsey, *Integer and combinatorial optimazation*, Wiley-Interscience Series in Discrete Mathematics and Optimization, 1999.

[3] C. D. Olds, *Continued fractions*, The Mathematical Association of America, 1963.

[4] R. L. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM **21** (1978), no. 2, 120–126.

[5] Michael J. Wiener, *Cryptanalysis of short rsa secret exponents*, IEEE Transaction on Information Theory **36** (1990), no. 3, 553–558.