Factoring a semiprime n by estimating $\phi(n)$

Kyle Kloster

May 7, 2010

Abstract

A factoring algorithm, called the Phi-Finder algorithm, is presented that factors a product of two primes, $n = pq$, by determining $\phi(n)$. The algorithm uses precisely $(\sqrt{p} - \sqrt{q})^2 / \lfloor \log_2(pq) \rfloor$ iterations. It is demonstrated that if $|p - q| \leq \sqrt{n}2^{-k}$ then n can be factored in less than $\sqrt{n}2^{-2(k+1)} / \lfloor \log_2(pq) \rfloor$ iterations. Furthermore, the algorithm can quickly factor some semiprimes that are constructed according to current RSA standards, though it is shown to be sufficiently improbable that such semiprimes would be constructed at random.

Phi-Finder is shown to be strictly faster than Fermat Factorization and trial division, and faster than the Quadratic Sieve and Coppersmith's known-bit attack under the condition that $|p-q|$ is small. The possibility of generalizing Phi-Finder to factor n when p/q is very near some obvious ratio is discussed.

A condition for an early successful termination is discussed, namely if $\operatorname{ord}_n 2 < (\sqrt{p} - \sqrt{q})^2$. Experimental data is presented that strongly suggests the condition is highly improbable.

1 Introduction

The RSA encryption system operates with a modulus n which is the product of two distinct primes p and q. Such integers are called "semiprimes." Factoring n and computing $\phi(n)$ are equivalent in the following sense: computing $\phi(n)$ can be done efficiently if we factor $n = pq$, since $\phi(n) = (p - 1)(q - 1)$, and, conversely, if $\phi(n)$ is known and it is known that $n = pq$ is a product of two primes, then p and q are the roots of the quadratic equation

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

Although, computing $\phi(n)$ and factoring n are equivalent, most number-theoretic attacks on RSA attempt to factor n rather than directly compute $\phi(n)$ [14] [2] [5] [13] [9] [16]. If either n is factored or $\phi(n)$ is known, it is trivial to derive the private key from the public key, resulting in the possibility of private messages being read, digital signatures being forged, etc. Breaking the RSA encryption system is conjectured to be as difficult as factoring the modulus, $n = pq$ [11] [17]. Hence, to ensure that an RSA modulus n is secure it is necessary to make n difficult to factor.

In this thesis is presented a method of estimating $\phi(n)$ and an algorithm for computing $\phi(n)$ without factoring n that uses $(\sqrt{p} - \sqrt{q})^2/\lfloor \log_2(pq) \rfloor$ iterations. Once $\phi(n)$ is computed, n can be factored efficiently using the above quadratic equation. It is shown that if the prime factors of an RSA modulus are chosen such that they are "too close together" the algorithm presented below in Figure 3.3 can compute $\phi(n)$ easily (and thus factor n). The notion of "too close together" is made precise in Theorem 10.

Because it computes $\phi(n)$, the algorithm is referred to here as the Phi-Finder algorithm, given in Figure 3.3. It is possible, however, that the Phi-Finder algorithm computes a multiple of the order of 2 modulo n that is not $\phi(n)$. This exception is discussed in detail in Section 3.2, and it is shown to be improbable.

It is shown that Phi-Finder is faster, in the case of factoring integers with the structure of an RSA modulus, than Fermat factorization, another method that works efficiently when $|p - q|$ is small, as well as the Quadratic Sieve (QS) and General Number Field Sieve (GNFS), which are general purpose algorithms incapable of taking advantage of the special structure of n.

Phi-Finder is also compared to Coppersmith's known-bit attack in which knowledge of $(\log_2 n)/4$ of the MSBs of either prime provides an efficient factorization of n [3]. The possibility is discussed of a generalization of Phi-Finder that would enable the algorithm to factor n given knowledge of some of the bits of $r = p/q$. Runtimes of Phi-Finder, Fermat factorization, and the QS are given in Tables 3, 2, and 1.

Although Phi-Finder was independently developed as presented in this paper, some of the essential ideas behind the algorithm were explored in 2005 in the doctoral dissertation of Yousef Bani Hammad [4]. Hammad named his algorithm the RAK algorithm after the city of Ras al-Khaimah in the United Arab Emirates.

2 Background

Integer factorization methods for factoring the product of two primes can be divided into two categories: general-purpose algorithms, which have running times that depend on only the size of n, and special-purpose algorithms, which have running times that depend on some special structure of n (e.g. the size of the smallest prime divisor of n), and depend only marginally on the size of n.

Generally speaking, a special-purpose method suceeds much more quickly than general-purpose methods if n has a special structure, e.g. if $|p - q|$ or the smallest prime divisor of n is small. If n has no such form that can be exploited by known algorithms, then general-purpose methods are asymptotically faster.

2.1 General-Purpose Factoring Methods

Currently, the fastest useable general-purpose factoring algorithms are the General Number Field Sieve (GNFS) and the Multiple Polynomial Quadratic Sieve (MPQS) [2] [10] [14]. Peter Shor's algorithm for quantum computers can factor integers in polynomial time, but it requires a quantum computer that is far larger than can currently be built [12].

Since the runtimes of general-purpose algorithms depend entirely on the size of n, an RSA modulus that is chosen to be sufficiently large enough cannot be factored by such algorithms in a reasonable time. For example, in the most recent factoring record, set in December 2009, the 768-bit RSA challenge number took approximately 1500 processor years using the GNFS [15]. The authors estimated that the factorization of a 1024-bit modulus using the GNFS would be "about a thousand times harder" [15].

Thus, it is easy to choose primes that are large enough to guard against these general-purpose factoring methods, so choosing an RSA modulus that is difficult to factor is then a matter of selecting its prime factors so that they have no special structure that would make the modulus easy to factor.

2.2 Special-Purpose Factoring Methods

The parameters that determine the runtime of special-purpose methods vary from method to method. For example, the runtime of a factoring method like the Elliptic Curve Method (ECM) or Pollard's Rho Method depends mostly on the size of the smallest prime factor of n [5] [8].

On the other hand, Pollard's $p-1$ and Williams' $p+1$ factoring methods have runtimes that depend mostly on whether or not a prime factor p of n has the property that $p-1$ or $p+1$ is smooth [9] [16]. But because these particular factoring algorithms work efficiently only when n has some special form, RSA moduli can be constructed so that they are secure against algorithms like these.

Constructing an RSA modulus so that it has no small prime divisor avoids the threat posed by ECM and Pollard's Rho. Similarly, constructing $n = pq$ by choosing p and q so that $p \pm 1$ and $q \pm 1$ each have at least one large prime divisor guards against the $p \pm 1$ factoring methods. Frequently, a prime p is chosen so that $(p-1)/2$ is also prime. Primes that have this property are called *safeprimes.*

In general, the invention of a special-purpose factoring algorithm further restricts the selection of primes that can be used to build an RSA modulus that is safe from known factoring attacks. Below, an algorithm is presented that finds $\phi(n)$, and so can factor n, using $(\sqrt{p}-\sqrt{q})^2/\lfloor \log_2(pq) \rfloor$ steps. Thus, if p and q are too close together then $(\sqrt{p}-\sqrt{q})^2$ will be small, and so n can be easily factored. The notion of p and q being "too close together" is made precise in Theorem 10. First, the Phi-Finder algorithm is described.

3 The Phi-Finder Algorithm

The motivation for factoring an RSA modulus n is that a factorization of n allows us to calculate $\phi(n)$ easily, and then the private key can be computed. Suppose that instead of factoring n, we try to guess $\phi(n)$. Note that we know something about the structure of any RSA modulus, that it is always the product of exactly two primes, so we know that $\phi(n)$ has the form $\phi(n) = pq - p - q + 1$.

The National Institute of Standards and Technology (NIST) maintains a set of standards called the Digital Signature Standard (DSS) for the generation of secure parameters for cryptographic systems, including standards for generating RSA moduli [7]. We therefore have more knowledge about the form of any RSA modulus constructed according to the standards in the DSS.

We know that usually the two primes chosen are safeprimes, i.e. $p = 2u + 1, q = 2v + 1$ for u and v primes. If p and q are not safeprimes, then the DSS requires that $p - 1$ and $q - 1$ each have at least one prime factor that is greater than 2^{100}, and in some cases even larger [7]. We also know that p and q are somewhat close together: if we let r denote the ratio of the prime factors of n so that $r = p/q$, where p is the larger prime, then we know $1 < r < \sqrt{2}$ is true [7]. Now we will express $\phi(n)$ in a form that allows us to take advantage of this information.

3.1 Estimating $\phi(n)$

Lemma 1. *Let $n = pq$ and $r = p/q$. Then $\phi(n) = n - \sqrt{n}\left(\frac{r+1}{\sqrt{r}}\right) + 1$.*

Proof. Since $r = p/q$, we have $p = rq$ and so $n = pq = rq^2$. Thus we can write $q = \sqrt{n/r}$ and $p = \sqrt{nr}$. Now we can express

$$\begin{aligned}\phi(n) &= pq - p - q + 1 \\ &= n - \sqrt{nr} - \sqrt{n/r} + 1 \\ &= n - \sqrt{n}\left(\sqrt{r} + 1/\sqrt{r}\right) + 1 \\ &= n - \sqrt{n}\left((r+1)/\sqrt{r}\right) + 1\end{aligned}$$

□

Since we know that $1 < r < \sqrt{2}$, we can "guess" the value of r as g and then estimate $\phi(n)$ using the above formula.

Definition 2. *Let $n = pq$ and $r = p/q$. Let g be our guess of the value r. Define $\hat{\phi}_g(n)$ as*

$$\hat{\phi}_g(n) = n - \sqrt{n}\,((g+1)/\sqrt{g}) + 1.$$

Thus, $\hat{\phi}_g(n)$ provides an estimate of $\phi(n)$. Now we would like to know the error term in this estimation.

Theorem 3. *Let $n = pq$, a product of distinct primes, and let $\hat{\phi}_g(n)$ be defined as above. Then*

$$\begin{aligned} \hat{\phi}_g(n) - \phi(n) &= q(\sqrt{r} - \sqrt{g})(\sqrt{r} - 1/\sqrt{g}) \qquad (1) \\ &= (\sqrt{p} - \sqrt{q}\sqrt{g})(\sqrt{p} - \sqrt{q}/\sqrt{g}) \qquad (2) \end{aligned}$$

Furthermore, if $1/r < g < r$, then $\hat{\phi}_g(n) > \phi(n)$, i.e. $\hat{\phi}_g(n)$ is an overestimate.

Proof. From Definition 2 and from Lemma 1 we have

$$\begin{aligned} \hat{\phi}_g(n) - \phi(n) &= n - \left(\sqrt{n}(g+1)\right)/\sqrt{g} + 1 - \left(n - \left(\sqrt{n}(r+1)\right)/\sqrt{r} + 1\right) \qquad (3) \\ &= \sqrt{n}\left((r+1)/\sqrt{r}\right) - \sqrt{n}\left((g+1)/\sqrt{g}\right) \qquad (4) \\ &= \sqrt{n}\left(\sqrt{r} + 1/\sqrt{r} - \sqrt{g} - 1/\sqrt{g}\right) \qquad (5) \\ &= \sqrt{n}(\sqrt{r} - \sqrt{g})(1 - 1/\sqrt{rg}) \qquad (6) \\ &= q\sqrt{r}(\sqrt{r} - \sqrt{g})(1 - 1/\sqrt{rg}) \qquad (7) \\ &= q(\sqrt{r} - \sqrt{g})(\sqrt{r} - 1/\sqrt{g}) \qquad (8) \\ &= \sqrt{q}(\sqrt{r} - \sqrt{g})\sqrt{q}(\sqrt{r} - 1/\sqrt{g}) \qquad (9) \\ &= (\sqrt{p} - \sqrt{q}\sqrt{g})(\sqrt{p} - \sqrt{q}/\sqrt{g}) \qquad (10) \end{aligned}$$

Equations (8) and (10) above prove equations (1) and (2) of the theorem. Observe from (8) that if $0 < g < r$, then $(\sqrt{r} - \sqrt{g})$ is positive, and if $g > 1/r > 0$ then $1 > 1/rg$ and so $(1 - 1/\sqrt{rg})$ is positive. Thus, as long as $1/r < g < r$, we have $\hat{\phi}_g(n) - \phi(n) > 0$ and so $\hat{\phi}_g(n)$ will be an overestimate. □

The fact that $\hat{\phi}_g(n)$ is an overestimate is important because it guarantees that if we decrement from $\hat{\phi}_g(n)$, we are guaranteed to reach $\phi(n)$ eventually. In other words, decrementing from $\hat{\phi}_g(n)$ will successfully terminate by copmuting $\phi(n)$. It is simple to ensure that $1/r < g < r$ since we know from the DSS that $1 < r < \sqrt{2}$ is true [7].

If, in addition, we know enough of the most significant bits (MSBs) of either p or q then Phi-Finder provides a known-bit attack because we can compute the same number of MSBs of r and use the above formula for estimating $\phi(n)$. This possibility is discussed more in Section 4. On the other hand, if we continue with our above assumption that p and q are close together, we can conclude that $p/q = r$ is probably very near 1. Thus, the particular estimate $g = 1$, or $\hat{\phi}_1(n)$, seems especially important.

Corollary 4. *If we set $g = 1$, i.e. it is assumed that $r = 1 + \epsilon$, then*

$$\begin{aligned} \hat{\phi}_1(n) - \phi(n) &= (\sqrt{p} - \sqrt{q})^2 > 0 \qquad (11) \\ &= q(\sqrt{r} - 1)^2. \qquad (12) \end{aligned}$$

Proof. Equation (11) follows from plugging $g = 1$ into equation (2) from Theorem 3. Equation (12) follows from plugging $g = 1$ into equation (1) from Theorem 3. □

Lemma 5. *If* $1 < r < \sqrt{2}$ *then* $\phi(n) < \hat{\phi}_g(n) < 2\phi(n)$.

Proof. We already know $\phi(n) < \hat{\phi}_g(n)$ as long as $1/r < g < r$. It suffices to show that $\hat{\phi}_g(n) < 2\phi(n)$, i.e. $\hat{\phi}_g(n) - 2\phi(n) < 0$.

$$\begin{aligned}
\hat{\phi}_g(n) - 2\phi(n) &= n - \frac{\sqrt{n}(g+1)}{\sqrt{g}} + 1 - 2\left(n - \frac{\sqrt{n}(r+1)}{\sqrt{r}} + 1\right) \\
&= \sqrt{n}\left((2r+2)/\sqrt{r} - (g+1)/\sqrt{g}\right) - n - 1 \\
&= \sqrt{n}\left((2r+2)/\sqrt{r} - (g+1)/\sqrt{g} - \sqrt{n}\right) - 1
\end{aligned}$$

But $(2r+2)/\sqrt{r} - (g+1)/\sqrt{g} - \sqrt{n}$ must be negative: $(2r+2)/\sqrt{r} - (g+1)/\sqrt{g}$ is at most $\sqrt{2}$, when $r = \sqrt{2}$ and $g = 1$, and $\sqrt{n}$ is several hundred digits, so their difference must be negative. Thus, $\hat{\phi}_g(n) - 2\phi(n) < 0$ and so $\hat{\phi}_g(n) < 2\phi(n)$. □

Hence, as long as $1/r < g < r$ we always have $2\phi(n) > \hat{\phi}_g(n) > \phi(n)$. From the proof given it seems likely that the bounds could be made more tight, but this result is all that is needed in this thesis. This guarantees that if we denote our estimate $\phi(n)$ as $\hat{\phi} = \hat{\phi}_g(n) = n - 2\sqrt{n} + 1$, and naively check all integers $\hat{\phi} - 1, \hat{\phi} - 2, \hat{\phi} - 3, ...$, then we will eventually reach the true value of $\phi(n)$, and not a multiple of $\phi(n)$. Not only that, Theorem 3 tells us that the number of integers that we would have to go through is exactly $\left(\sqrt{p} - \sqrt{qg}\right)\left(\sqrt{p} - \sqrt{q/g}\right)$. This process would be inefficient, but at least we are now certain that it will terminate successfully.

Observe that we can test

$$2^{\hat{\phi} - x} \equiv 1 \bmod n$$

to check a given estimate $\hat{\phi} - x$. This is because if $\hat{\phi} - x$ is equal to the true value $\phi(n)$, then $2^{\hat{\phi} - x}$ will be congruent to 1 because the multiplicative group $\mathbb{Z}/n\mathbb{Z}^\times$ has order $\phi(n)$. Note that this introduces the issue of whether $\hat{\phi} - x$ is actually $\phi(n)$ or some other multiple of $\text{ord}_n 2$ inside the interval $(\hat{\phi}, \phi(n))$.

Now we want to address the problem of whether we compute $\text{ord}_n 2$ or $\phi(n)$, and we would like a method of reaching $\phi(n)$ that is more efficient than checking via expensive modular exponentiations whether or not $2^{\hat{\phi} - x} \equiv 1 \mod n$ for each possible x.

3.2 Comparing $\text{ord}_n 2$ and $\phi(n)$

In this section it is shown that it is highly improbable that Phi-Finder computes a multiple of $\text{ord}_n 2$ other than $\phi(n)$, and in some cases it is even impossible. First, it is shown that it is impossible when n is the product of two safeprimes.

Theorem 6. *Let* $n = pq$ *and* $r = p/q$ *with* $p > q$*, where* p *and* q *are safeprimes. If* $1 < r < \frac{23}{8}$ *and* $16 < n$*, then no integer multiple of* $\text{ord}_n 2$ *exists inside the interval* $(\hat{\phi}_1(n), \phi(n))$.

Proof. Since p and q are safeprimes, there exist primes a and b so that $a = (p-1)/2, b = (q-1)/2$, and so $\phi(n) = (p-1)(q-1) = 4ab$. Since $\operatorname{ord}_n 2$ must divide $\phi(n)$ and $\operatorname{ord}_n 2$ is positive by definition, the set of possible values for $\operatorname{ord}_n 2$ is then $\{b, 2b, 4b, a, 2a, 4a, ab, 2ab, 4ab\}$. The numbers 1, 2, and 4 are omitted from the set since clearly $2 \neq 1$, $4 \neq 1$, and $16 \neq 1 \mod n$, because $n > 16$. Finally, since $2a+1 = p > q = 2b+1$, a is greater than b, and so b is the smallest element of the set.

We want to show that no integer multiple of $\operatorname{ord}_n 2$ exists in $(\hat{\phi}_1(n), \phi(n))$. Since $\phi(n)$ is an integer multiple of $\operatorname{ord}_n 2$, i.e. $\phi(n) = k \operatorname{ord}_n 2$ for some integer k, it suffices to show that $\phi(n) + \operatorname{ord}_n 2 = (k+1)\operatorname{ord}_n 2 > \hat{\phi}_1(n)$. Then we will have that

$$(k+1)\operatorname{ord}_n 2 > \hat{\phi}_1(n) > \phi(n) = k \operatorname{ord}_n 2.$$

Now since b is the smallest possible value of $\operatorname{ord}_n 2$, $(4a+1)b$ is the smallest possible multiple of $\operatorname{ord}_n 2$ that is greater than $\phi(n) = 4ab$. Thus, it suffices to show that $(4a+1)b > \hat{\phi}_1(n)$.

Recall that $\hat{\phi}_1(n) = n - 2\sqrt{n} + 1$ and $n = pq = q^2 r$. We also have $\sqrt{n} = \sqrt{q^2 r} = q\sqrt{r}$, and so we can write $\hat{\phi}_1(n) = q^2 r - 2q\sqrt{r} + 1$. Also, since $a = (p-1)/2$ and $b = (q-1)/2$, we have that

$$(4a+1)b = (2(p-1)+1)(q-1)/2 = (2qr-2+1)(q-1)/2 = (2qr-1)(q-1)/2.$$

Thus, the theorem holds, i.e. $(4a+1)b > \hat{\phi}_1(n)$ is true, when we have the following:

$$\begin{aligned}
(2qr-1)(q-1)/2 &> q^2 r - 2q\sqrt{r} + 1 \\
2q^2 r - q - 2qr + 1 &> 2q^2 r - 4q\sqrt{r} + 2 \\
-q - 2qr &> -4q\sqrt{r} + 1 \\
0 &> 2qr - 4q\sqrt{r} + 1 + q \\
0 &> 2r - 4\sqrt{r} + 1 + 1/q
\end{aligned}$$

Note that this is a quadratic equation in $\sqrt{r}$. Solving it for $\sqrt{r}$ yields

$$\sqrt{r} = \frac{4 \pm \sqrt{16 - 4(2)(1+1/q)}}{4} = 1 \pm \frac{\sqrt{4 - 2(1+1/q)}}{2} = 1 \pm \frac{1}{2}\sqrt{2 - 2/q}$$

and so we have

$$\begin{aligned}
1 - (1/2)\sqrt{2-2/q} &< \sqrt{r} < & 1 + (1/2)\sqrt{2-2/q} \\
1 - \sqrt{2-2/q} + 1/4\,(2-2/q) &< r < & 1 + \sqrt{2-2/q} + 1/4\,(2-2/q) \\
3/2 - \sqrt{2-2/q} - 1/(2q) &< r < & 3/2 + \sqrt{2-2/q} - 1/(2q)
\end{aligned}$$

Now since q is enormous, $1/q$ is negligible, and so $\sqrt{2-(2/q)} - 1/(2q) \approx \sqrt{2}$, hence the left-most part of the inequality is very near $3/2 - \sqrt{(2)} < 1$. Thus, since $r > 1$ the left side of the inequality is always satisfied. Similarly, since $\sqrt{2-(2/q)} - 1/(2q) \approx \sqrt{2}$, the right-most part of the inequality is very near

$3/2 + \sqrt{2} > 23/8$. Thus, when $r < 23/8$ the above inequalities hold, and so $(4a + 1)b > \hat{\phi}_1(n)$ holds. Thus, there is no integer multiple of $\mathrm{ord}_n 2$ in the interval $(\hat{\phi}_1(n), \phi(n))$ when $1 < r < 23/8$ and n is a product of two large safeprimes. $\square$

The case when p and q are not safeprimes: Since n is not always the product of two safeprimes, we want to analyze the size of the order of 2 in the case that n is not the product of two safeprimes. Recall from the proof of Theorem 6 that no integer multiple of $\mathrm{ord}_n 2$ exists in the interval $(\hat{\phi}_1(n), \phi(n))$ when

$$\phi(n) + \mathrm{ord}_n 2 > \hat{\phi}_1(n)$$

i.e. when

$$\mathrm{ord}_n 2 > \hat{\phi}_1(n) - \phi(n) = (\sqrt{p} - \sqrt{q})^2.$$

Now we would like to compute the probability of having $\mathrm{ord}_n 2 > (\sqrt{p} - \sqrt{q})^2$. First we recall from elementary number theory that for any integer b coprime to two primes p and q,

$$\mathrm{ord}_{pq}\, b = \mathrm{lcm}(\mathrm{ord}_p\, b, \mathrm{ord}_q\, b).$$

Thus, a lowerbound for $\mathrm{ord}_p 2$ is also a lowerbound for $\mathrm{ord}_n 2$.

An Experiment To find a probabilistic lowerbound for $\mathrm{ord}_p 2$ where p is a random prime, primes were generated randomly using the mpz_rrandomb function from the GNU Multiple Precision (GMP) library, and $\mathrm{ord}_p 2$ was computed. Over 150,000 primes were generated, and for *every* prime p it was found that $\mathrm{ord}_p 2 > p^{1/3}$. In fact, for all but one hundred or so of the primes it was found that $\mathrm{ord}_p 2 > p^{1/2}$, and all but a couple hundred even satisfied $\mathrm{ord}_p 2 > p^{2/3}$.

Therefore, we will assume that $\mathrm{ord}_p 2 > p^{1/3}$ and $\mathrm{ord}_q 2 > q^{1/3}$. Since we know that $\mathrm{ord}_n 2 = \mathrm{lcm}(\mathrm{ord}_p\, b, \mathrm{ord}_q\, b)$, we then have

$$(pq)^{1/3} \geq \mathrm{ord}_n 2 \geq \max(q^{1/3}, p^{1/3}) > n^{1/6}.$$

Hence, it seems that $\mathrm{ord}_n 2 > n^{1/6} > (\sqrt{p} - \sqrt{q})^2$ will hold with very high probability as long as $(\sqrt{p} - \sqrt{q})^2$ is less than $n^{1/6}$. But since we are assuming that $|p - q|$ is small, the condition that $n^{1/6} > (\sqrt{p} - \sqrt{q})^2$ is guaranteed.

However, this is a conservative estimate: it is possible that $\mathrm{ord}_p\, b$ and $\mathrm{ord}_q\, b$ are "mostly" coprime, or entirely coprime, and so $\mathrm{lcm}(\mathrm{ord}_p\, b, \mathrm{ord}_q\, b)$ would be closer to $n^{1/3}$ than to $n^{1/6}$. In this case, it seems probable, if slightly less so, that $\mathrm{ord}_n 2 > n^{1/3} > (\sqrt{p} - \sqrt{q})^2$.

An Exception It should be emphasized that the size of the primes tested varied between 2^{20} and 2^{50} because computing $\mathrm{ord}_p 2$ is difficult for very large primes as it requires factoring $p - 1$. Thus, the experiments described above might not provide an accurate representation of the probability of $\mathrm{ord}_p 2 > p^{1/3}$

being true for primes of the size required by the DSS. However, during the experimentation, the larger the prime was, the fewer exceptions there were when the prime was tested for $\operatorname{ord}_p 2 > p^{1/2}$. Thus, it seems that increasing the size of a prime increases the likelihood that $\operatorname{ord}_p 2 > p^{1/2}$.

Note that inequalities such as $\operatorname{ord}_p 2 > p^{1/3}$ do not hold for all primes, though it seems highly likely. For example, for any Mersenne prime, i.e. a prime of the form $p = 2^k - 1$ where k is prime, it is always the case that $\operatorname{ord}_p 2 = k = \lceil \log_2 p \rceil$ which is always less than $p^{1/3}$ if $k > 3$. Despite this, it seems that $\operatorname{ord}_n 2 > n^{1/3}$ is true with high probability for n the product of two randomly chosen primes.

Two conjectures It should be noted that for all numbers n for which a factorization of n was attempted using Phi-Finder (except those with $\log_2 n = 128$), $\hat{\phi}_1(n) - \phi(n)$ was vastly smaller than even $n^{1/3}$, and yet $\phi(n)$ was successfully computed by the Phi-Finder algorithm, in all cases. Thus, for all experimental data (except possibly the numbers with bit-length 128), it was found that $\operatorname{ord}_n 2 > \hat{\phi}_1(n) - \phi(n) = (\sqrt{p} - \sqrt{q})^2$ was true. The experimental data given here and above lead us to make the following conjecture.

Conjecture 7. *For a positive integer n, $\operatorname{ord}_n 2 > n^{1/3}$ with very high probability, and so Phi-Finder will compute $\phi(n)$ with very high probability.*

Conjecture 8. *For a positive integer n, $\operatorname{ord}_n 2 > n^{2/3}$ with high probability, and so Phi-Finder will compute $\phi(n)$ with high probability.*

Finally, it should be noted that if this condition does not hold, and Phi-Finder computes a multiple of $\operatorname{ord}_n 2$ other than $\phi(n)$, this is actually a condition for the sucessful early termination of Phi-Finder. This is because if a multiple of the order of 2, call it $k \operatorname{ord}_n 2$, exists in the interval $(\hat{\phi}_1(n), \phi(n))$ then $k \operatorname{ord}_n 2 > \phi(n)$ and so we have

$$\hat{\phi}_1(n) - k \operatorname{ord}_n 2 < \hat{\phi}_1(n) - \phi(n).$$

Hence, $\operatorname{ord}_n 2$ is reached before $\phi(n)$ by decrementing from $\hat{\phi}_1(n)$, so Phi-Finder requires fewer than the expected number of iterations, $(\sqrt{p} - \sqrt{q})^2 / \lfloor \log_2 pq \rfloor$.

Summary To summarize thus far, we are given n, a product of two primes that are fairly close together, and we want to find $\phi(n)$. First, we estimate $\phi(n)$ to be $\hat{\phi}_g(n)$. We know that $\hat{\phi}_g(n)$ is a certain positive distance above the actual value of $\phi(n)$, but we cannot compute the error of our estimate without having more information about p or q.

In the next section is developed a method of decrementing $\hat{\phi}_g(n)$ closer and closer to $\phi(n)$ and checking whether or not $\hat{\phi}_g(n) = \phi(n)$ that is faster than the naive method described above.

3.3 Improvements

Let E denote $\hat{\phi}_g(n)$, our estimate of $\phi(n)$. Note that if we compute $2^E \pmod{n}$, and the result is still a power of 2, say 2^t, then we would have

$$2^E \equiv 2^{E-\phi(n)} \equiv 2^t \mod n.$$

Because $2^E \equiv 2^t \pmod{n}$, then we have

$$2^{E-t} \equiv 1 \equiv 2^0 \mod n.$$

This equivalency implies that $E - t$ is an integer multiple of $\operatorname{ord}_n 2$ satisfying $E > E - t \geq \phi(n)$. However, if we assume that n is a product of two safeprimes and we are using the estimate $g = 1$, then we know by Theorem 6 that $E - t$ is a multiple of $\phi(n)$ since no integer multiple of $\operatorname{ord}_n 2$ exists inside the interval $(E, \phi(n))$. Then by Corollary 5 we have that $2\phi(n) > E$, and so we know $E - t = \phi(n)$. If n is not the product of two safeprimes then it is still highly probable that $E - t = \phi(n)$, as discussed above in Section 3.2.

Now observe that it will only be the case that $2^E \equiv 2^t$ for $1 \leq 2^t < n$ if we have $0 \leq t \leq \log_2(n)$. This is because the inequality $t > \log_2(n)$ implies $2^t > n$. From this we can conclude that if $\hat{\phi}$ is within $\lfloor \log_2(n) \rfloor$ of $\phi(n)$, then $2^{\hat{\phi}}$ reduced modulo n will be congruent to some lower power of 2 in the integers, 2^t. This means that we need not test every value $E - x$ to see if $E - x = \phi(n)$. Instead, we can take larger steps of size $\lfloor \log_2(n) \rfloor$.

Testing each value $E - x$ by evaluating 2^{E-x} introduces the problem of repeatedly taking modular exponentiations. This is costly, especially since $E-x$ is enormous, i.e. $O(n)$. To improve on this, we first set $L = \lfloor \log_2(n) \rfloor$ and $E_0 = 2^{-E} \mod n$. Now we can decrement our estimate E so that it approaches $\phi(n)$. We do this by repeatedly multiplying E_i by 2^L and then reducing mod n. Note that multiplying by 2^L is really just shifting left by $\lfloor \log_2(n) \rfloor$ bits. Thus, we have an initial value E_0 and an iterative process wherein we multiply E_0 by $2^L \pmod{n}$, by shifting bits.

After each iteration, we need to check whether the current value

$$E_i = E_{i-1} \cdot 2^L = E_0 \cdot 2^{iL}$$

is a power of 2 after being reduced mod n. This can be done efficiently by checking the binary representation of the number. Thus, each iteration costs

- one bit shift by L bits
- one modular reduction by n
- checking if the result is a power of 2 in the integers
- and then incrementing a counter i.

```
INPUT: A product of two distinct primes, $n$.
OUTPUT: $\phi(n)$
  1. Set inital values:
     (a) $E \leftarrow \hat{\phi}_1(n)$
     (b) $E_0 \leftarrow (2^E)^{-1}$ modulo $n$
     (c) $L \leftarrow \lfloor \log_2 n \rfloor$
     (d) $i \leftarrow 0$
  2. While $E_i$ “is not a power of 2 in the integers” do
     (a) shift $E_i$ left by $L$ bits
     (b) reduce $E_i$ modulo $n$
     (c) $i \leftarrow i + 1$
  Note: At this point we know:
     • $E_i \equiv 2^t (\text{mod } n)$, for some integer $t$.
     • $0 \le t \le L$
     • $E_i = 2^{iL-E}$
     • $2^{iL-E} \equiv 2^t \mod n$
     • $2^{iL-E-t} \equiv 1 \equiv 2^0 \mod n$
     • $iL - E - t \equiv 0 \mod \text{ord}_n 2$
     • Thus, $iL - E - t$ is a multiple of $\text{ord}_n 2$, but by Theorem 6 we know
       that $iL - E - t = \phi(n)$.
  3. Return $\phi(n) \leftarrow iL - E - t$
```

Figure 1: The Phi-Finder Algorithm

4 Analysis of the Phi-Finder Algorithm

As mentioned at the end of Section 3.3, each iteration of the main loop of the algorithm has four operations. Out of these four steps, reducing $E_i \cdot 2^L$ modulo n, is the rate-determining step, as it is a multi-byte operation. The process of checking whether E_i is a power of 2 in the integers consists of evaluating the floor of size of E_i in bits and then scanning E_i to check that the position of its first (right-most or least-significant) non-zero bit is the same as its size in bits. The other three operations are done with the “mpz_mul_2exp” and “mpz_mod” functions in the GMP Library.

The algorithm is more efficient when r is near 1 and the user estimates $g = 1$, so we will consider this case first. The algorithm checks the integers between $\hat{\phi}_1(n)$ and the true value of $\phi(n)$ in intervals of $L = \lfloor \log_2(n) \rfloor$. There are $\hat{\phi}_1(n) - \phi(n) = (\sqrt{p} - \sqrt{q})^2$ such integers to check, but since the algorithm decrements by intervals of L, we see that the algorithm uses $(\sqrt{p} - \sqrt{q})^2 / \lfloor \log_2(pq) \rfloor$ iterations. Thus the number of bit-shifts and modular reductions required is

$$\frac{\left(\sqrt{p} - \sqrt{q}\right)^2}{\lfloor \log_2(pq) \rfloor}$$

plus one modular inversion in the setup phase. Now one can calculate how close together p and q must be for the algorithm to work efficiently.

Lemma 9. *If $r = 1 + 2^x$ then $\sqrt{r} < 1 + 2^{x-1}$.*

Proof.

$$\begin{aligned} r &= 1 + 2^x \\ &= 1 + 2 \cdot 2^{x-1} \\ &< 1 + 2 \cdot 2^{x-1} + 2^{2x-2} \\ &< (1 + 2^{x-1})^2 \\ \sqrt{r} &< 1 + 2^{x-1} \end{aligned}$$

□

Theorem 10. *Let $n = pq$, let $1 < p/q < \sqrt{2}$, and let $k \in \mathbb{R}$ be such that $|p - q| = \sqrt{n}2^{-k}$. Then*

$$\hat{\phi}_1(n) - \phi(n) < \sqrt{n}2^{-2k-7/4}.$$

Proof. Let $\epsilon \in \mathbb{R}$ such that $q = \sqrt{n}2^{-\epsilon}$. Now since $|p - q| = \sqrt{n}2^{-k}$, we have that

$$\begin{aligned} r &= p/q = (q + p - q)/q = 1 + (p - q)/q \\ &= 1 + \frac{\sqrt{n}2^{-k}}{\sqrt{n}2^{-\epsilon}} = 1 + 2^{-k+\epsilon}. \end{aligned}$$

Since $r = 1 + 2^{-k+\epsilon}$, by Lemma 9 we have that

$$\begin{aligned} \sqrt{r} &< 1 + 2^{-k-1+\epsilon} \\ \sqrt{r} - 1 &< 2^{-k-1+\epsilon} \\ (\sqrt{r} - 1)^2 &< 2^{-2k-2+2\epsilon} \end{aligned}$$

and by Corollary 4, we have that

$$\begin{aligned} \hat{\phi}_1(n) - \phi(n) &= q(\sqrt{r} - 1)^2 < q \cdot 2^{-2k-2+2\epsilon} \\ &< \sqrt{n}2^{-\epsilon} \cdot 2^{-2k-2+2\epsilon} \\ &< \sqrt{n}2^{-2k-2+\epsilon} \end{aligned}$$

Then $p = \frac{n}{q} = \frac{n}{\sqrt{n}2^{-\epsilon}} = \sqrt{n}2^{\epsilon}$ which implies $r = \frac{p}{q} = \frac{\sqrt{n}2^{\epsilon}}{\sqrt{n}2^{-\epsilon}} = 2^{2\epsilon}$. But we also know that $1 < r < \sqrt{2}$, so we have $2^0 < 2^{2\epsilon} < 2^{1/2}$. Thus, $0 < \epsilon < 1/4$, and so we have

$$\hat{\phi}_1(n) - \phi(n) < \sqrt{n}2^{-2k-2+\epsilon} < \sqrt{n}2^{-2k-7/4}$$

□

In the case that $g \neq 1$, the Phi-Finder algorithm is less efficient and its runtime is more difficult to describe. However, the following theorem provides rough bounds on the runtime of the Phi-Finder algorithm when the ratio $r = p/q$ is estimated to be $g \neq 1$.

Theorem 11. *Let $n = pq$, let $r = p/q$ satisfy $1 < r < \sqrt{2}$, and let g, our estimate of r, satistfy $0 < r - g < 2^{-k}$. Then*

$$\hat{\phi}_g(n) - \phi(n) < \sqrt{n}2^{-k-2}.$$

Proof. By equation (6) of Theorem 3 we have that

$$\hat{\phi}_g(n) - \phi(n) < \sqrt{n}(\sqrt{r} - \sqrt{g})(1 - 1/\sqrt{rg}).$$

Since $1 < g < r < \sqrt{2}$, we have that $gr < 2$, which implies that $\sqrt{gr} < \sqrt{2}$. This gives us that $-1/\sqrt{rg} < -\sqrt{2}/2 < 0$. Hence, we have that

$$(1 - 1/\sqrt{rg}) < 1 - \sqrt{2}/2 < 1/2.$$

This inequality implies that

$$\hat{\phi}_g(n) - \phi(n) < \sqrt{n}(\sqrt{r} - \sqrt{g})(1 - 1/\sqrt{rg}) < \sqrt{n}(\sqrt{r} - \sqrt{g})(1/2).$$

Furthermore, we know that $r - g = 2^{-k}$, so we have

$$\begin{aligned} r - g &= \sqrt{r}^2 - \sqrt{g}^2 \\ &= (\sqrt{r} - \sqrt{g})(\sqrt{r} + \sqrt{g}) = 2^{-k} \\ \left(\sqrt{r} - \sqrt{g}\right) &= 2^{-k} / \left(\sqrt{r} + \sqrt{g}\right) \end{aligned}$$

But r and g are both at least 1, and so $\sqrt{r} + \sqrt{g}$ is at least $1^{1/2} + 1^{1/2} = 2$. This implies that

$$(\sqrt{r} - \sqrt{g}) < 2^{-k}(1/2) = 2^{-k-1}.$$

Finally, we have

$$\hat{\phi}_g(n) - \phi(n) < \sqrt{n}(\sqrt{r} - \sqrt{g})(1/2) < \sqrt{n}2^{-k-2}$$

which proves the theorem. □

The version of Phi-Finder with $g \neq 1$ is inferior to the runtime of Phi-Finder when g = 1, since the exponent k in the expression of the runtimes has a coefficient of 2 in the latter whereas the coefficient of k is 1 in the former. This implies that Phi-Finder with $g \neq 1$ uses 2^k times the number of iterations that would be used if $g = 1$. Surely improvements can be made on the bounds for the case when $g \neq 1$, but this is left for future work.

5 Experimental Results

We present in the tables below the sizes of products of two primes p and q, the sizes of the differences $|p-q|$ and $(\sqrt{p}-\sqrt{q})^2$, as well as the time to factor each. Take as a concrete example of these data the following product of two primes:

> pq = 12 564 733 994 859 564 943 378 851 488 880 676 440 897 987 946 823 714 584 149 539 076 705 853 364 647 623 021 561 002 480 605 777 661 118 593 343 755 993 562 300 984 166 544 929 173 711 618 863 066 289 571,
>
> and
>
> p = $\underline{\text{112 092 524 259 468 637 159 311 703 158 589 44}}$6 157 609 360 330 066 292 477 642 325 201 809 507 359 171,
>
> q = $\underline{\text{112 092 524 259 468 637 159 311 703 158 589 44}}$4 935 093 419 977 906 478 398 217 165 142 400 568 522 401

The product is 512 bits in length, and each prime is 256 bits long. The difference, $p-q$, is 139 bits, and the Phi-Finder algorithm factored n in under .01 seconds. Note that the 35 underlined digits of p and q above are identical, and p and q each have 78 decimal digits, while n has 155 digits.

The runtimes displayed in Tables 3 and 2 show that Phi-Finder factors a semiprime n instantly when $|p-q| \approx n^{1/4}$. The tables show that the algorithm succeeds rather quickly even when $|p-q|$ is slightly greater than $n^{1/4}$. In addition, the algorithm can succeed that quickly with integers that are very large: the tables show runtimes for integers n with $\log_2 n$ up to 3072, which is quite large even by the DSS, which requires that $\log_2 n \geq 2048$ [7].

The DSS also requires that

$$|p-q| > 2^{(\log_2 n)/2-100} = \sqrt{n}2^{-100}.$$

Let us consider two sizes of an integer n meeting this standard. If $\log_2 n \approx 2048$, as required by the DSS, then $|p-q| > \sqrt{n}2^{-100} \approx 2^{924}$ [7]. If we suppose that $|p-q|$ is very near this lowerbound, then $|p-q| \approx \sqrt{n}2^{-100}$. By Theorem 10, an integer n with these properties would require less than $\sqrt{n}2^{-200-7/4} = 2^{823+1/4}$ iterations. Note from the data tables that when $(\sqrt{p}-\sqrt{q})^2$ was near 2^{40}, i.e. when the algorithm required roughly 2^{40} iterations, the algorithm took about 30 minutes to succeed. Using the number of iterations per second given in Table 4, we can estimate that an integer n with $\log_2 n \approx 2048$ and $|p-q| > 2^{924}$ would take at least 2^{815} seconds (about $6.9 \cdot 10^2 37$ CPU years) to succeed.

On the other hand, if $\log_2 n \approx 512$, then $|p-q| > 2^{156}$ [7]. By Theorem 10, an integer n with these properties would require less than $\sqrt{n}2^{-200-7/4} = 2^{55+1/4}$ iterations. Note from the data tables that when $(\sqrt{p}-\sqrt{q})^2$ was near 2^{40}, i.e. when the algorithm required roughly 2^{40} iterations, the algorithm took about 30 minutes to succeed. Using the number of iterations per second given in Table 4, we can estimate that an integer n with $\log_2 n \approx 512$ and $|p-q| \approx 2^{156}$ would take approximately 2^{234} processor seconds (about 645 CPU years) to succeed.

Although this amount of time seems infeasible, the RSA challenge number RSA-768 mentioned above was factored with the GNFS in 3 calendar years using approximately 1500 CPU years [15]. Thus, it seems that computations of this size are now possible, although they have been accomplished so far only through large-scale parallelization [15].

Conclusion We conclude that the minimum requirement for $|p-q|$ given in the DSS ensures that n is difficult to factor via a naive Phi-Factor approach, i.e. starting from 1. However, by beginning the Phi-Finder algorithm so that it starts its search with the first value beyond the $|p-q|$ lower bound of the standard, Phi-Finder can factor an RSA moduli constructed according to the DSS standards, if $|p-q|$ is too close to the minimum.

Furthermore, this problem cannot be solved by increasing the minimum requirement for the size of $|p-q|$, as Phi-Finder can begin searching with the first value beyond the standard lower bound for $|p-q|$, no matter how large the lower bound is. Thus, we recommened that p and q are chosen at random so that the event of $|p-q|$ being too close to the minimum is sufficiently improbable.

Here we provide a very rough estimation of the probability of this event. Note that if p and q are chosen at random from the interval $(2^{m/2-1/2}, 2^{m/2})$ where $m = \log_2 n$, then once p is chosen, the probability of q being chosen within a certain distance d of p is roughly

$$2d/(2^{m/2} - 2^{m/2-1/2}) = 2d/\left((2^{m/2})(1-2^{-1/2})\right).$$

Because $|p-q| \leq 2^{20}n^{1/4}$ (or slightly larger) is required for Phi-Finder to factor n quickly, we then want to know the probability that $d \leq 2^{20}n^{1/4} = 2^{m/4+20}$. We can estimate the probability of this occurring to be

$$P\left(d \leq 2^{m/4+20}\right) = \frac{2 \cdot 2^{m/4+20}}{(2^{m/2})(1-2^{-1/2})} \approx 2^{20-m/4}.$$

Thus, as an example, if $m \approx 2048$, the probability of randomly choosing primes that are "too close together" is approximately 2^{-492}, which is sufficiently improbable to conclude that n is difficult to factor using the Phi-Finder algorithm.

6 Comparison to Other Algorithms

Previously, the factors of an RSA modulus were chosen so that they were not too close together for fear that trial division or Fermat Factorization might succeed. Therefore, the Phi-Finder algorithm is compared with Fermat Factorization below, and a comparison in runtimes is given in Tables 2 and 3. However, in the case of these algorithms the words "too close together" refer to a much smaller difference than that required to protect the modulus from factoring by Phi-Finder.

In comparing these algorithm, it is also worth noting that Phi-Finder requires almost no memory, merely the amount of memory needed to store n and bit shift n. In contrast, the QS, GNFS, and trial division use a great deal of memory. The QS and GNFS involve linear algebra phases which involve matrices of sizes that are often $500,000 \times 500,000$ and larger. Trial division would require storing all primes in the interval $[q, \sqrt{n}]$, which is discussed in further detail below.

Another comparable factoring method uses a method due to Coppersmith that factors n when enough bits of one of the prime factors of n are known [3]. First, we explain the relevance of Coppersmith's known-bit attack.

6.1 Coppersmith's Known-bit Attack

A method due to Coppersmith gives us the following result [3].

Theorem 12 (Coppersmith)**.** *Let $n = pq$ have m bits. If the $m/4$ MSBs or LSBs of p are known, then n can be factored in polynomial time.*

When $|p - q| < n^{1/4} = 2^{m/4}$, then $\sqrt{n}$ has the same $m/4$-MSBs as p and q and since $\sqrt{n}$ is publically computable, Coppersmith's algorithm provides a polynomial time factorization of n. In the case when $|p-q| < n^{1/4}$, the runtimes in Tables 2 and 3 show that Phi-Finder factors n instantly.

Note that Coppersmith's algorithm can still provide an efficient facorization even if less than $m/4$ bits are known because we can guess the value of the unknown bits and run the algorithm for each guess. For example, if the $m/4-k$ MSBs of one prime are known for some small positive integer k, then there would be 2^k different possibilities for those unknown bits, and so n could be factored by running Coppersmith's polynomial time algorithm no more than 2^k times.

However, this quickly becomes infeasible as missing knowledge of even $k = 10$ of the $m/4$ MSBs would require $2^{10} = 1024$ calls to Coppersmith's algorithm. In comparison, the runtimes in Tables 2 and 3 show that Phi-Finder succeeds in less than one second when $|p - q| \approx 2^{m/4+10}$ for even a 3072-bit modulus, and still succeeds in under 18 minutes even when $|p - q| \approx 2^{m/4+20}$.

6.2 Comparison to Fermat Factorization

Fermat factorization can also quickly factor n if p and q are too close together. This is because if the factors of a modulus are too close together, then $n = pq$ can easily be expressed as the difference of two squares: $n = x^2 - y^2$, which can then be factored as $n = (x - y)(x + y)$ and therefore p and q are $(x + y)$ and $(x - y)$. But how close must two primes be for their product to be quickly factored this way?

Suppose we try to factor $n = pq$ using Fermat Factorization. Then we want integers x and y so that $n = x^2 - y^2$, or, written another way, so that $n + y^2 = x^2$. To accomplish this, we start with $\lceil \sqrt{n} \rceil^2$, subtract n, and then check to see if the difference, $\lceil \sqrt{n} \rceil^2 - n$, is itself a perfect square. If $\lceil \sqrt{n} \rceil^2 - n$ is not a perfect square, then we try positive integers $i = 1, 2, 3, 4, ...$ with $(\lceil \sqrt{n} \rceil + i)^2 - n$ until it is a perfect square.

Note that this process is the same as checking to see if p and q are a certain distance apart. To see that, observe that if $(\lceil \sqrt{n} \rceil + i)^2 - n = x^2$ and we find the factorization $n = pq = (\lceil \sqrt{n} \rceil + i)^2 - x^2 = (\lceil \sqrt{n} \rceil + i + x)(\lceil \sqrt{n} \rceil + i - x)$, then $p = \lceil\sqrt{n}\,\rceil + i + x$ and $q = \lceil\sqrt{n}\,\rceil + i - x$, and so we have $p - q = 2x$. Thus, iterating i and checking to see whether or not $(\lceil\sqrt{n}\,\rceil + i)^2 - n$ is a perfect square is equivalent to checking if p and q are separated by a distance of $2x$, i.e. $|p - q| = 2x = 2\sqrt{(\lceil\sqrt{n}\,\rceil + i)^2 - n}$. This equation allows us to solve for for i, the number of iterations required, in terms of p and q.

Theorem 13. *Let $n = pq$, a product of distinct primes. Then factoring n using Fermat's method requires $\lfloor(\sqrt{p} - \sqrt{q})^2/2\rfloor$ iterations.*

Proof.

$$\begin{aligned}
2((\lceil\sqrt{n}\rceil + i)^2 - n)^{1/2} &= |p - q| \\
4\left((\lceil\sqrt{n}\rceil + i)^2 - n\right) &= (p - q)^2 \\
4(\lceil\sqrt{n}\rceil + i)^2 - 4n &= p^2 - 2n + q^2 \\
4(\lceil\sqrt{n}\rceil + i)^2 &= p^2 + 2n + q^2 = (p + q)^2 \\
2(\lceil\sqrt{n}\rceil + i) &= p + q \\
2(\sqrt{n} + o(1) + i) &= p + q \\
2(i + o(1)) &= \sqrt{p}^2 - 2\sqrt{n} + \sqrt{q}^2 = (\sqrt{p} - \sqrt{q})^2 \\
i &= (\sqrt{p} - \sqrt{q})^2/2 - o(1)
\end{aligned}$$

Thus, Fermat's method requires $\lfloor(\sqrt{p} - \sqrt{q})^2/2\rfloor$ iterations. □

We emphasize that this is a naive version of Fermat's algorithm, and versions of the algorithm exist that are slightly faster than this naive approach [6]. It is also important to note that the efficient success of Fermat's method does not depend on n being the product of two primes. If n can be expressed as a product fg and $f - g$ is small, then n will be factored efficiently regardless of the structure of f and g. The Phi-Finder algorithm will succeed quickly only if n is a product of two primes that are close together. If n is not the product of two primes that are close together then the Phi-Finder method will still factor n eventually but with a possibly enormous runtime.

6.3 Comparison to Trial Division

During trial division, one divides n by each consecutive prime less than $\sqrt{n}$ until a divisor of n is found. This process is guaranteed to produce a factor of n because n must have a prime factor less than its square root. Thus, succeeding using trial division requires that we divide n by each prime in the interval $[q, \sqrt{n}]$.

We can use the prime number theorem to estimate that the probability of a randomly chosen integer from that interval being prime is roughly $1/\ln(\sqrt{n}) = 2/\ln n$. Note that the probability of an integer being prime if it is chosen near q is closer to $1/\ln q$, which is greater than $2/\ln n$ because $q < \sqrt{n}$. Thus, $(\sqrt{n} - q)(2/\ln n)$ gives an underestimate for the number of primes in the interval, and hence the number of trial divisions necessary.

Now we can estimate the number of primes in this interval that we need to check. The estimated number of trial divisions can be expressed as follows:

$$\frac{2(\sqrt{n}-q)}{\ln n} = \frac{2}{\ln n}(\sqrt{q}(\sqrt{p}-\sqrt{q})).$$

This form allows us to readily see that trial division requires more iterations than the Phi-Finder algorithm. Since we have assumed that p and q are close together, we can assume that $4 > p/q$. But this implies $2\sqrt{q} > \sqrt{p}$, which in turn implies $\sqrt{q} > \sqrt{p} - \sqrt{q}$. Thus, we know that $\sqrt{q}(\sqrt{p}-\sqrt{q}) > (\sqrt{p}-\sqrt{q})^2$.

We also have that $\log_2(pq) > \ln(pq)$, which only further shows that

$$\sqrt{q}(\sqrt{p}-\sqrt{q})/\ln(pq) > (\sqrt{p}-\sqrt{q})^2/\log_2(pq).$$

Thus, trial division requires more operations than our algorithm, as long as $4q > p > q$. Keep in mind that this analysis does not take into account the enormous difficulty of finding and storing all of the primes between $\sqrt{n}$ and q, which is totally infeasible.

6.4 Comparison to the “qsieve” and “factor” commands in SAGE

The open source computer algebra system SAGE has a highly optimized implementation of the Quadratic Sieve (QS) as well as a factoring command “factor,” that uses a variety of algorithms [1]. Note from Table 2 that Phi-Finder can factor 128-bit integers in under 30 minutes even when $|p-q|$ is large. However, other factoring methods are so effective on integers of this small size that Phi-Finder is not useful here: SAGE’s factor command factored these in less than one second, regardless of the value $|p-q|$, see [1].

The qsieve command in SAGE, which accepts as inputs only 140-bit integers and larger, factored integers up to 260 bits in length within 12 minutes, regardless of the size of $|p-q|$. The runtimes given in Table 1 confirm that the size of $|p-q|$ does not affect the runtime of the Quadratic Sieve. Integers of four different bit-lengths were factored : 200, 220, 240, and 260. For each bit-length, one integer was factored for which $|p-q|$ was very small, and one for which it was very large. The QS took roughly the same amount of time for integers of the same bit-length, despite the different sizes of $|p-q|$.

In contrast, for numbers of the sizes listed in Table 1, Phi-Finder succeeded instantly for the integers for which $|p-q|$ was small, whereas the QS took as long as ten minutes. However, Phi-Finder would take at least 2^{52} seconds (more than $1.4 \cdot 10^8$ CPU years) to factor the integers with large $|p-q|$, whereas the QS took less than twelve minutes.

Clearly the QS factors n faster than Phi-Finder for integers $n = pq$ of this size (between 200 and 260 bits in length) when $|p-q|$ is not very small, but integers for which $\log_2 n = 260$ are much smaller than RSA moduli, which are required by the DSS to have $\log_2 n \geq 2048$ [7]. For integers n with $\log_2 n \geq 2048$, even the GNFS requires an infeasible amount of time to factor n, regardless of whether or not n has any special structure [15]. In contrast, Phi-Finder can quickly factor integers $n = pq$ of this large size in only minutes, if $|p - q|$ is very small.

$\log_2(n)$	$\log_2(p-q)$	$\log_2(\sqrt{p}-\sqrt{q})^2$	Time (sec)
199.029	57.9978	0.0000	14.90
199.023	91.7761	82.0405	11.30
220.261	58.0000	0.0000	40.23
220.268	99.7890	87.4440	45.02
240.000	70.0000	0.0000	187.51
240.000	115.9827	109.96508	176.10
260.000	75.0000	0.0000	715.86
260.000	125.34312	118.68612	594.37

Table 1: Run times for the Quadratic Sieve as implemented on Sage.

7 Acknowledgements

Many thanks to Dr. Gregory V. Bard of Fordham University, the author's thesis advisor, for much advice in research and writing as well as many revisions of the paper. Thanks also to the Faculty Research Council of Fordham College of Arts and Sciences at Rose Hill for a faculty research grant given to Dr. Bard in the summer of 2009, under which the author of this thesis was supported. The author is also indebted to Dr. Donna Heald, Associate Dean for Science Education at Fordham University for funding in the summer of 2009.

Thanks to Dr. Lawrence C. Washington of the University of Maryland at College Park for observing that Coppersmith's algorithm can be used if $|p-q|$ is small enough (and therefore p and q share many of their MSBs with $\sqrt{n}$). Thanks also to Dr. William Hart of Warwick University for help with Fermat's factorization method. We thank Dr. William Stein of the University of Washington in Seattle and National Science Foundation Division of Mathematical Sciences Grant No. DMS-0821725 for the use of the computers in running the algorithms. Finally, the author is grateful to the entire Mathematics Department at Fordham College Rose Hill for support and instruction in many aspects of mathematics.

References

[1] SAGE. Software Package. Available at http://www.sagemath.org/.

[2] M. S. Manasse A. K. Lenstra, H. W. Lenstra Jr. and J. M. Pollard. The number field sieve. *Preprint*, December, 1989.

[3] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology*, 10:233–260, 1997.

[4] Yousef Bani Hammad. Novel methods for primality testing and factoring. *Dissertation for a PhD in Mathematics at Queensland University of Technology*, 2005.

[5] H. W. Lenstra Jr. Factoring integers with elliptic curves. *Ann. of Math.*, 126(2):649–673, 1987.

[6] James McKee. Speeding fermat's factoring method. *Math. Comp*, (68):228–1729, 1999.

[7] National Institute of Standards and Technology (NIST). FIPS publication 186-3: Digital signature standard (DSS), June 2009.

[8] J M Pollard. A monte carlo methods for factorization. *BIT*, (15), 1975.

[9] J.M. Pollard. Theorems on factorization and primality testing. *Proc. Cambridge Phil. Soc.*, 76:521–528, 1974.

[10] Carl Pomerance. The quadratic sieve factoring algorithm. In *Advances in cryptology*, pages 169–182, 1984.

[11] A. Shamir R. Rivest and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[12] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26:1484–1509, 1997.

[13] R D Silverman. Massively distributed computing and factoring large integers. *Communications of the ACM*, (34):95–103, 1991.

[14] Robert D. Silverman. The multiple polynomial quadratic sieve. *Mathematics of Computation*, 1987.

[15] Jens Franke Arjen K. Lenstra Emmanuel Thom Pierrick Gaudry Peter L. Montgomery Dag Arne Osvik Herman Te Riele Andrey Timofeev Paul Zimmermann Thorsten Kleinjung, Kazumaro Aoki. Factorization of a 768-bit rsa modulus, February.

[16] H C Williams. A p+1 method of factoring. *Mathematics of Computation*, (39), 1982.

[17] Song Y. Yan. *Cryptanalytic Attacks on RSA*. Springer-Verlag, 2008.

Appendix I: Probabalistic Analysis of the Order of 2

An additional two experiments were carried out to test Conjectures 7 and 8, each consisting of 1,000,000 trials. In both experiments, each trial consisted of generating a prime q, calculating the order of $2 \mod q$, and then checking whether $\mathrm{ord}_q 2$ was greater than $n^{1/2}$. In both experiments it was found that $\mathrm{ord}_q 2 > n^{1/2}$ was true for all 1,000,000 trials.

The only difference in the two experiments was the size of the primes generated. In the first experiment, each trial involved generating a prime at random in the interval $(2, 2^{45})$. In the second experiment, the interval used was $(2, 2^{47})$. Again we emphasize that these sizes of primes are much smaller than the size of a prime used in an RSA modulus.

Confidence intervals: Given the data from these experiments, we would like to compute some confidence intervals for the probability p of the conjecture failing for a randomly selected prime. Now we would like to know that for some upper bound U we have $P(p \geq U) = 1 - \alpha$, i.e. that $p \in (0, U]$ with $100(1 - \alpha)\%$ confidence. Note that, since p is the probability of the conjecture failing, then $1 - p$ is the probability of a success occurring, and $(1-p)^{10^6}$ is the probability of 1,000,000 consecutive successes occurring.

$$\begin{aligned}(1-p)^{10^6} &\geq \alpha \\ 10^6 \log_{10}(1-p) &\geq \log_{10}\alpha \\ \log_{10}(1-p) &\geq 10^{-6}\log_{10}\alpha \\ (1-p) &\geq \alpha 10^{10^{-6}} \\ 1 - \alpha 10^{10^{-6}} &\geq p\end{aligned}$$

Thus, we have that $p \leq 1 - \alpha 10^{10^{-6}}$ with $100(1-\alpha)\%$ confidence, where p is the probability of a prime violating the conjecture. Using this calculation,we give several upperbounds U for p as well as the confidence of each U in the table below. Note that since both experiments have the same data (1,000,000 trials with 1,000,000 successes), these confidence intervals apply for both experiments, i.e. for primes of both sizes.

Confidence	Upperbound for p	Lowerbound for $1-p$
99.99%	0.00001381	0.999986
99.9%	0.00001151	0.99998849
99%	0.000009210	0.99999079
90%	0.000006907	0.999993093

Appendix II: A More Rigorous Analysis of the Random Selection of RSA Primes

A rough probabilistic estimate is given at the end of Section 5 suggesting that random selection of primes for an RSA modulus guarantees with sufficiently high probability that the modulus will be difficult to factor using Phi-Finder. Here we provide a more rigorous analysis of this probability.

Recall that we are randomly choosing two primes p, q in an interval, call the interval (a, b), and we want to calculate the probability that $|p-q| \leq D$ for some pre-determined distance, D. This requires counting the number of primes in the interval (a, b).

Approximating $\pi(n)$: Since we are counting the number of primes in the interval, we would like a function that approximates the numbers of primes in the interval (a, b), and we would like this function to be simpler than the prime number function $\pi(n) = n/\ln n$ to make the calculations simpler. We estimate $\pi(n) \approx \hat{\pi}(n) = n/\ln((a + b)/2) = nK$ (where $K = 1/\ln((a + b)/2)$), i.e. we will use the density of primes in the center of the interval as an estimate for the density of the primes over the whole interval. To give bounds on the error of this estimation we need only check the error at the interval boundaries. This is because $\pi(n)$ is a monotonically increasing function and so the worst that the error of $\hat{\pi}(n)$ can be is at the extremes of the interval, i.e. at a and b.

A standard interval for the selection for RSA primes given in [7] is the interval $(2^{1023.5}, 2^{1024})$. We calculate the error for $\hat{\pi}(n)$ for this interval as follows.

$$\begin{aligned}
\hat{\pi}(2^{1023.5}) &= 2^{1023.5}/\ln((2^{1023.5} + 2^{1024})/2) \\
&= 2^{1023.5}/\ln(2^{1022.5} + 2^{1023}) \\
&= 2^{1023.5}/\ln(2^{1022.5}(1 + 2^{.5})) \\
&= 2^{1023.5}/(\ln(2^{1022.5}) + \ln(1 + 2^{.5})) \\
&= 2^{1023.5}/(1022.5 \ln 2 + \ln(1 + 2^{.5})) \\
&= 2^{1023.5}/(1022.5 \ln 2 + .881373587)
\end{aligned}$$

Now we compute our absolute error as

$$\begin{aligned}
\pi(2^{1023.5}) - \hat{\pi}(2^{1023.5}) &= 2^{1023.5}/(1023.5 \ln 2) - 2^{1023.5}/(1022.5 \ln 2 + .881373587) \\
&= 2^{1023.5}[1/(1023.5 \ln 2) - 1/(1022.5 \ln 2 + .881373587)] \\
&= 2^{1023.5}\left[\frac{(1022.5 \ln 2 + .881373587) - (1023.5 \ln 2)}{(1023.5 \ln 2)(1022.5 \ln 2 + .881373587)}\right] \\
&= 2^{1023.5}\left[\frac{.881373587 - \ln 2}{(1023.5 \ln 2)(1022.5 \ln 2 + .881373587)}\right] \\
&= 2^{1023.5}\left[\frac{.5429440881}{503433.170}\right]
\end{aligned}$$

Finally, we compute our relative error at the interval's lower boundary to be

$$\frac{\pi(2^{1023.5}) - \hat{\pi}(2^{1023.5})}{\pi(2^{1023.5})} = 0.0007651 = .07651\%$$

and a similar calculation gives our relative error at the interval's upper boundary

$$\frac{\pi(2^{1024}) - \hat{\pi}(2^{1024})}{\pi(2^{1024})} = -0.0002231 = -.02231\%$$

Thus, we see that $\hat{\pi}(n)$ provides an accurate approximation of $\pi(n)$ in the given interval. Now we use our approximation $\hat{\pi}(n) = Kn$ to approximate the number of primes in our interval that satisfy $|p - q| \le D$.

To do this we will count the number of pairs of primes in (a, b) that satisfy $|p-q| \le D$ and divide by the total number of pairs of primes in (a, b). First we divide our interval (a, b) into three subintervals, $(a, a + D)$; $(a + D, b - D)$; and $(b - D, b)$. We do this because the number of primes that we want to count changes near the boundaries: for example, if p is chosen in the interval $(a+D, b-D)$ then q can be any prime within

D of p. However, if p is chosen in either of the near-boundary intervals, for example $(a, a + D)$, then q cannot be chosen if it is less than a, and so the interval around p from which q can be chosen shrinks from $(p - D, p + D)$ to $(a, p + D)$.

The first prime, p, must be chosen from $(a, b) = (a, a+D) \cup (a+D, b-D) \cup (b-D, b)$. However, we have shown that the number of possible primes q then depends on which of the three subintervals that p is chosen from. Since $(a + D, b - D)$ is the largest subinterval by far, it will dominate the probability. Note that there are

$$\begin{aligned} \pi(b - D) - \pi(a + D) &\approx \hat{\pi}(b - D) - \hat{\pi}(a + D) \\ &= K(b - D) - K(a + D) \\ &= K(b - a - 2D) \end{aligned}$$

primes in the interval $(a+D, b-D)$. Similarly, there are approximately $K(b) - K(a) = K(b - a)$ primes in the entire interval (a, b), and so the probability of p being selected from $(a + D, b - D)$ at random is given by

$$P\{p \in (a + D, b - D)\} = \frac{K(b - a - 2D)}{K(b - a)} = \frac{b - a - 2D}{b - a} = 1 - \frac{2D}{b - a}$$

Furthermore, the number of primes q within the distance D of p is then $K(p + D) - K(p - D) = 2KD$. Thus, the probability of q being chosen out of the interval (a, b) such that q is within D of p, given that $p \in (a + D, b - D)$, is then $2KD/(K(b - a)) = 2D/(b - a)$. Therefore, the probability that $p \in (a + D, b - D)$ and $|p - q| \leq D$ is then

$$Pp \in (a + D, b - D) \wedge |p - q| \leq D = \left(1 - \frac{2D}{b - a}\right) \frac{2D}{b - a}$$

Now note that the probability of p being selected randomly from one of the outer intervals is given by

$$P\{p \in (a, a + D) \cup (b - D, b)\} = 1 - P\{p \in (a + D, b - D)\} = \frac{2D}{b - a}$$

In practice, $b - a$ is on the order of $n^{1/2}$ and D is on the order of $2^{20}n1/4$, and so this probability is on the order of $2^{20}n^{-1/4}$. Since $n \approx 2^{2048}$, this probability is roughly 2^{-492}, and so the case where p is not in the dominant interval is negligible.

Hence, we estimate that the probability of $|p - q| \leq D$ when p and q are chosen at random is then $\left(1 - \frac{2D}{b-a}\right) \frac{2D}{b-a}$. For example, using the standard values $a = 2^{1023.5}$, $b = 2^{1024}$, and $D \leq 2^{532}$, we have

$$\begin{aligned} P\{|p - q| \leq 2^{532}\} &= \left(1 - \frac{2 \cdot 2^{532}}{b - a}\right) \frac{2 \cdot 2^{532}}{b - a} \\ &= \left(1 - \frac{2^{533}}{2^{1024} - 2^{1023.5}}\right) \frac{2^{533}}{2^{1024} - 2^{1023.5}} \\ &= 2^{-490.5}(\sqrt{2} + 1) - \left(2^{-490.5}(\sqrt{2} + 1)\right)^2 \\ &\approx 2^{-490.5}(\sqrt{2} + 1) \approx 2^{-489.2284} \end{aligned}$$

which is sufficiently improbable.

$\log_2(n)$	$\log_2(p-q)$	$\log_2(\sqrt{p}-\sqrt{q})^2$	Phi-Finder		Fermat
128	39.9910	14.0447	0.00		0.00
128	44.9253	23.9830	0.02		0.35
128	45.4076	25.2188	0.04		0.85
128	46.6700	27.4590	0.20		3.93
128	47.5329	29.0746	0.63		10.88
128	48.8298	31.7992	3.84		77.80
128	49.4470	33.0060	9.20		179.98
128	50.2547	34.8579	34.83		609.72
128	51.4826	37.1537	162.26		$>$2000.00
128	52.1715	38.4032	374.33		$>$2000.00
128	53.17184	40.34585	1524.40		$>$2000.00
128	54.59147	43.255947	$>$2000.00		$>$2000.00
512	139.957	21.9631	0.00		0.13
512	140.999	24.1443	0.02		0.53
512	141.546	25.5872	0.04		1.39
512	142.361	26.7307	0.10		3.18
512	143.795	29.9616	0.96		30.56
512	144.303	30.9417	2.08		60.22
512	145.496	33.0012	7.99		237.49
512	146.885	36.003	72.12		1904.23
512	147.665	37.8254	217.65		$>$ 2000.00
512	148.512	39.0332	522.63		$>$ 2000.00
512	149.266	40.6691	1552.13		$>$ 2000.00
1024	267.498	21.0608	0.01		0.08
1024	268.788	24.0035	0.03		0.68
1024	269.543	25.1192	0.04		1.75
1024	270.818	27.7879	0.29		8.98
1024	271.472	29.2853	0.82		25.94
1024	272.528	31.1892	2.96		96.98
1024	273.978	34.1086	22.48		762.74
1024	274.662	35.4257	52.95		1867.15
1024	275.311	37.0332	195.69		$>$ 2000.00
1024	276.557	39.355	774.27		$>$ 2000.00
1024	277.603	41.2856	$>$ 2000.00		$>$ 2000.00

Table 2: Run times are in seconds.

$\log_2(n)$	$\log_2(p-q)$	$\log_2(\sqrt{p}-\sqrt{q})^2$	Phi-Finder		Fermat
2048	521.098	16.2379	0.01		< 0.01
2048	522.026	18.5101	0.01		0.02
2048	523.382	20.9696	0.01		0.12
2048	524.428	23.0637	0.03		0.53
2048	525.033	24.1835	0.05		1.11
2048	526.75	27.5525	0.34		12.10
2048	527.754	29.5410	1.33		47.81
2048	528.787	31.9979	7.80		261.97
2048	529.927	33.9446	28.95		998.85
2048	530.334	34.8726	54.35		1940.36
2048	531.217	36.8897	227.87		>2000.00
2048	532.395	38.9955	947.28		>2000.00
2048	533.487	41.3567	>2000.00		>2000.00
3072	777.592	17.3730	0.04		< 0.01
3072	778.555	19.4147	0.04		0.06
3072	779.913	22.1133	0.04		0.35
3072	780.129	22.7074	0.06		0.57
3072	781.949	26.1382	0.22		6.15
3072	782.964	28.4105	0.95		30.50
3072	783.688	29.4796	1.87		60.35
3072	784.11	30.5733	3.90		125.46
3072	785.755	33.6979	33.69		1133.10
3072	786.262	34.6330	61.89		> 2000.00
3072	787.399	36.9869	333.21		> 2000.00
3072	788.187	38.6777	1051.78		> 2000.00
3072	789.359	41.0980	> 2000.00		> 2000.00

Table 3: Run times are in seconds.

$\log_2(n)$	Iterations per second	integers tested as $\phi(n)$ per second
128	7161500	909511000
512	2107880	1077130000
1024	811058	829713000
2048	281713	576667000
3072	112164	344455000
4096	76407	312889000

Table 4: Number of Iterations run per second.