# Primality Testing and Factorization Methods

Eli Howey

May 27, 2014

## Abstract

Since the days of Euclid and Eratosthenes, mathematicians have taken a keen interest in finding the nontrivial factors of integers, as well as in finding prime numbers, which have no such factors. Until only recently, however, the problem of factoring numbers had no practical application beyond the advancement of pure mathematics. That changed when the RSA algorithm was published in 1977; now, prime factorization and random number generation are vital to the security of nearly every form of sensitive data, from private communications to financial transaction information.

This project will focus on a few of the various algorithms that have been developed over the centuries to test the primality of numbers and to factor large numbers into their prime components, with regard given to each algorithm's time complexity.

## 1 The RSA Algorithm

Since the primary modern motivation for factoring numbers into primes is to crack various cryptosystems, it will benefit us to understand the process by which we use these primes to encrypt information in the first place. The RSA algorithm, developed in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman [1], is perhaps the most prevalent cryptographic algorithm in use today. This section shall describe the algorithm, and the number theory relevant to understanding its implementation.

RSA relies on two keys: a *public key*, used for information encryption and known to anyone, and a *private key*, used to decrypt the information and known only to the key generator.

The algorithm makes use of *Euler's totient function*, $\phi(n)$, defined as the number of positive integers less than $n$ and coprime to $n$, i.e.,

$$\phi(n) = \#\{k \in \mathbb{Z}^+ \mid k \leq n \text{ and } \gcd(k, n) = 1\}.$$

It is simple to show that this function is multiplicative, that is, $\phi(ab) = \phi(a)\phi(b)$ for all $a, b \in \mathbb{Z}^+$. Moreover, if $p$ is prime, then it shares no nontrivial factors with any smaller integer, so $\phi(p) = p - 1$.

RSA key generation also requires computing the *multiplicative inverse* of a number modulo $\phi(n)$. The multiplicative inverse of $a$ modulo $n$ is the value $b$ such that $ab \equiv 1 \mod n$. In the case where $a$ and $b$ are coprime, the integer solutions $x$ and $y$ to $ax + by = 1$ represent the multiplicative inverses of $a$ modulo $b$, and of $b$ modulo $a$, respectively.

**Algorithm** (RSA key generation)**.** Suppose we have access to a set of prime numbers, or a prime number generator.

1. Randomly choose two distinct primes $p$ and $q$ of approximately equal size.[1]
2. Compute $n = pq$.
3. Compute $\phi(n) = \phi(p)\phi(q) = (p - 1)(q - 1)$.
4. Choose $e \in \mathbb{Z}$ with $\gcd(e, \phi(n)) = 1$. If $e$ is prime, then one must only check that $e \nmid \phi(n)$.
5. Find the multiplicative inverse $d = e^{-1} \mod \phi(n)$.
6. The public key is defined as the pair $(e, n)$, and the private key as $(d, n)$.

Given a message $M$ with $0 \leq M < n$,[2] we can now encrypt $M$, and decrypt the resulting ciphertext $C$, with the functions

$$E(M) \equiv M^e \mod n, \qquad \text{and} \qquad D(C) \equiv C^d \mod n,$$

respectively.

Note that encryption can be performed with knowledge of only the public key, while decryption requires the private key. Since the private key was constructed using $\phi(n)$, it is thus crucial to the security of the message that third parties remain unable to reconstruct $\phi(n)$. Knowledge of $\phi(n)$ can be used to factor $n$, computing $\phi(n)$ is no simpler than factoring $n$; thus, RSA remains secure as long as factorization of large numbers remains difficult.

## 2 Primality Testing

In order to factor general large numbers into their prime components, we must first know what the possible prime factors are. But this becomes increasingly difficult to show by brute force as the numbers get large. Below are several methods used to determine the primality of numbers of various sizes.

[1] Some encryption standards put extra requirements on $p$ and $q$, e.g., they must be far enough apart that factoring $n$ by Fermat's method is infeasible.

[2] It may seem unintuitive that a message would be an integer, but any text can be converted to such an integer using what are called *padding schemes*, probabilistic conversion systems that are consistent and reversible.

## 2.1 Trial Division

Clearly, a positive integer $n$ is prime if no positive integer $1 < k < n$ divides $n$. Hence, the most naïve test of the primality of $n$ is to test each $k$ for divisibility into $n$. To optimize this approach, we may make the following observations (proofs omitted):

**Theorem.** *If $n$ is composite, then there is some $k \in \mathbb{Z}^+$, with $1 < k \leq \sqrt{n}$, where $k \mid n$.*

**Theorem.** *If $p > 3$ is prime, then $p \equiv \pm 1 \mod 6$.*

Thus, to test the primality of $n$ by trial division, we need only check integers less than $\sqrt{n}$ of the form $6k \pm 1$, a total of about $\sqrt{n}/3$ divisions. Alternatively, if we know all the primes less than $\sqrt{n}$ (which is likely, or at least easy to find, for small $n$), we can test only those primes, for $\pi(\sqrt{n}) = O(\sqrt{n})$ divisions. Therefore, trial division requires

$$O(\sqrt{n}) \cdot O(D \ln^2 D) = O(n^{1/2} \ln n (\ln \ln n)^2)$$

operations.[3] Depending on the circumstance, this method could thus be useful for $n$ up to about $10^{10}$.

## 2.2 The Sieve of Eratosthenes

Perhaps the oldest algorithm to determine large quantities of primes is the sieve of Eratosthenes, presented in Nicomachus's *Introduction to Arithmetic* some 1900 years ago [6]. The process is as follows:

**Algorithm** (Sieve of Eratosthenes)**.** Given an upper bound $n$:

1. Create an array of $n$ Boolean values, all initialized to True. These represent the numbers 0 to $n - 1$.
2. Set elements 0 and 1 to False.
3. For $i = 2, 3, \ldots, \lceil \sqrt{n} \rceil$,[4] if element $i$ of the sieve is True, set all elements $ki < n$ to False.

The numbers marked True by the end of the algorithm are the primes contained in the list. Such a result is guaranteed by the observation that the first number marked True in the sublist $\{i + 1, \ldots, n\}$ after an iteration of Step 3 must be prime.[5]

[3] Note that here, and in all other complexities in this paper, $n$ is the number to be factored, not the number of digits in that number. Since the number of digits is $\log_{10} n$, we have approximated the number of digits $D$ by $D \sim \ln n$, which holds asymptotically.

[4] For $i > \sqrt{n}$, $ki \in \Omega \iff k < i$. But since we have crossed off all multiples of every number less than $i$, every multiple of $i$ must have already been crossed off.

[5] If this number were composite, then it would have a prime factor less than itself. But then it would be a multiple of a number that we have already checked!

~~2~~ 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~
11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~
~~21~~ ~~22~~ 23 ~~24~~ ~~25~~ ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~

Figure 1: Sieve of Eratosthenes for $n = 30$

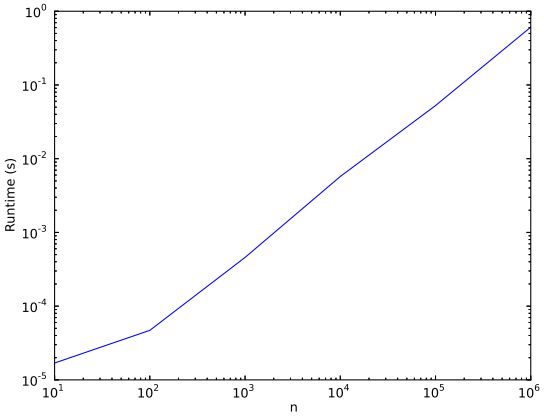


Figure 2: log-log plot of $n$ vs. runtime for sieve calculation

The sieve method is highly efficient in terms of its time complexity: the number of operations needed to perform the entire sieve is proportional to

$$\sum_{i=1}^{\pi(n)} \frac{n}{p_i} = n \ln \ln n + O(n) = O(n \ln \ln n),$$

or $O(\ln \ln n)$ per sieve element [4]. Notice that the performance here for the *entire sieve* is only slightly worse than trial division's performance for a single number, and in fact the sieve method is much more efficient per prime found for large $n$. This is a consequence of problem reduction: as we test each new prime $p_i$, we reduce the number of elements to test in future iterations by $n/p_1 \cdots p_i$.

However, the method's weakness lies in its poor space complexity; the sieve requires that a marker be stored for every number in the sieve, for a total of $O(n)$ bits of information. Thus, the sieve is only useful to us for relatively small numbers, say, less than $10^{12}$ (i.e., less than about 1 TB of storage). There are ways around this constraint, known as *sieve segmentations*, that split larger sieves into blocks, evaluate each block sequentially, and save only the primes from each block for evaluation in subsequent blocks. The space complexity for a segmentation is $O(\pi(n))$, extending the sieve's range to $O(10^{14})$.

# 3 Prime Factorization

Now that we have methods for determining the prime numbers, we can proceed with the task of decomposing numbers into their prime factors. In class, we covered several factorization methods, including trial division and Fermat's method, both deterministic algorithms with exponential runtime. In this section, we will cover a probabilistic algorithm known as the Pollard rho method, and explore in detail Pomerance's quadratic sieve method, which is subexponential. We will also discuss a method specifically useful for cracking multiple RSA keys.

## 3.1 The Pollard rho ($\rho$) Method

All of the other methods described in this paper are deterministic; given a particular input, they provide the same output no matter how many times they are run. In 1975, however, John Pollard used an observation about functions defined on discrete finite sets to create a *probabilistic* factorization method—one that may give different factors for $n$, determined randomly at runtime [7].

Pollard's observation is as follows: Let $n \in \mathbb{Z}^+$ and $p$ the least prime factor of $n$. If $\mathcal{S} = \{1, 2, \ldots, p-1\}$, then we can define a function $f = x^2 + a \mod p$ for some $a \in \mathcal{S}$ such that for any $s \in \mathcal{S}$, the sequence

$$s, f(s), f(f(s)), \ldots$$

will eventually begin to repeat itself (i.e., will be cyclic). We can conceptualize this cyclic behavior with the Greek letter $\rho$,[6] and a probabilistic argument will show that the repetition occurs within $O(\sqrt{p})$ iterations.

Under reasonable circumstances, we do not know the value of $p$, and so $f$ is impossible to compute. Pollard's breakthrough was to construct another function $F = x^2 + a \mod n$ that exhibits similar cyclic behavior as $f$ without recourse to the value of $p$. In fact, $F \equiv f \mod p$, and if we let $F^{(i)}(x)$ denote the action of $F$ on $x$ taken $i$ times, then there exists $i \geq 0$ such that $2i = O(\sqrt{p})$ and $F^{(i)}(s) \equiv F^{(2i)}(s) \mod p$. In that case, the number $g = \gcd(F^{(i)}(s) - F^{(2i)}(s), n)$ is a multiple of $p$. If $g \neq n$, then $g$ is a nontrivial factor.

**Algorithm** (Pollard $\rho$ method)**.** Given a composite integer $n$,

1. Randomly choose integers $a \in [1, n-3]$[7] and $s \in [0, n-1]$.
2. Define $F(x) = (x^2 + a) \mod n$.

[6] Imagine the tail of the $\rho$ depicting the non-cyclic part of the sequence, leading into the body of the $\rho$, the cyclic part.

[7] We could in theory allow $a \in [0, n-1]$, but it turns out that the additional choices of $a$ in this set perform poorly in the algorithm, i.e., we are only guaranteed a factor after more than $O(\sqrt{p})$ iterations, if at all. Thus, it behooves us to limit our choices.

3. Set $U = V = s$.
4. Iterate $U$ and $V$ as follows:
   - $U = F(U)$,
   - $V = F^{(2)}(V)$.
5. Calculate $g = \gcd(U - V, n)$.
6. If $g = 1$, go back to Step 4. If $g = n$, go back to Step 1.
7. Return $g$.

The algorithm is probabilistic in that the values of $a$ and $s$ are chosen randomly. It is possible that certain combinations of $a$ and $s$ work with prime factors other than $p$, and thus the result obtained by the algorithm will likely differ from run to run, and these results might not even share any factors between them.

Step 4 may be repeated up to $O(\sqrt{p})$ times, for an operation complexity of

$$O(p^{1/2}) \cdot O(N \ln N) \approx O(n^{1/4} \ln n \ln \ln n),$$

assuming $p$ is of maximum order proportional to $\sqrt{n}$.

## 3.2 The Quadratic Sieve

So far, the factorization methods discussed have run no faster than exponential time. But there are several factorization methods that are faster than this. We will discuss a method that runs in subexponential time—proportional to $O(e^{n^{\varepsilon}})$ for some $\varepsilon > 0$.

This method—the quadratic sieve method, created by Carl Pomerance in 1981 [8]—involves sieving integers for a quality called *smoothness*. A number is $B$-smooth if all of its prime factors are less than $B$. The quadratic sieve uses a series of $B$-smooth numbers to construct quadratic congruences modulo $n$ that result in some factor of $n$.

### The Quadratic Sieve

Suppose $n$ is an odd composite that is not a power. We will consider the sequence of values

$$x^2 \mod n \qquad \text{for} \qquad x = \lceil \sqrt{n} \rceil, \lceil \sqrt{n} \rceil + 1, \ldots$$

that are $B$-smooth for some $B$ of our choice. Let us put an extra condition on smoothness here: we will consider only the $x^2 - n$ whose prime factors are the primes less than $B$ that are quadratic residues modulo $n$. Each such prime $p_i$ will produce two integers $\pm a_i$, such that $a_i^2 \equiv n \mod p_i$; these are the *quadratic residues* of $n$ modulo $p_i$.

We shall sieve the above sequence for these primes. Notice that if $p_i \mid x^2 - n$, then $x \equiv \pm a_i \mod p_i$; thus, we can quickly divide out each $p_i$ from the $x^2 - n$ in our sequence. If some $x^2 - n$ is $B$-smooth, then after division by each prime power $p_i^a \leq B$, it will equal 1. This gives us a very quick way to determine $B$-smooth values $x_i$.

Suppose we find $\pi(B) + 1$ such $B$-smooth values $x_1, x_2, \ldots, x_k$. Then as a consequence of a result from linear algebra, the product of some subset of these $x_i$ will be a perfect square modulo $n$, i.e., for some $i, \ldots, j$,

$$u^2 = (x_i \ldots x_j)^2 \equiv a_i \cdots a_j = v^2 \mod n.$$

If $u \not\equiv \pm v \mod n$, then $n \mid (u + v)(u - v)$, but $n \nmid u + v$ and $n \nmid u - v$; hence, $\gcd(u + v, n)$ and $\gcd(u - v, n)$ must be factors of $n$.

**Algorithm** (Quadratic Sieve [4])**.** Given an odd composite integer $n$ that is not a power,

1. Choose a smoothness bound $B$. In most cases, the optimal $B \approx \lceil e^{.5\sqrt{\ln n \ln\ln n}} \rceil$.
2. Find the factor base $F = \{2\} \cup \{p \leq B \mid p \text{ odd prime and } (\frac{n}{p}) = 1\}$.
3. For each prime $p_i \in F$, find $\pm a_i \mod p_i$ such that $a_i^2 \equiv n \mod p_i$.
4. Sieve the sequence of values $x^2 - n$ for $x = \lceil \sqrt{n} \rceil, \lceil \sqrt{n} \rceil + 1, \ldots$ for $B$-smooth values, stopping when you have found $\pi(B) + 1$ of them. For each $B$-smooth value, save the pair $(x, x^2 - n)$.
5. Find the prime factorization $\prod p_i^{e_i}$ of each $B$-smooth value found, and set $v_i = (e_1, e_2, \ldots, e_{|F|})$.
6. Form a matrix in $\mathbb{Z}_2$ with each row $i$ being $v_i$ reduced modulo 2.
7. Find the null space for this matrix (i.e., a collection of vectors $V$ with entries from $\mathbb{Z}_2$ where $\sum V = \mathbf{0}$).
8. For each vector $V$ in the null space:
   (a) Compute $u = \prod_{i, V_i = 1} x_i$.
   (b) Compute $v = \sqrt{\prod_{i, V_i = 1} (x_i^2 - n)}$ from the prime factorizations of the values $x_i^2 - n$.
   (c) Compute $\gcd(u + v, n)$ and $\gcd(u - v, n)$. If each does not equal 1, return them.

The complexity for the quadratic sieve is dominated by the sieving process. We must find at most $\pi(B) + 1$ values that are each $B$-smooth, and perform operations on a $(\pi(B) + 1) \times B$ matrix; hence, the sieve's runtime is[8]

$$O(\pi(B)) \cdot O(B) \sim O(B^2) \approx O(e^{.5\sqrt{\ln n \ln\ln n}}).$$

[8]It should be noted that the complexity for the quadratic sieve is found only through an heuristic argument; a rigorous proof eludes us.

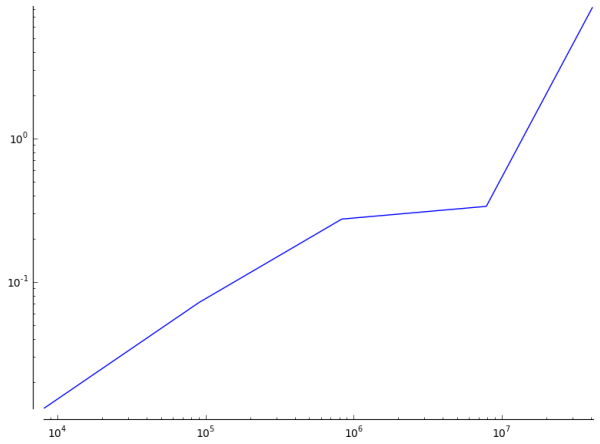


Figure 3: log-log plot of $n$ vs. runtime of the quadratic sieve

Figure 3 depicts a log-log plot of $n$ vs. the runtime of the quadratic sieve to factor $n$. Notice that it grows faster than linearly, implying that the quadratic sieve is at least superalgebraic, consistent with our claim that it is subexponential (the latter implies the former).

There is a modified form of the quadratic sieve, known as the number field sieve, that implements essentially the above process to find quadratic residues on a pair of polynomial rings $\mathbb{Z}[r_1]$ and $\mathbb{Z}[r_2]$ instead of on $\mathbb{Z}$, which is currently the fastest known factorization method [5]. Due to the complexity of the mathematics behind the details of its implementation, it has been omitted.

## 3.3 Factoring Multiple RSA Keys: The Batch GCD Method

In general, RSA keys are very difficult to factor: the largest RSA modulus factored by standard algorithms has 768 bits (232 decimal digits) [2], and the current standards for most high-level encryption services use 1024- or 2048-bit RSA moduli. However, if a large collection of RSA keys (say, $O(10^6)$ of them) are constructed using primes given by a faulty random-number generator, a significant number of those keys may share prime factors.

The batch GCD algorithm exploits this weakness in real-world RSA key generation [3]. Suppose two distinct RSA keys $N_1$ and $N_2$ share a factor, that is, $d = \gcd(N_1, N_2) > 1$; then, since $N_i = p_i q_i$ for prime $p_i$ and $q_i$, it must be the case that either $d = p_1$ or $d = q_1$. Thus, if we find that two keys share factors, we can quickly factor both keys using their GCD. The batch GCD process

performs this quickly on large sets of keys $\{N_1, N_2, \ldots, N_m\}$ by calculating:

$$\begin{gathered}\gcd(N_1, N_2 N_3 \ldots N_m),\\ \gcd(N_2, N_1 N_3 \ldots N_m),\\ \ldots\\ \gcd(N_m, N_1 N_2 \ldots N_{m-1}).\end{gathered}$$

**Algorithm** (Batch GCD)**.** Given a list $\{N_1, N_2, \ldots, N_m\}$ of integers,

1. Calculate $N = N_1 N_2 \cdots N_m$.
2. Create a list $G$ of length $m$.
3. For each $i = 1, 2, \ldots, m$:
   - Calculate $R_i = N \mod N_i^2$.
   - Set $G_i = \gcd(N_i, R_i/N_i) = \gcd(N_i, N/N_i)$.
4. Return $G$.

Notice that $\{N_i \mid G_i > 1\}$ is the set of keys that share a factor with some other key in the list. Most of the time, $G_i$ is prime and thus a nontrivial factor of $N_i$. By recognizing these prime $G_i$, we can factor the corresponding keys $N_i$.[9]

The batch GCD method requires calculating $m$ GCDs, for a complexity of

$$m \cdot O(N^2) = O(m \ln^2 n).$$

[9] In the less frequent case, $G_i$ is composite $\iff$ $G_i = N_i$. Then there exist some $N_j, N_k \in B$, with $i \neq j \neq k$, such that $p_i \mid N_j$ and $q_i \mid N_k$. It is possible to have some subset of $C \subseteq B$ where each $N_i \in C$ fits this criterion. In such a case, the elements of $C$ form one or more cyclic chains of factor sharing, e.g., $N_1 = p_1 p_2$, $N_2 = p_2 p_3$, and $N_3 = p_3 p_1$. Batch GCD does nothing to help us here. Luckily, this setup is relatively rare in practice.

# References

[1] Adleman, L.; R. Rivest; A. Shamir: "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, Volume 21, Issue 2, 120-126 (1978).

[2] Aoki, K.; Bos, J.; Franke, J.; Gaudry, P.; Kleinjung, T.; Kruppa, A.; Lenstra, A.; Montgomery, P.; Osvik, D.; Riele, H.; Thomé, E.; Timofeev, A.; Zimmermann, P.: "Factorization of a 768-bit RSA modulus," *Cryptology ePrint Archive*, Report 2010/006 (2010).

[3] Bernstein, D.; N. Heninger; T. Lange: "FactHacks: RSA factorization in the real world," facthacks.cr.yp.to (2012).

[4] Crandall, R.; C. Pomerance: *Prime Numbers: A Computational Perspective*, Second Edition (2005).

[5] Lenstra, A.; H. Lenstra; M. Manasse; J. Pollard: "The development of the number field sieve," *Lecture Notes in Mathematics*, Volume 1554, 11-42 (1993).

[6] Nicomachus: *Introduction to Arithmetic*, Volume I, 13 (year unknown).

[7] Pollard, J.M.: "A Monte Carlo method for factorization," *BIT Numerical Mathematics*, Volume 15, Issue 3, 331-334 (1975).

[8] Pomerance, C: "Smooth Numbers and the Quadratic Sieve," *Algorithmic Number Theory*, Volume 44, 69-81 (2008).