topics in number theory
Contents

Articles

Euclidean algorithm 1
Prime number 25
Miller–Rabin primality test 42
Modular arithmetic 46
Modular exponentiation 52
RSA (algorithm) 56

References

Article Sources and Contributors 66
Image Sources, Licenses and Contributors 68

Article Licenses

License 69
Euclidean algorithm

In mathematics, the **Euclidean algorithm**[^6], or **Euclid's algorithm**, is a method for computing the greatest common divisor (GCD) of two (usually positive) integers, also known as the greatest common factor (GCF) or highest common factor (HCF). It is named after the Greek mathematician Euclid, who described it in Books VII and X of his *Elements*.[^1]

The GCD of two positive integers is the largest integer that divides both of them without leaving a remainder (the GCD of two integers in general is defined in a more subtle way).

In its simplest form, Euclid's algorithm starts with a pair of positive integers, and forms a new pair that consists of the smaller number and the difference between the larger and smaller numbers. The process repeats until the numbers in the pair are equal. That number then is the greatest common divisor of the original pair of integers.

The main principle is that the GCD does not change if the smaller number is subtracted from the larger number. For example, the GCD of 252 and 105 is exactly the GCD of 147 ($= 252 - 105$) and 105. Since the larger of the two numbers is reduced, repeating this process gives successively smaller numbers, so this repetition will necessarily stop sooner or later — when the numbers are equal (if the process is attempted once more, one of the numbers will become 0).

The earliest surviving description of the Euclidean algorithm is in Euclid's *Elements* (c. 300 BC), making it one of the oldest numerical algorithms still in common use. The original algorithm was described only for natural numbers and geometric lengths (real numbers), but the algorithm was generalized in the 19th century to other types of numbers, such as Gaussian integers and polynomials in one variable. This led to modern abstract algebraic notions, such as Euclidean domains. The Euclidean algorithm has been generalized further to other mathematical structures, such as knots and multivariate polynomials.

The algorithm has many theoretical and practical applications. It may be used to generate almost all the most important traditional musical rhythms used in different cultures throughout the world. It is a key element of the RSA algorithm, a public-key encryption method widely used in electronic commerce. It is used to solve Diophantine equations, such as finding numbers that satisfy multiple congruences (Chinese remainder theorem) or multiplicative inverses of a finite field. It can also be used to construct continued fractions, in the Sturm chain method for finding real roots of a polynomial, and in several modern integer factorization algorithms. Finally, it is a basic tool for proving theorems in modern number theory, such as Lagrange's four-square theorem and the fundamental theorem of arithmetic (unique factorization).
If implemented using remainders of Euclidean division rather than subtractions, Euclid's algorithm computes the GCD of large numbers efficiently: it never requires more division steps than five times the number of digits (in base 10) of the smaller integer. This was proved by Gabriel Lamé in 1844, and marks the beginning of computational complexity theory. Methods for improving the algorithm's efficiency were developed in the 20th century.

By reversing the steps in the Euclidean algorithm, the GCD can be expressed as a sum of the two original numbers each multiplied by a positive or negative integer, e.g., the GCD of 252 and 105 is 21, and 21 = [5 × 105] + [(-2) × 252]. This important property is known as Bézout's identity.

### Background — Greatest common divisor

The Euclidean algorithm calculates the greatest common divisor (GCD) of two natural numbers \( a \) and \( b \). The greatest common divisor \( g \) is the largest natural number that divides both \( a \) and \( b \) without leaving a remainder. Synonyms for the GCD include the greatest common factor (GCF), the highest common factor (HCF), and the greatest common measure (GCM). The greatest common divisor is often written as \( \gcd(a, b) \) or, more simply, as \( (a, b) \), although the latter notation is also used for other mathematical concepts, such as two-dimensional vectors.

If \( \gcd(a, b) = 1 \), then \( a \) and \( b \) are said to be coprime (or relatively prime). This property does not imply that \( a \) or \( b \) are themselves prime numbers. For example, neither 6 nor 35 is a prime number, since they both have two prime factors: \( 6 = 2 \times 3 \) and \( 35 = 5 \times 7 \). Nevertheless, 6 and 35 are coprime. No natural number other than 1 divides both 6 and 35, since they have no prime factors in common.

Let \( g = \gcd(a, b) \). Since \( a \) and \( b \) are both multiples of \( g \), they can be written \( a = mg \) and \( b = ng \), and there is no larger number \( G > g \) for which this is true. The natural numbers \( m \) and \( n \) must be coprime, since any common factor could be factored out of \( m \) and \( n \) to make \( g \) greater. Thus, any other number \( c \) that divides both \( a \) and \( b \) must also divide \( g \). The greatest common divisor \( g \) of \( a \) and \( b \) is the unique (positive) common divisor of \( a \) and \( b \) that is divisible by any other common divisor \( c \).

The GCD can be visualized as follows. Consider a rectangular area \( a \) by \( b \), and any common divisor \( c \) that divides both \( a \) and \( b \) exactly. The sides of the rectangle can be divided into segments of length \( c \), which divides the rectangle into a grid of squares of side length \( c \). The greatest common divisor \( g \) is the largest value of \( c \) for which this is possible. For illustration, a 24-by-60 rectangular area can be divided into a grid of: 1-by-1 squares, 2-by-2 squares, 3-by-3 squares, 4-by-4 squares, 6-by-6 squares or 12-by-12 squares. Therefore, \( 12 \) is the greatest common divisor of 24 and 60. A 24-by-60 rectangular area can be divided into a grid of 12-by-12 squares, with two squares along one edge \( (24/12 = 2) \) and five squares along the other \( (60/12 = 5) \).

The GCD of two numbers \( a \) and \( b \) is the product of the prime factors shared by the two numbers, where a same prime factor can be used multiple times, but only as long as the product of these factors divides both \( a \) and \( b \). For example, since 1386 can be factored into \( 2 \times 3 \times 3 \times 7 \times 11 \), and 3213 can be factored into \( 3 \times 3 \times 3 \times 7 \times 17 \), the greatest common divisor of 1386 and 3213 equals \( 63 = 3 \times 3 \times 7 \), the product of their shared prime factors. If two numbers have no prime factors in common, their greatest common divisor is 1 (obtained here as an instance of the empty product), in other words they are coprime. A key

A 24-by-60 rectangle is covered with ten 12-by-12 square tiles, where 12 is the GCD of 24 and 60. More generally, an \( a \)-by-\( b \) rectangle can be covered with square tiles of side-length \( c \) only if \( c \) is a common divisor of \( a \) and \( b \).
advantage of the Euclidean algorithm is that it can find the GCD efficiently without having to compute the prime factors. Factorization of large integers is believed to be a computationally very difficult problem, and the security of many modern cryptography systems is based upon its infeasibility.

Another definition of the GCD is helpful in advanced mathematics, particularly ring theory. The greatest common divisor \( g \) of two nonzero numbers \( a \) and \( b \) is also their smallest positive integral linear combination, that is, the smallest positive number of the form \( ua + vb \) where \( u \) and \( v \) are integers. The set of all integral linear combinations of \( a \) and \( b \) is actually the same as the set of all multiples of \( g \) (\( mg \), where \( m \) is an integer). In modern mathematical language, the ideal generated by \( a \) and \( b \) is the ideal generated by \( g \) alone (an ideal generated by a single element is called a principal ideal, and all ideals of the integers are principal ideals). Some properties of the GCD are in fact easier to see with this description, for instance the fact that any common divisor of \( a \) and \( b \) also divides the GCD (it divides both terms of \( ua + vb \)). The equivalence of this GCD definition with the other definitions is described below.

The GCD of three or more numbers equals the product of the prime factors common to all the numbers, but it can also be calculated by repeatedly taking the GCDs of pairs of numbers. For example,

\[
gcd(a, b, c) = gcd(a, gcd(b, c)) = gcd(gcd(a, b), c) = gcd(gcd(a, c), b).
\]

Thus, Euclid's algorithm, which computes the GCD of two integers, suffices to calculate the GCD of arbitrarily many integers.

**Description**

The simple form of Euclid's algorithm uses only subtraction and comparison. Starting with a pair of positive integers, form a new pair consisting of the smaller number and the difference between the larger number and the smaller number. This process repeats until the numbers in the new pair are equal to each other; that value is the greatest common divisor of the original pair. If one number is much smaller than the other, many subtraction steps will be needed before the larger number is reduced to a value less than or equal to the other number in the pair.

The common form of Euclid's algorithm replaces subtracting the small positive number from the big number (possibly many times) with finding the remainder in long division. This form of Euclid's algorithm also starts with a pair of positive integers, then forms a new pair consisting of the smaller number and the remainder obtained by dividing the larger number by the smaller number. The process repeats until one number is zero. The other number then is the greatest common divisor of the original pair.

**Procedure**

The Euclidean algorithm proceeds in a series of steps such that the output of each step is used as an input for the next one. Let \( k \) be an integer that counts the steps of the algorithm, starting with zero. Thus, the initial step corresponds to \( k = 0 \), the next step corresponds to \( k = 1 \), and so on.

Each step begins with two nonnegative remainders \( r_{k-1} \) and \( r_{k-2} \). Since the algorithm ensures that the remainders decrease steadily with every step, \( r_{k-1} \) is less than its predecessor \( r_{k-2} \). The goal of the \( k \)th step is to find a quotient \( q_k \) and remainder \( r_k \) such that the equation is satisfied

\[
r_{k-2} = q_k \cdot r_{k-1} + r_k
\]

where \( r_k < r_{k-1} \). In other words, multiples of the smaller number \( r_{k-1} \) are subtracted from the larger number \( r_{k-2} \) until the remainder is smaller than \( r_{k-1} \).

In the initial step \( (k = 0) \), the remainders \( r_{-2} \) and \( r_{-1} \) equal \( a \) and \( b \), the numbers for which the GCD is sought. In the next step \( (k = 1) \), the remainders equal \( b \) and the remainder \( r_0 \) of the initial step, and so on. Thus, the algorithm can be written as a sequence of equations

\[
a = q_0 \cdot b + r_0
\]

\[
b = q_1 \cdot r_0 + r_1
\]
\begin{align*}
r_0 &= q_2 r_1 + r_2 \\
r_1 &= q_3 r_2 + r_3 \\
&\ldots
\end{align*}

If \( a \) is smaller than \( b \), the first step of the algorithm swaps the numbers. For example, if \( a < b \), the initial quotient \( q_0 \) equals zero, and the remainder \( r_0 \) is \( a \). Thus, \( r_k \) is smaller than its predecessor \( r_{k-1} \) for all \( k \geq 0 \).

Since the remainders decrease with every step but can never be negative, a remainder \( r_N \) must eventually equal zero, at which point the algorithm stops. The final nonzero remainder \( r_{N-1} \) is the greatest common divisor of \( a \) and \( b \). The number \( N \) cannot be infinite because there are only a finite number of nonnegative integers between the initial remainder \( r_0 \) and zero.

**Proof of validity**

The validity of the Euclidean algorithm can be proven by a two-step argument. In the first step, the final nonzero remainder \( r_{N-1} \) is shown to divide both \( a \) and \( b \). Since it is a common divisor, it must be less than or equal to the greatest common divisor \( g \). In the second step, it is shown that any common divisor of \( a \) and \( b \), including \( g \), must divide \( r_{N-1} \); therefore, \( g \) must be less than or equal to \( r_{N-1} \). These two conclusions are inconsistent unless \( r_{N-1} = g \).

To demonstrate that \( r_{N-1} \) divides both \( a \) and \( b \) (the first step), \( r_{N-1} \) divides its predecessor \( r_{N-2} \)

\[ r_{N-2} = q_{N-1} r_{N-1} \]

since the final remainder \( r_N \) is zero. \( r_{N-1} \) also divides its next predecessor \( r_{N-3} \)

\[ r_{N-3} = q_{N-1} r_{N-2} + r_{N-1} \]

because it divides both terms on the right-hand side of the equation. Iterating the same argument, \( r_{N-1} \) divides all the preceding remainders, including \( a \) and \( b \). None of the preceding remainders \( r_{N-2}, r_{N-3}, \) etc. divide \( a \) and \( b \), since they leave a remainder. Since \( r_{N-1} \) is a common divisor of \( a \) and \( b \), \( r_{N-1} \leq g \).

In the second step, any natural number \( c \) that divides both \( a \) and \( b \) (in other words, any common divisor of \( a \) and \( b \)) divides the remainders \( r_k \). By definition, \( a \) and \( b \) can be written as multiples of \( c \): \( a = mc \) and \( b = nc \), where \( m \) and \( n \) are natural numbers. Therefore, \( c \) divides the initial remainder \( r_0 \) since \( r_0 = a - q_0 b = mc - q_0 nc = (m - q_0 n)c \). An analogous argument shows that \( c \) also divides the subsequent remainders \( r_1, r_2, \) etc. Therefore, the greatest common divisor \( g \) must divide \( r_{N-1} \), which implies that \( g \leq r_{N-1} \). Since the first part of the argument showed the reverse \( (r_{N-1} \leq g) \), it follows that \( g = r_{N-1} \). Thus, \( g \) is the greatest common divisor of all the succeeding pairs:

\[ g = \gcd(a, b) = \gcd(b, r_0) = \gcd(r_0, r_1) = \ldots = \gcd(r_{N-2}, r_{N-1}) = r_{N-1} \]
Euclidean algorithm

Worked example

For illustration, the Euclidean algorithm can be used to find the greatest common divisor of \(a = 1071\) and \(b = 462\). To begin, multiples of 462 are subtracted from 1071 until the remainder is less than 462. Two such multiples can be subtracted \((q_0 = 2)\), leaving a remainder of 147

\[1071 = 2 \times 462 + 147.\]

Then multiples of 147 are subtracted from 462 until the remainder is less than 147. Three multiples can be subtracted \((q_1 = 3)\), leaving a remainder of 21

\[462 = 3 \times 147 + 21.\]

Then multiples of 21 are subtracted from 147 until the remainder is less than 21. Seven multiples can be subtracted \((q_2 = 7)\), leaving no remainder

\[147 = 7 \times 21 + 0.\]

Since the last remainder is zero, the algorithm ends with 21 as the greatest common divisor of 1071 and 462. This agrees with the \(\text{gcd}(1071, 462)\) found by prime factorization above. In tabular form, the steps are

<table>
<thead>
<tr>
<th>Step (k)</th>
<th>Equation</th>
<th>Quotient and remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(1071 = q_0 \cdot 462 + r_0)</td>
<td>(q_0 = 2) and (r_0 = 147)</td>
</tr>
<tr>
<td>1</td>
<td>(462 = q_1 \cdot 147 + r_1)</td>
<td>(q_1 = 3) and (r_1 = 21)</td>
</tr>
<tr>
<td>2</td>
<td>(147 = q_2 \cdot 21 + r_2)</td>
<td>(q_2 = 7) and (r_2 = 0); algorithm ends</td>
</tr>
</tbody>
</table>

Visualization

The Euclidean algorithm can be visualized in terms of the tiling analogy given above for the greatest common divisor. Assume that we wish to cover an \(a\)-by-\(b\) rectangle with square tiles exactly, where \(a\) is the larger of the two numbers. We first attempt to tile the rectangle using \(b\)-by-\(b\) square tiles; however, this leaves an \(r_0\)-by-\(b\) residual rectangle untiled, where \(r_0 < b\). We then attempt to tile the residual rectangle with \(r_0\)-by-\(r_0\) square tiles. This leaves a second residual rectangle \(r_1\)-by-\(r_0\), which we attempt to tile using \(r_1\)-by-\(r_1\) square tiles, and so on. The sequence ends when there is no residual rectangle, i.e., when the square tiles cover the previous residual rectangle exactly. The
length of the sides of the smallest square tile is the GCD of the dimensions of the original rectangle. For example, the smallest square tile in the adjacent figure is 21-by-21 (shown in red), and 21 is the GCD of 1071 and 462, the dimensions of the original rectangle (shown in green).

**Euclidean division**

At every step $k$, the Euclidean algorithm computes a quotient $q_k$ and remainder $r_k$ from two numbers $r_{k-1}$ and $r_{k-2}$

$$r_{k-2} = q_k r_{k-1} + r_k$$

where the magnitude of $r_k$ is strictly less than that of $r_{k-1}$. The theorem which underlies the definition of the Euclidean division ensures that such a quotient and remainder always exist and are unique.

In Euclid's original version of the algorithm, the quotient and remainder are found by repeated subtraction; that is, $r_{k-1}$ is subtracted from $r_{k-2}$ repeatedly until the remainder $r_k$ is smaller than $r_{k-1}$. After that $r_k$ and $r_{k-1}$ are exchanged and the process is iterated. Euclidean division reduces all the steps between two exchanges into a single step, which is thus more efficient. Moreover, the quotients are not needed, thus one may replace Euclidean division by the modulo operation, which gives only the remainder. Thus the iteration of the Euclidean algorithm becomes simply

$$r_k = r_{k-2} \mod r_{k-1}.$$  

**Implementations**

Implementations of the algorithm may be expressed in pseudocode. For example, the division-based version may be programmed as:

```plaintext
function gcd(a, b)
    while b ≠ 0
        t := b
        b := a mod b
        a := t
    return a
```

At the beginning of the $k$th iteration, the variable $b$ holds the latest remainder $r_{k-1}$, whereas the variable $a$ holds its predecessor, $r_{k-2}$. The step $b := a \mod b$ is equivalent to the above recursion formula $r_k \equiv r_{k-2} \mod r_{k-1}$. The dummy variable $t$ holds the value of $r_{k-1}$ while the next remainder $r_k$ is being calculated. At the end of the loop iteration, the variable $b$ holds the remainder $r_k$, whereas the variable $a$ holds its predecessor, $r_{k-1}$.

In the subtraction-based version which was Euclid's original version, the remainder calculation ($b = a \mod b$) is replaced by repeated subtraction. Contrary to the division-based version, which works with arbitrary integers as input, the subtraction-based version supposes that the input consists of positive integers and stops when $a = b$:

```plaintext
function gcd(a, b)
    while a ≠ b
        if a > b
            a := a - b
        else
            b := b - a
    return a
```

The variables $a$ and $b$ alternate holding the previous remainders $r_{k-1}$ and $r_{k-2}$. Assume that $a$ is larger than $b$ at the beginning of an iteration; then $a$ equals $r_{k-2}$, since $r_{k-2} > r_{k-1}$. During the loop iteration, $a$ is reduced by multiples of the previous remainder $b$ until $a$ is smaller than $b$. Then $a$ is the next remainder $r_k$. Then $b$ is reduced by multiples of $a$ until it is again smaller than $a$, giving the next remainder $r_{k+1}$, and so on.
The recursive version is based on the equality of the GCDs of successive remainders and the stopping condition \( \text{gcd}(r_{N-1}, 0) = r_{N-1} \).

```plaintext
function \text{gcd}(a, b)
  if \( b = 0 \)
    return \( a \)
  else
    return \text{gcd}(b, a \mod b)
```

For illustration, the \( \text{gcd}(1071, 462) \) is calculated from the equivalent \( \text{gcd}(462, 1071 \mod 462) = \text{gcd}(462, 147) \). The latter GCD is calculated from the \( \text{gcd}(147, 462 \mod 147) = \text{gcd}(147, 21) \), which in turn is calculated from the \( \text{gcd}(21, 147 \mod 21) = \text{gcd}(21, 0) = 21 \).

**Method of least absolute remainders**

In another version of Euclid's algorithm, the quotient at each step is increased by one if the resulting negative remainder is smaller in magnitude than the typical positive remainder. Previously, the equation

\[
r_{k-2} = q_k r_{k-1} + r_k
\]

assumed that \( |r_{k-1}| > r_k > 0 \). However, an alternative negative remainder \( e_k \) can be computed:

\[
r_{k-2} = (q_k + 1) r_{k-1} + e_k
\]

if \( r_{k-1} > 0 \) or

\[
r_{k-2} = (q_k - 1) r_{k-1} + e_k
\]

if \( r_{k-1} < 0 \).

If \( |e_k| < |r_k| \), then \( r_k \) is replaced by \( e_k \). As \( |r_{k-1}| = r_k - e_k \), this new \( r_k \) satisfies

\[
|e_k| < |r_{k-1}| / 2.
\]

Leopold Kronecker has shown that this version requires the fewest number of steps of any version of Euclid's algorithm.
Historical development

The Euclidean algorithm is one of the oldest algorithms still in common use. It appears in Euclid's *Elements* (c. 300 BC), specifically in Book 7 (Propositions 1–2) and Book 10 (Propositions 2–3). In Book 7, the algorithm is formulated for integers, whereas in Book 10, it is formulated for lengths of line segments. (In modern usage, one would say it was formulated there for real numbers. But lengths, areas, and volumes, represented as real numbers in modern usage, are not measured in the same units and there is no natural unit of length, area, or volume, and the concept of real numbers was unknown at that time.) The latter algorithm is geometrical. The GCD of two lengths $a$ and $b$ corresponds to the greatest length $g$ that measures $a$ and $b$ evenly; in other words, the lengths $a$ and $b$ are both integer multiples of the length $g$.

The algorithm was probably not discovered by Euclid, who compiled results from earlier mathematicians in his *Elements*. The mathematician and historian B. L. van der Waerden suggests that Book VII derives from a textbook on number theory written by mathematicians in the school of Pythagoras. The algorithm was probably known by Eudoxus of Cnidus (about 375 BC). The algorithm may even pre-date Eudoxus, judging from the use of the technical term ἀνθυφαίρεσις (anthyphairesis, reciprocal subtraction) in works by Euclid and Aristotle.

Centuries later, Euclid's algorithm was discovered independently both in India and in China, primarily to solve Diophantine equations that arise in astronomy and making accurate calendars. In the late 5th century, the Indian mathematician and astronomer Aryabhata described the algorithm as the "pulverizer", perhaps because of its effectiveness in solving Diophantine equations. Although a special case of the Chinese remainder theorem had already been described by Chinese mathematician and astronomer Sun Tzu, the general solution was published by Qin Jiushao in his 1247 book *Shushu Jiuzhang* (數書九章 Mathematical Treatise in Nine Sections). The Euclidean algorithm was first described in Europe in the second edition of Bachet's *Problèmes plaisants et délectables* (Pleasant and enjoyable problems, 1624). In Europe, it was likewise used to solve Diophantine equations and in developing continued fractions. The extended Euclidean algorithm was published by the English mathematician Nicholas Saunderson, who attributed it to Roger Cotes as a method for computing continued fractions efficiently.

In the 19th century, the Euclidean algorithm led to the development of new number systems, such as Gaussian integers and Eisenstein integers. In 1815, Carl Gauss used the Euclidean algorithm to demonstrate unique factorization of Gaussian integers, although his work was first published in 1832. Gauss mentioned the algorithm in his *Disquisitiones Arithmeticae* (published 1801), but only as a method for continued fractions. Peter Gustav Lejeune Dirichlet seems to have been the first to describe the Euclidean algorithm as the basis for much of number theory. Lejeune Dirichlet noted that many results of number theory, such as unique factorization, would hold true for any other system of numbers to which the Euclidean algorithm could be applied. Lejeune Dirichlet's lectures on number theory were edited and extended by Richard Dedekind, who used Euclid's algorithm to study algebraic integers, a new general type of number. For example, Dedekind was the first to prove Fermat's two-square theorem using the unique factorization of Gaussian integers. Dedekind also defined the concept of a Euclidean domain, a number system in which a generalized version of the Euclidean algorithm can be defined (as described below). In the closing
decades of the 19th century, however, the Euclidean algorithm gradually became eclipsed by Dedekind's more general theory of ideals.

"[The Euclidean algorithm] is the granddaddy of all algorithms, because it is the oldest nontrivial algorithm that has survived to the present day."


Other applications of Euclid's algorithm were developed in the 19th century. In 1829, Charles Sturm showed that the algorithm was useful in the Sturm chain method for counting the real roots of polynomials in any given interval.

The Euclidean algorithm was the first integer relation algorithm, which is a method for finding integer relations between commensurate real numbers. Several novel integer relation algorithms have been developed in recent years, such as the Ferguson–Forcade algorithm (1979) of Helaman Ferguson and R.W. Forcade, and its relatives, the LLL algorithm, the HJLS algorithm, and the PSLQ algorithm.

In 1969, Cole and Davie developed a two-player game based on the Euclidean algorithm, called The Game of Euclid, which has an optimal strategy. The players begin with two piles of $a$ and $b$ stones. The players take turns removing $m$ multiples of the smaller pile from the larger. Thus, if the two piles consist of $x$ and $y$ stones, where $x$ is larger than $y$, the next player can reduce the larger pile from $x$ stones to $x - my$ stones, as long as the latter is a nonnegative integer. The winner is the first player to reduce one pile to zero stones.

Mathematical applications

Bézout's identity

Bézout's identity states that the greatest common divisor $g$ of two integers $a$ and $b$ can be represented as a linear sum of the original two numbers $a$ and $b$. In other words, it is always possible to find integers $s$ and $t$ such that $g = sa + tb$.

The integers $s$ and $t$ can be calculated from the quotients $q_0, q_1$, etc. by reversing the order of equations in Euclid's algorithm. Beginning with the next-to-last equation, $g$ can be expressed in terms of the quotient $q_{N-1}$ and the two preceding remainders, $r_{N-2}$ and $r_{N-3}$:

$$g = r_{N-1} = r_{N-3} - q_{N-1} r_{N-2}$$

Those two remainders can be likewise expressed in terms of their quotients and preceding remainders,

$$r_{N-2} = r_{N-4} - q_{N-2} r_{N-3}$$

$$r_{N-3} = r_{N-5} - q_{N-3} r_{N-4}$$

Substituting these formulae for $r_{N-2}$ and $r_{N-3}$ into the first equation yields $g$ as a linear sum of the remainders $r_{N-4}$ and $r_{N-5}$. The process of substituting remainders by formulae involving their predecessors can be continued until the original numbers $a$ and $b$ are reached

$$r_2 = r_0 - q_2 r_1$$

$$r_1 = b - q_1 r_0$$

$$r_0 = a - q_0 b.$$ 

After all the remainders $r_0, r_1$, etc. have been substituted, the final equation expresses $g$ as a linear sum of $a$ and $b$: $g = sa + tb$. Bézout's identity, and therefore the previous algorithm, can both be generalized to the context of Euclidean domains.
Principal ideals and related problems

Bézout's identity provides yet another definition of the greatest common divisor $g$ of two numbers $a$ and $b$. Consider the set of all numbers $ua + vb$, where $u$ and $v$ are any two integers. Since $a$ and $b$ are both divisible by $g$, every number in the set is divisible by $g$. In other words, every number of the set is an integer multiple of $g$. This is true for every common divisor of $a$ and $b$. However, unlike other common divisors, the greatest common divisor is a member of the set; by Bézout's identity, choosing $u = s$ and $v = t$ gives $g$. A smaller common divisor cannot be a member of the set, since every member of the set must be divisible by $g$. Conversely, any multiple $m$ of $g$ can be obtained by choosing $u = ms$ and $v = mt$, where $s$ and $t$ are the integers of Bézout's identity. This may be seen by multiplying Bézout's identity by $m$,

$$ mg = msa + mtb. $$

Therefore, the set of all numbers $ua + vb$ is equivalent to the set of multiples $m$ of $g$. In other words, the set of all possible sums of integer multiples of two numbers ($a$ and $b$) is equivalent to the set of multiples of $\gcd(a, b)$. The GCD is said to be the generator of the ideal of $a$ and $b$. This GCD definition led to the modern abstract algebraic concepts of a principal ideal (an ideal generated by a single element) and a principal ideal domain (a domain in which every ideal is a principal ideal).

Certain problems can be solved using this result. For example, consider two measuring cups of volume $a$ and $b$. By adding/subtracting $u$ multiples of the first cup and $v$ multiples of the second cup, any volume $ua + vb$ can be measured out. These volumes are all multiples of $g = \gcd(a, b)$.

Extended Euclidean algorithm

The integers $s$ and $t$ of Bézout's identity can be computed efficiently using the extended Euclidean algorithm. This extension adds two recursive equations to Euclid's algorithm

$$ s_k = s_{k-2} - q_k s_{k-1} $$
$$ t_k = t_{k-2} - q_k t_{k-1} $$

with the starting values

$$ s_{-2} = 1, t_{-2} = 0 $$
$$ s_{-1} = 0, t_{-1} = 1. $$

Using this recursion, Bézout's integers $s$ and $t$ are given by $s = s_N$ and $t = t_N$, where $N$ is the step on which the algorithm terminates with $r_N = 0$. The validity of this approach can be shown by induction. Assume that the recursion formula is correct up to step $k - 1$ of the algorithm; in other words, assume that

$$ r_j = s_j a + t_j b $$

for all $j$ less than $k$. The $k$th step of the algorithm gives the equation

$$ r_k = r_{k-2} - q_k r_{k-1}. $$

Since the recursion formula has been assumed to be correct for $r_{k-2}$ and $r_{k-1}$, they may be expressed in terms of the corresponding $s$ and $t$ variables

$$ r_k = (s_{k-2} a + t_{k-2} b) - q_k (s_{k-1} a + t_{k-1} b). $$

Rearranging this equation yields the recursion formula for step $k$, as required

$$ r_k = s_k a + t_k b = (s_{k-2} - q_k s_{k-1}) a + (t_{k-2} - q_k t_{k-1}) b. $$
Matrix method
The integers \( s \) and \( t \) can also be found using an equivalent matrix method. The sequence of equations of Euclid's algorithm
\[
\begin{align*}
a &= q_0 b + r_0 \\
b &= q_1 r_0 + r_1 \\
\vdots \\
r_{N-2} &= q_N r_{N-1} + 0
\end{align*}
\]
can be written as a product of 2-by-2 quotient matrices multiplying a two-dimensional remainder vector
\[
\begin{pmatrix}
a \\ b
\end{pmatrix} =
\begin{pmatrix}
q_0 & 1 \\ 1 & 0
\end{pmatrix}
\begin{pmatrix}
r_0 \\ r_1
\end{pmatrix} =
\cdots =
\prod_{i=0}^{N}
\begin{pmatrix}
q_i & 1 \\ 1 & 0
\end{pmatrix}
\begin{pmatrix}
r_N \\ 0
\end{pmatrix}
\]
Let \( M \) represent the product of all the quotient matrices
\[
M = \begin{pmatrix}
m_{11} & m_{12} \\ m_{21} & m_{22}
\end{pmatrix} = \prod_{i=0}^{N}
\begin{pmatrix}
q_i & 1 \\ 1 & 0
\end{pmatrix} =
\begin{pmatrix}
q_0 & 1 \\ 1 & 0
\end{pmatrix}
\begin{pmatrix}
q_1 & 1 \\ 1 & 0
\end{pmatrix}
\cdots
\begin{pmatrix}
q_N & 1 \\ 1 & 0
\end{pmatrix}
\]
This simplifies the Euclidean algorithm to the form
\[
\begin{pmatrix}
a \\ b
\end{pmatrix} = M \begin{pmatrix}
r_{N-1} \\ 0
\end{pmatrix} = M \begin{pmatrix}
g \\ 0
\end{pmatrix}
\]
To express \( g \) as a linear sum of \( a \) and \( b \), both sides of this equation can be multiplied by the inverse of the matrix \( M \). The determinant of \( M \) equals \((-1)^{N+1}\), since it equals the product of the determinants of the quotient matrices, each of which is negative one. Since the determinant of \( M \) is never zero, the vector of the final remainders can be solved using the inverse of \( M \)
\[
\begin{pmatrix}
g \\ 0
\end{pmatrix} = M^{-1} \begin{pmatrix}
a \\ b
\end{pmatrix} = (-1)^{N+1} \begin{pmatrix}
m_{22} & -m_{12} \\ -m_{21} & m_{11}
\end{pmatrix} \begin{pmatrix}
a \\ b
\end{pmatrix}
\]
Since the top equation gives
\[
g = (-1)^{N+1} (m_{22} a - m_{12} b)
\]
the two integers of Bézout's identity are \( s = (-1)^{N+1} m_{22} \) and \( t = (-1)^{N} m_{12} \). The matrix method is as efficient as the equivalent recursion, with two multiplications and two additions per step of the Euclidean algorithm.

Euclid's lemma and unique factorization
Bézout's identity is essential to many applications of Euclid's algorithm, such as demonstrating the unique factorization of numbers into prime factors. To illustrate this, suppose that a number \( L \) can be written as a product of two factors \( u \) and \( v \), that is, \( L = uv \). If another number \( w \) also divides \( L \) but is coprime with \( u \), then \( w \) must divide \( v \), by the following argument: If the greatest common divisor of \( u \) and \( w \) is 1, then integers \( s \) and \( t \) can be found such that
\[
1 = su + tw
\]
by Bézout's identity. Multiplying both sides by \( v \) gives the relation
\[
v = suv + twv = sL + twv
\]
Since \( w \) divides both terms on the right-hand side, it must also divide the left-hand side, \( v \). This result is known as Euclid's lemma. Specifically, if a prime number divides \( L \), then it must divide at least one factor of \( L \). Conversely, if a number \( w \) is coprime to each of a series of numbers \( a_1, a_2, \ldots, a_n \), then \( w \) is also coprime to their product, \( a_1 \times a_2 \times \cdots \times a_n \).

Euclid's lemma suffices to prove that every number has a unique factorization into prime numbers. To see this, assume the contrary, that there are two independent factorizations of \( L \) into \( m \) and \( n \) prime factors, respectively
Euclidean algorithm

\[ L = p_1 p_2 \cdots p_m = q_1 q_2 \cdots q_n \]

Since each prime \( p \) divides \( L \) by assumption, it must also divide one of the \( q \) factors; since each \( q \) is prime as well, it must be that \( p = q \). Iteratively dividing by the \( p \) factors shows that each \( p \) has an equal counterpart \( q \); the two prime factorizations are identical except for their order. The unique factorization of numbers into primes has many applications in mathematical proofs, as shown below.

**Linear Diophantine equations**

Diophantine equations are equations in which the solutions are restricted to integers; they are named after the 3rd-century Alexandrian mathematician Diophantus. A typical *linear* Diophantine equation seeks integers \( x \) and \( y \) such that

\[ ax + by = c \]

where \( a, b \) and \( c \) are given integers. This can be written as an equation for \( x \) in modular arithmetic

\[ ax \equiv c \pmod{b} \]

Let \( g \) be the greatest common divisor of \( a \) and \( b \). Both terms in \( ax + by \) are divisible by \( g \); therefore, \( c \) must also be divisible by \( g \), or the equation has no solutions. By dividing both sides by \( c/g \), the equation can be reduced to Bezout’s identity

\[ sa + tb = g \]

where \( s \) and \( t \) can be found by the extended Euclidean algorithm. This provides one solution to the Diophantine equation, \( x_1 = s (c/g) \) and \( y_1 = t (c/g) \).

In general, a linear Diophantine equation has no solutions, or an infinite number of solutions. To find the latter, consider two solutions, \( (x_1, y_1) \) and \( (x_2, y_2) \)

\[ ax_1 + by_1 = c = ax_2 + by_2 \]

or equivalently

\[ a(x_1 - x_2) = b(y_2 - y_1) \]

Therefore, the smallest difference between two \( x \) solutions is \( b/g \), whereas the smallest difference between two \( y \) solutions is \( a/g \). Thus, the solutions may be expressed as

\[ x = x_1 - bt \]
\[ y = y_1 + at \]

By allowing \( t \) to vary over all possible integers, an infinite family of solutions can be generated from a single solution \( (x_1, y_1) \). If the solutions are required to be *positive* integers \( (x > 0, y > 0) \), only a finite number of solutions may be possible. This restriction on the acceptable solutions allows systems of Diophantine equations to be solved with more unknowns than equations; this is impossible for a system of linear equations when the solutions can be any real number.

**Multiplicative inverses and the RSA algorithm**

A finite field is a set of numbers with four generalized operations. The operations are called addition, subtraction, multiplication and division and have their usual properties, such as commutativity, associativity and distributivity. An example of a finite field is the set of 13 numbers \{0, 1, 2, …, 12\} using modular arithmetic. In this field, the results of any mathematical operation (addition/subtraction/multiplication/division) is reduced modulo 13; that is, multiples of 13 are added or subtracted until the result is brought within the range 0–12. For example, the result of
Euclidean algorithm

$5 \times 7 = 35 \mod 13 = 9$. Such finite fields can be defined for any prime $p$; using more sophisticated definitions, they can also be defined for any power $m$ of a prime $p^m$. Finite fields are often called Galois fields, and are abbreviated as $\text{GF}(p)$ or $\text{GF}(p^m)$.

In such a field with $m$ numbers, every nonzero element $a$ has a unique modular multiplicative inverse, $a^{-1}$ such that $aa^{-1} = a^{-1}a \equiv 1 \mod m$. This inverse can be found by solving the congruence equation $ax \equiv 1 \mod m$, or the equivalent linear Diophantine equation

$$ax + my = 1.$$ 

This equation can be solved by the Euclidean algorithm, as described above. Finding multiplicative inverses is an essential step in the RSA algorithm, which is widely used in electronic commerce; specifically, the equation determines the integer used to decrypt the message. Note that although the RSA algorithm uses rings rather than fields, the Euclidean algorithm can still be used to find a multiplicative inverse where one exists. The Euclidean algorithm also has other applications in error-correcting codes; for example, it can be used as an alternative to the Berlekamp–Massey algorithm for decoding BCH and Reed–Solomon codes, which are based on Galois fields.[7]

**Chinese remainder theorem**

Euclid's algorithm can also be used to solve multiple linear Diophantine equations. Such equations arise in the Chinese remainder theorem, which describes a novel method to represent an integer $x$. Instead of representing an integer by its digits, it may be represented by its remainders $x_i$ modulo a set of $N$ coprime numbers $m_i$.

$$x_1 \equiv x \mod m_1$$

$$x_2 \equiv x \mod m_2$$

... 

$$x_N \equiv x \mod m_N$$

The goal is to determine $x$ from its $N$ remainders $x_i$. The solution is to combine the multiple equations into a single linear Diophantine equation with a much larger modulus $M$ that is the product of all the individual moduli $m_i$, and define the $M_i$

$$M_i = M / m_i$$

Thus, each $M_i$ is the product of all the moduli except $m_i$. The solution depends on finding $N$ new numbers $h_i$ such that

$$M_i h_i \equiv 1 \mod m_i$$

With these numbers $h_i$, any integer $x$ can be reconstructed from its remainders $x_i$ by the equation

$$x \equiv (x_1 M_1 h_1 + x_2 M_2 h_2 + \ldots + x_N M_N h_N) \mod M$$

Since these numbers $h_i$ are the multiplicative inverses of the $M_i$, they may be found using Euclid's algorithm as described in the previous subsection.
**Stern–Brocot tree**

The sequence of subtractions used by the Euclidean algorithm gives a path from the root of the Stern–Brocot tree to any given rational number. This fact can be used to prove that there is a 1-1 correspondence between the vertices of tree and the positive rational numbers.

For example, \( \frac{3}{4} \) can be found by starting at the root, going to the left once, then to the right twice.

\[
\begin{align*}
gcd(3, 4) & \leftarrow \\
gcd(3, 1) & \rightarrow \\
gcd(2, 1) & \rightarrow \\
gcd(1, 1) & 
\end{align*}
\]

The Euclidean algorithm has almost the same relationship to the Calkin–Wilf tree. The difference is that the path is reversed: instead of producing a path from the root of the tree to a target, it produces a path from the target to the root.

**Continued fractions**

The Euclidean algorithm has a close relationship with continued fractions. The sequence of equations can be written in the form

\[
\begin{align*}
\frac{a}{b} & = q_0 + \frac{r_0}{b} \\
\frac{b}{r_0} & = q_1 + \frac{r_1}{r_0} \\
\frac{r_0}{r_1} & = q_2 + \frac{r_2}{r_1} \\
\vdots \\
\frac{r_{k-2}}{r_{k-1}} & = q_k + \frac{r_k}{r_{k-1}} \\
\vdots \\
\frac{r_{N-2}}{r_{N-1}} & = q_N
\end{align*}
\]

The last term on the right-hand side always equals the inverse of the left-hand side of the next equation. Thus, the first two equations may be combined to form

\[
\frac{a}{b} = q_0 + \frac{1}{q_1 + \frac{r_1}{r_0}}
\]

The third equation may be used to substitute the denominator term \( \frac{r_1}{r_0} \), yielding

\[
\frac{a}{b} = q_0 + \frac{1}{q_1 + \frac{1}{q_2 + \frac{r_2}{r_1}}}
\]
The final ratio of remainders \( r_k / r_{k-1} \) can always be replaced using the next equation in the series, up to the final equation. The result is a continued fraction

\[
\frac{a}{b} = q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \cfrac{1}{\ddots + \cfrac{1}{q_N}}}} = [q_0; q_1, q_2, \ldots, q_N]
\]

In the worked example above, the \( \gcd(1071, 462) \) was calculated, and the quotients \( q_k \) were 2, 3 and 7, respectively. Therefore, the fraction 1071/462 may be written

\[
\frac{1071}{462} = 2 + \cfrac{1}{3 + \cfrac{1}{7}} = [2; 3, 7]
\]
as can be confirmed by calculation.

**Factorization algorithms**

Calculating a greatest common divisor is an essential step in several integer factorization algorithms, such as Pollard's rho algorithm, Shor's algorithm, Dixon's factorization method and the Lenstra elliptic curve factorization. The Euclidean algorithm may be used to find this GCD efficiently. Continued fraction factorization uses continued fractions, which are determined using Euclid's algorithm.

**Algorithmic efficiency**

The computational efficiency of Euclid's algorithm has been studied thoroughly. This efficiency can be described by the number of division steps the algorithm requires, multiplied by the computational expense of each step. The first known analysis of Euclid's algorithm is due to A.-A.-L. Reynaud in 1811, who showed that the number of division steps on input \((u, v)\) is bounded by \(v\); later he improved this to \(v/2 + 2\). Later, in 1841, P.-J.-E. Finck showed that the number of division steps is at most \(2 \log_2 v + 1\), and hence Euclid's algorithm runs in time polynomial in the size of the input; also see. His analysis was refined by Gabriel Lamé in 1844, who showed that the number of steps required for completion is never more than five times the number \(h\) of base-10 digits of the smaller number \(b\). Since the computational expense of each step is also typically of order \(h\), the overall expense grows like \(h^2\).

**Number of steps**

The number of steps to calculate the GCD of two natural numbers, \(a\) and \(b\), may be denoted by \(T(a, b)\). If \(g\) is the GCD of \(a\) and \(b\), then \(a = mg\) and \(b = ng\) for two coprime numbers \(m\) and \(n\). Then

\[
T(a, b) = T(m, n)
\]
as may be seen by dividing all the steps in the Euclidean algorithm by \(g\). By the same argument, the number of steps remains the same if \(a\) and \(b\) are multiplied by a common factor \(w\): \(T(a, b) = T(aw, wb)\). Therefore, the number of steps \(T\) may vary dramatically between neighboring pairs of numbers, such as \(T(a, b)\) and \(T(a, b + 1)\), depending on
the size of the two GCDs.

The recursive nature of the Euclidean algorithm gives another equation

\[ T(a, b) = 1 + T(b, r_0) = 2 + T(r_0, r_1) = \ldots = N + T(r_{N-2}, r_{N-1}) = N + 1 \]

where \( T(x, 0) = 0 \) by assumption.

Worst-case number of steps

If the Euclidean algorithm requires \( N \) steps for a pair of natural numbers \( a > b > 0 \), the smallest values of \( a \) and \( b \) for which this is true are the Fibonacci numbers \( F_{N+2} \) and \( F_{N+1} \) respectively.\(^{[12]}\) This can be shown by induction. If \( N = 1 \), \( b \) divides \( a \) with no remainder; the smallest natural numbers for which this is true is \( b = 1 \) and \( a = 2 \), which are \( F_2 \) and \( F_3 \), respectively. Now assume that the result holds for all values of \( N \) up to \( M - 1 \). The first step of the \( M \)-step algorithm is

\[ a = q_0 b + r_0 \]

and the second step is

\[ b = q_1 r_0 + r_1 \]

Since the algorithm is recursive, it required \( M - 1 \) steps to find \( \gcd(b, r_0) \) and their smallest values are \( F_{M+1} \) and \( F_M \). The smallest value of \( a \) is therefore when \( q_0 = 1 \), which gives

\[ a = b + r_0 = F_{M+1} + F_M = F_{M+2} \]

This proof, published by Gabriel Lamé in 1844, represents the beginning of computational complexity theory, and also the first practical application of the Fibonacci numbers.

This result suffices to show that the number of steps in Euclid's algorithm can never be more than five times the number of its digits (base 10). For if the algorithm requires \( N \) steps, then \( b \) is greater than or equal to \( F_{N+1} \) which in turn is greater than or equal to \( \varphi^{N-1} \), where \( \varphi \) is the golden ratio. Since \( b \geq \varphi^{N-1} \), then \( N - 1 \leq \log_\varphi b \). Since \( \log_\varphi b > 1/5 \), \( (N - 1)/5 < \log_\varphi b \). Thus, \( N \leq 5 \log_{10} b \). Thus, the Euclidean algorithm always needs less than \( O(h) \) divisions, where \( h \) is the number of digits in the smaller number \( b \).

Average number of steps

The average number of steps taken by the Euclidean algorithm has been defined in three different ways. The first definition is the average time \( T(a) \) required to calculate the GCD of a given number \( a \) and a smaller natural number \( b \) chosen with equal probability from the integers 0 to \( a - 1 \)

\[ T(a) = \frac{1}{a} \sum_{0 \leq b \leq a} T(a, b). \]

However, since \( T(a, b) \) fluctuates dramatically with the GCD of the two numbers, the averaged function \( T(a) \) is likewise "noisy".\(^{[13]}\)

To reduce this noise, a second average \( \tau(a) \) is taken over all numbers coprime with \( a \)

\[ \tau(a) = \frac{1}{\varphi(a)} \sum_{0 \leq b < a, \gcd(a, b) = 1} T(a, b). \]

There are \( \varphi(a) \) coprime integers less than \( a \), where \( \varphi \) is Euler's totient function. This tau average grows smoothly with \( a \)\(^{[14]}\)

\[ \tau(a) = (12/\pi^2) \ln a + C + O(a^{-(1/6) + \varepsilon}) \]

with the residual error being of order \( a^{-(1/6) + \varepsilon} \), where \( \varepsilon \) is infinitesimal. The constant \( C \) (Porter's Constant)\(^{[15]}\) in this formula equals

\[ C = -(1/2) + 6(\ln 2/\pi^2)(4\gamma - 24\pi^2 \zeta'(2) + 3 \ln 2 - 2) \approx 1.467 \]

where \( \gamma \) is the Euler–Mascheroni constant and \( \zeta' \) is the derivative of the Riemann zeta function. The leading coefficient \( (12/\pi^2) \ln 2 \) was determined by two independent methods.

Since the first average can be calculated from the tau average by summing over the divisors \( d \) of \( a \)\(^{[16]}\)

\[ T(a) = \frac{1}{a} \sum_{d|a} \varphi(d) \tau(d) \]

it can be approximated by the formula
Euclidean algorithm

\[ T(a) \approx C + (12/\pi^2) \ln 2 \left( \ln a - \sum_{d|a} \Lambda(d)/d \right) \]

where \( \Lambda(d) \) is the Mangoldt function.\(^{[17]} \)

A third average \( Y(n) \) is defined as the mean number of steps required when both \( a \) and \( b \) are chosen randomly (with uniform distribution) from 1 to \( n \)

\[ Y(n) = \frac{1}{n^2} \sum_{a=1}^{n} \sum_{b=1}^{n} T(a, b) = \frac{1}{n} \sum_{a=1}^{n} T(a). \]

Substituting the approximate formula for \( T(a) \) into this equation yields an estimate for \( Y(n) \)\(^{[18]} \)

\[ Y(n) \approx (12/\pi^2) \ln 2 \ln n + 0.06. \]

**Computational expense per step**

In each step \( k \) of the Euclidean algorithm, the quotient \( q_k \) and remainder \( r_k \) are computed for a given pair of integers \( r_{k-2} \) and \( r_{k-1} \)

\[ r_{k-2} = q_k r_{k-1} + r'_k. \]

The computational expense per step is associated chiefly with finding \( q_k \); since the remainder \( r_k \) can be calculated quickly from \( r_{k-2}, r_{k-1} \), and \( q_k \)

\[ r_k = r_{k-2} - q_k r_{k-1}. \]

The computational expense of dividing \( h \)-bit numbers scales as \( O(h(\|q\|+1)) \), where \( \|q\| \) is the length of the quotient.\(^{[19]} \)

For comparison, Euclid's original subtraction-based algorithm can be much slower. A single integer division is equivalent to the quotient \( q \) number of subtractions. If the ratio of \( a \) and \( b \) is very large, the quotient is large and many subtractions will be required. On the other hand, it has been shown that the quotients are very likely to be small integers. The probability of a given quotient \( q \) is approximately \( \ln u/(u-1)! \) where \( u = (q+1)^2. \)\(^{[20]} \)

For illustration, the probability of a quotient of 1, 2, 3, or 4 is roughly 41.5%, 17.0%, 9.3%, and 5.9%, respectively. Since the operation of subtraction is faster than division, particularly for large numbers, the subtraction-based Euclid's algorithm is competitive with the division-based version. This is exploited in the binary version of Euclid's algorithm.

Combining the estimated number of steps with the estimated computational expense per step shows that the Euclid's algorithm grows quadratically \( (h^2) \) with the average number of digits \( h \) in the initial two numbers \( a \) and \( b \). Let \( h_0, h_1, \ldots, h_{N-1} \) represent the number of digits in the successive remainders \( r_0, r'_1, \ldots, r'_{N-1} \). Since the number of steps \( N \) grows linearly with \( h \), the running time is bounded by

\[ O\left( \sum_{i<N} h_i(h_i-h_{i+1}+2) \right) \subseteq O\left( h \sum_{i<N} (h_i-h_{i+1}+2) \right) \subseteq O(h(h_0+2N)) \subseteq O(h^2). \]

**Efficiency of alternative methods**

Euclid's algorithm is widely used in practice, especially for small numbers, due to its simplicity. For comparison, the efficiency of alternatives to Euclid's algorithm may be determined.

One inefficient approach to finding the GCD of two natural numbers \( a \) and \( b \) is to calculate all their common divisors; the GCD is then the largest common divisor. The common divisors can be found by dividing both numbers by successive integers from 2 to the smaller number \( b \). The number of steps of this approach grows linearly with \( b \), or exponentially in the number of digits. Another inefficient approach is to find the prime factors of one or both numbers. As noted above, the GCD equals the product of the prime factors shared by the two numbers \( a \) and \( b \).

Present methods for prime factorization are also inefficient; many modern cryptography systems even rely on that inefficiency.

The binary GCD algorithm is an efficient alternative that substitutes division with faster operations by exploiting the binary representation used by computers.\(^{[21]} \) However, this alternative also scales like \( O(h^2) \). It is generally faster
than the Euclidean algorithm on real computers, even though it scales in the same way. Additional efficiency can be
gleaned by examining only the leading digits of the two numbers \(a\) and \(b\). The binary algorithm can be extended
to other bases (\(k\)-ary algorithms), with up to fivefold increases in speed.

A recursive approach for very large integers (with more than 25,000 digits) leads to subquadratic integer GCD
algorithms, such as those of Schönhage, and Stehlé and Zimmermann. These algorithms exploit the 2x2 matrix
form of the Euclidean algorithm given above. These subquadratic methods generally scale as \(O(h (\log h)^2 (\log \log h))\).

**Other number systems**

As described above, the Euclidean algorithm is used to find the greatest common divisor of two natural numbers
(positive integers). However, it may be generalized to the real numbers, and to more exotic number systems such as
polynomials, quadratic integers and Hurwitz quaternions. In the latter cases, the Euclidean algorithm is used to
demonstrate the crucial property of unique factorization, i.e., that such numbers can be factored uniquely into
irreducible elements, the counterparts of prime numbers. Unique factorization is essential to many proofs of number
theory.

**Rational and real numbers**

Euclid’s algorithm can be applied to real numbers, as described by Euclid in Book 10 of his *Elements*. The goal of
the algorithm is to identify a real number \(g\) such that two given real numbers, \(a\) and \(b\), are integer multiples of it: \(a = mg\)
and \(b = ng\), where \(m\) and \(n\) are integers. This identification is equivalent to finding an integer relation among the
real numbers \(a\) and \(b\); that is, it determines integers \(s\) and \(t\) such that \(sa + ib = 0\). Euclid uses this algorithm to treat
the question of incommensurable lengths.

The real-number Euclidean algorithm differs from its integer counterpart in two respects. First, the remainders \(r_k\)
are real numbers, although the quotients \(q_k\) are integers as before. Second, the algorithm is not guaranteed to end in a
finite number \(N\) of steps. If it does, the fraction \(a/b\) is a rational number, i.e., the ratio of two integers
\[
\frac{a}{b} = \frac{mg}{ng} = \frac{m}{n}
\]
and can be written as a finite continued fraction \([q_0; q_1, q_2, \ldots, q_N]\). If the algorithm does not stop, the fraction \(a/b\)
is an irrational number and can be described by an infinite continued fraction \([q_0; q_1, q_2, \ldots]\). Examples of infinite
continued fractions are the golden ratio \(\varphi = [1; 1, 1, \ldots]\) and the square root of two, \(\sqrt{2} = [1; 2, 2, \ldots]\). Generally
speaking, the algorithm is unlikely to stop, since almost all ratios \(a/b\) of two real numbers are irrational.

An infinite continued fraction may be truncated at a step \(k\) \([q_0; q_1, q_2, \ldots, q_k]\) to yield an approximation to \(a/b\) that
improves as \(k\) is increased. The approximation is described by convergents \(m_k/n_k\): the numerator and denominators
are coprime and obey the recursion
\[
\begin{align*}
m_k &= q_k m_{k-1} + m_{k-2} \\
n_k &= q_k n_{k-1} + n_{k-2}
\end{align*}
\]
where \(m_{-1} = n_{-1} = 1\) and \(m_{-2} = n_{-2} = 0\) are the initial values of the recursion. The convergent \(m_k/n_k\) is the best
rational number approximation to \(a/b\) with denominator \(n_k\):
\[
\left| \frac{a}{b} - \frac{m_k}{n_k} \right| < \frac{1}{n_k^2}.
\]
Euclidean algorithm

Polynomials

Polynomials in a single variable $x$ can be added, multiplied and factored into irreducible polynomials, which are the analogs of the prime numbers for integers. The greatest common divisor polynomial $g(x)$ of two polynomials $a(x)$ and $b(x)$ is defined as the product of their shared irreducible polynomials, which can be identified using the Euclidean algorithm. The basic procedure is similar to integers. At each step $k$, a quotient polynomial $q_k(x)$ and a remainder polynomial $r_k(x)$ are identified to satisfy the recursive equation

$$r_{k-2}(x) = q_k(x) r_{k-1}(x) + r_k(x)$$

where $r_{-2}(x) = a(x)$ and $r_{-1}(x) = b(x)$. The quotient polynomial is chosen so that the leading term of $q_k(x) r_{k-1}(x)$ equals the leading term of $r_{k-2}(x)$; this ensures that the degree of each remainder is smaller than the degree of its predecessor $\deg[r_k(x)] < \deg[r_{k-1}(x)]$. Since the degree is a nonnegative integer, and since it decreases with every step, the Euclidean algorithm concludes in a finite number of steps. The final nonzero remainder is the greatest common divisor of the original two polynomials, $a(x)$ and $b(x)$.

For example, consider the following two quartic polynomials, which each factor into two quadratic polynomials

$$a(x) = x^4 - 4x^3 + 4x^2 - 3x + 14 = (x^2 - 5x + 7)(x^2 + x + 2)$$

and

$$b(x) = x^4 + 8x^3 + 12x^2 + 17x + 6 = (x^2 + 7x + 3)(x^2 + x + 2).$$

Dividing $a(x)$ by $b(x)$ yields a remainder $r_0(x) = x^3 + (2/3) x^2 + (5/3) x - (2/3)$. In the next step, $b(x)$ is divided by $r_0(x)$ yielding a remainder $r_1(x) = x^2 + x + 2$. Finally, dividing $r_0(x)$ by $r_1(x)$ yields a zero remainder, indicating that $r_1(x)$ is the greatest common divisor polynomial of $a(x)$ and $b(x)$, consistent with their factorization.

Many of the applications described above for integers carry over to polynomials. The Euclidean algorithm can be used to solve linear Diophantine equations and Chinese remainder problems for polynomials; continued fractions of polynomials can also be defined.

The polynomial Euclidean algorithm has other applications as well, such as Sturm chains, a method for counting the number of real roots of a polynomial within a given interval on the real axis. This has applications in several areas, such as the Routh–Hurwitz stability criterion in control theory.

Finally, the coefficients of the polynomials need not be drawn from integers, real numbers or even the complex numbers. For example, the coefficients may be drawn from a general field, such as the finite fields GF($p$) described above. The corresponding conclusions about the Euclidean algorithm and its applications hold even for such polynomials.
Gaussian integers

The Gaussian integers are complex numbers of the form $\alpha = u + vi$, where $u$ and $v$ are ordinary integers and $i$ is the square root of negative one. By defining an analog of the Euclidean algorithm, Gaussian integers can be shown to be uniquely factorizable, by the argument above. This unique factorization is helpful in many applications, such as deriving all Pythagorean triples or proving Fermat’s theorem on sums of two squares. In general, the Euclidean algorithm is convenient in such applications, but not essential; for example, the theorems can often be proven by other arguments.

The Euclidean algorithm developed for two Gaussian integers $\alpha$ and $\beta$ is nearly the same as that for normal integers, but differs in two respects. As before, the task at each step $k$ is to identify a quotient $q_k$ and a remainder $r_k$ such that

$$r_k = r_{k-2} - q_k r_{k-1}$$

where $r_{k-2} = \alpha$, $r_{k-1} = \beta$, and every remainder is strictly smaller than its predecessor, $|r_k| < |r_{k-1}|$. The first difference is that the quotients and remainders are themselves Gaussian integers, and thus are complex numbers. The quotients $q_k$ are generally found by rounding the real and complex parts of the exact ratio (such as the complex number $\alpha/\beta$) to the nearest integers. The second difference lies in the necessity of defining how one complex remainder can be "smaller" than another. To do this, we define a norm function $f(u + vi) = u^2 + v^2$, which converts every Gaussian integer $u + vi$ into a normal integer. After each step $k$ of the Euclidean algorithm, the norm of the remainder $f(r_k)$ is smaller than the norm of the preceding remainder, $f(r_{k-1})$. Since the norm is a nonnegative integer and decreases with every step, the Euclidean algorithm for Gaussian integers ends in a finite number of steps. The final nonzero remainder is the $\gcd(\alpha,\beta)$, the Gaussian integer of largest norm that divides both $\alpha$ and $\beta$; it is unique up to multiplication by a unit, $\pm 1$ or $\pm i$.

Many of the other applications of the Euclidean algorithm carry over to Gaussian integers. For example, it can be used to solve linear Diophantine equations and Chinese remainder problems for Gaussian integers; continued fractions of Gaussian integers can also be defined.

Euclidean domains

A set of elements under two binary operations, $+$ and $\cdot$, is called a Euclidean domain if it forms a commutative ring $R$ and, roughly speaking, if a generalized Euclidean algorithm can be performed on them. The two operations of such a ring need not be the addition and multiplication of ordinary arithmetic; rather, they can be more general, such as the operations of a mathematical group or monoid. Nevertheless, these general operations should respect many of the laws governing ordinary arithmetic, such as commutativity, associativity and distributivity.

The generalized Euclidean algorithm requires a Euclidean function, i.e., a mapping $f$ from $R$ into the set of nonnegative integers such that, for any two nonzero elements $a$ and $b$ in $R$, there exist $q$ and $r$ in $R$ such that $a = qb + r$ and $f(r) < f(b)$. An example of this mapping is the norm function used to order the Gaussian integers above. The function $f$ can be the magnitude of the number, or the degree of a polynomial. The basic principle is that each step of the algorithm reduces $f$ inexorably; hence, if $f$ can be reduced only a finite number of times, the algorithm must stop in a finite number of steps. This principle relies heavily on the natural well-ordering of the non-negative integers; roughly speaking, this requires that every non-empty set of non-negative integers has a smallest member.

The fundamental theorem of arithmetic applies to any Euclidean domain: Any number from a Euclidean domain can be factored uniquely into irreducible elements. Any Euclidean domain is a unique factorization domain (UFD), although the converse is not true. The Euclidean domains and the UFD's are subclasses of the GCD domains,
domains in which a greatest common divisor of two numbers always exists. In other words, a greatest common divisor may exist (for all pairs of elements in a domain), although it may not be possible to find it using a Euclidean algorithm. A Euclidean domain is always a principal ideal domain (PID), an integral domain in which every ideal is a principal ideal. Again, the converse is not true: not every PID is a Euclidean domain.

The unique factorization of Euclidean domains is useful in many applications. For example, the unique factorization of the Gaussian integers is convenient in deriving formulae for all Pythagorean triples and in proving Fermat's theorem on sums of two squares. Unique factorization was also a key element in an attempted proof of Fermat's Last Theorem published in 1847 by Gabriel Lamé, the same mathematician who analyzed the efficiency of Euclid's algorithm, based on a suggestion of Joseph Liouville. Lamé's approach required the unique factorization of numbers of the form \( x + \omega y \), where \( x \) and \( y \) are integers, and \( \omega = e^{2\pi i/n} \) is an \( n \)th root of 1, that is, \( \omega^n = 1 \). Although this approach succeeds for some values of \( n \) (such as \( n=3 \), the Eisenstein integers), in general such numbers do not factor uniquely. This failure of unique factorization in some cyclotomic fields led Ernst Kummer to the concept of ideal numbers and, later, Richard Dedekind to ideals. [citation needed]

Unique factorization of quadratic integers

The quadratic integer rings are helpful to illustrate Euclidean domains. Quadratic integers are generalizations of the Gaussian integers in which the imaginary unit \( i \) is replaced by a number \( \omega \). Thus, they have the form \( u + v \omega \), where \( u \) and \( v \) are integers and \( \omega \) has one of two forms, depending on a parameter \( D \). If \( D \) does not equal a multiple of four plus one, then
\[
\omega = \sqrt{D}.
\]
If, however, \( D \) does equal a multiple of four plus one, then
\[
\omega = (1 + \sqrt{D})/2.
\]
If the function \( f \) corresponds to a norm function, such as that used to order the Gaussian integers above, then the domain is known as norm-Euclidean. The norm-Euclidean rings of quadratic integers are exactly those where \( D = -11, -7, -3, -2, -1, 2, 3, 5, 6, 7, 11, 13, 17, 19, 21, 29, 33, 37, 41, 57 \) or 73. The quadratic integers with \( D = -1 \) and \( -3 \) are known as the Gaussian integers and Eisenstein integers, respectively.

If \( f \) is allowed to be any Euclidean function, then the list of possible \( D \) values for which the domain is Euclidean is not yet known. The first example of a Euclidean domain that was not norm-Euclidean (with \( D = 69 \) was published in 1994. In 1973, Weinberger proved that a quadratic integer ring with \( D > 0 \) is Euclidean if, and only if, it is a principal ideal domain, provided that the generalized Riemann hypothesis holds.

Noncommutative rings

It is also possible to apply the Euclidean algorithm to noncommutative rings such as the set of Hurwitz quaternions. Let \( \alpha \) and \( \beta \) represent two elements from such a ring. They have a common right divisor \( \delta \) if \( \alpha = \xi \delta \) and \( \beta = \eta \delta \) for some choice of \( \xi \) and \( \eta \) in the ring. Similarly, they have a common left divisor if \( \alpha = \delta \xi \) and \( \beta = \delta \eta \) for some choice of \( \xi \) and \( \eta \) in the ring. Since multiplication is not commutative, there are two versions of the Euclidean algorithm, one for right divisors and one for left divisors. Choosing the right divisors, the first step in finding the \( \gcd(\alpha, \beta) \) by the Euclidean algorithm can be written
\[
\rho_0 = \alpha - \psi_0 \beta = (\xi - \psi_0 \eta)\delta
\]
where \( \psi_0 \) represents the quotient and \( \rho_0 \) the remainder. This equation shows that any common right divisor of \( \alpha \) and \( \beta \) is likewise a common divisor of the remainder \( \rho_0 \). The analogous equation for the left divisors would be
\[ \rho_0 = \alpha - \beta \psi_0 = \delta(\xi - \eta \psi_0) \]

With either choice, the process is repeated as above until the greatest common right or left divisor is identified. As in the Euclidean domain, the "size" of the remainder \( \rho_0 \) must be strictly smaller than \( \beta \), and there must be only a finite number of possible sizes for \( \rho_0 \), so that the algorithm is guaranteed to terminate.

Most of the results for the GCD carry over to noncommutative numbers. For example, Bézout's identity states that the right \( \gcd(\alpha, \beta) \) can be expressed as a linear combination of \( \alpha \) and \( \beta \). In other words, there are numbers \( \sigma \) and \( \tau \) such that

\[ \Gamma_{\text{right}} = \sigma \alpha + \tau \beta \]

The analogous identity for the left GCD is nearly the same

\[ \Gamma_{\text{left}} = \alpha \sigma + \beta \tau \]

Bézout's identity can be used to solve Diophantine equations.

**Generalizations to other mathematical structures**

The Euclidean algorithm has three general features that together ensure it will not continue indefinitely. First, it can be written as a sequence of recursive equations

\[ r_k = r_{k-2} - q_k r_{k-1} \]

where each remainder is strictly smaller than its predecessor, \( |r_k| < |r_{k-1}| \). Second, the size of each remainder has a strict lower limit, such as \( |r_k| \geq 0 \). Third, there is only a finite number of sizes smaller than a given remainder \( |r_k| \). Generalizations of Euclid's algorithm with these basic features have been applied to other mathematical structures, such as tangles and transfinite ordinal numbers.

An important generalization of the Euclidean algorithm is the concept of a Gröbner basis in algebraic geometry. As shown above, the GCD \( g \) of two integers \( a \) and \( b \) is the generator of their ideal. In other words, for any choice of the integers \( s \) and \( t \), there is another integer \( m \) such that

\[ sa + tb = mg. \]

Although this remains true when \( s, t, m, a \) and \( b \) represent polynomials of a single variable, it is not true for rings of more than one variable. In that case, a finite set of generator polynomials \( g_1, g_2, \text{ etc.} \) can be defined such that any linear combination of two multivariable polynomials \( a \) and \( b \) can be expressed as multiples of the generators

\[ sa + tb = \sum_k m_k g_k \]

where \( s, t \) and \( m_k \) are multivariable polynomials. Any such multivariable polynomial \( f \) can be expressed as such a sum of generator polynomials plus a unique remainder polynomial \( r \), sometimes called the normal form of polynomial \( f \)

\[ f = r + \sum_k q_k g_k \]

although the quotient polynomials \( q_k \) may not be unique. The set of these generator polynomials is known as a Gröbner basis.
Notes

- a. Some widely used textbooks, such as I. N. Herstein's *Topics in Algebra* and Serge Lang's *Algebra*, use the term "Euclidean algorithm" to refer to Euclidean division.

References


[23] Volume 1423 in *Lecture notes in Computer Science*.

Bibliography


Euclidean algorithm


External links
• Demonstrations of Euclid's algorithm (http://www.math.sc.edu/~sumner/numbertheory/euclidean/euclidean.html)
• Euclid's Algorithm (http://www.cut-the-knot.org/blue/Euclid.shtml) at cut-the-knot
• Euclid's algorithm (http://planetmath.org/encyclopedia/EuclidsAlgorithm.html) at PlanetMath
• The Euclidean Algorithm (http://www.mathpages.com/home/kmath384.htm) at MathPages
• Euclid's Game (http://www.cut-the-knot.org/blue/EuclidAlg.shtml) at cut-the-knot
• Music and Euclid's algorithm (http://plus.maths.org/issue40/features/wardhaugh/index.html)
**Prime number**

A prime number (or a prime) is a natural number greater than 1 that has no positive divisors other than 1 and itself. A natural number greater than 1 that is not a prime number is called a composite number. For example, 5 is prime because only 1 and 5 evenly divide it, whereas 6 is composite because it has the divisors 2 and 3 in addition to 1 and 6. The fundamental theorem of arithmetic establishes the central role of primes in number theory: any integer greater than 1 can be expressed as a product of primes that is unique up to ordering. The uniqueness in this theorem requires excluding 1 as a prime because one can include arbitrarily-many instances of 1 in any factorization, e.g., 3, 1 × 3, 1 × 1 × 3, etc. are all valid factorizations of 3.

The property of being prime (or not) is called primality. A simple but slow method of verifying the primality of a given number \( n \) is known as trial division. It consists of testing whether \( n \) is a multiple of any integer between 2 and \( \sqrt{n} \). Algorithms much more efficient than trial division have been devised to test the primality of large numbers. Particularly fast methods are available for numbers of special forms, such as Mersenne numbers. As of February 2013[1], the largest known prime number has 17,425,170 decimal digits.

There are infinitely many primes, as demonstrated by Euclid around 300 BC. There is no known useful formula that sets apart all of the prime numbers from composites. However, the distribution of primes, that is to say, the statistical behaviour of primes in the large, can be modelled. The first result in that direction is the prime number theorem, proven at the end of the 19th century, which says that the probability that a given, randomly chosen number \( n \) is prime is inversely proportional to its number of digits, or to the logarithm of \( n \).

Many questions around prime numbers remain open, such as Goldbach’s conjecture (that every even integer greater than 2 can be expressed as the sum of two primes), and the twin prime conjecture (that there are infinitely many pairs of primes whose difference is 2). Such questions spurred the development of various branches of number theory, focusing on analytic or algebraic aspects of numbers. Primes are used in several routines in information technology, such as public-key cryptography, which makes use of properties such as the difficulty of factoring large numbers into their prime factors. Prime numbers give rise to various generalizations in other mathematical domains, mainly algebra, such as prime elements and prime ideals.
Definition and examples
A natural number (i.e. 1, 2, 3, 4, 5, 6, etc.) is called a prime or a prime number if it has exactly two positive divisors, 1 and the number itself. Natural numbers greater than 1 that are not prime are called composite.

Among the numbers 1 to 6, the numbers 2, 3, and 5 are the prime numbers, while 1, 4, and 6 are not prime. 1 is excluded as a prime number, for reasons explained below. 2 is a prime number, since the only natural numbers dividing it are 1 and 2. Next, 3 is prime, too: 1 and 3 do divide 3 without remainder, but 3 divided by 2 gives remainder 1. Thus, 3 is prime. However, 4 is composite, since 2 is another number (in addition to 1 and 4) dividing 4 without remainder:

\[4 = 2 \cdot 2.\]

5 is again prime: none of the numbers 2, 3, or 4 divide 5. Next, 6 is divisible by 2 or 3, since

\[6 = 2 \cdot 3.\]

Hence, 6 is not prime. The image at the right illustrates that 12 is not prime: 12 = 3 · 4. No even number greater than 2 is prime because by definition, any such number \(n\) has at least three distinct divisors, namely 1, 2, and \(n\). This implies that \(n\) is not prime. Accordingly, the term odd prime refers to any prime number greater than 2. In a similar vein, all prime numbers bigger than 5, written in the usual decimal system, end in 1, 3, 7, or 9, since even numbers are multiples of 2 and numbers ending in 0 or 5 are multiples of 5.

If \(n\) is a natural number, then 1 and \(n\) divide \(n\) without remainder. Therefore, the condition of being a prime can also be restated as: a number is prime if it is greater than one and if none of

\[2, 3, ..., n-1\]

divides \(n\) (without remainder). Yet another way to say the same is: a number \(n > 1\) is prime if it cannot be written as a product of two integers \(a\) and \(b\), both of which are larger than 1:

\[n = a \cdot b.\]

In other words, \(n\) is prime if \(n\) items cannot be divided up into smaller equal-size groups of more than one item.

The smallest 168 prime numbers (all the prime numbers under 1000) are:


The set of all primes is often denoted \(P\).
**Fundamental theorem of arithmetic**

The crucial importance of prime numbers to number theory and mathematics in general stems from the *fundamental theorem of arithmetic*, which states that every integer larger than 1 can be written as a product of one or more primes in a way that is unique except for the order of the prime factors. Primes can thus be considered the "basic building blocks" of the natural numbers. For example:

\[
23244 = 2 \cdot 2 \cdot 3 \cdot 13 \cdot 149
\]

\[
= 2^2 \cdot 3 \cdot 13 \cdot 149. \quad (2^2 \text{ denotes the square or second power of } 2.)
\]

As in this example, the same prime factor may occur multiple times. A decomposition:

\[n = p_1 \cdot p_2 \cdots p_t\]

of a number \(n\) into (finitely many) prime factors \(p_1, p_2, \ldots, p_t\) is called *prime factorization* of \(n\). The fundamental theorem of arithmetic can be rephrased so as to say that any factorization into primes will be identical except for the order of the factors. So, albeit there are many prime factorization algorithms to do this in practice for larger numbers, they all have to yield the same result.

If \(p\) is a prime number and \(p\) divides a product \(ab\) of integers, then \(p\) divides \(a\) or \(p\) divides \(b\). This proposition is known as Euclid's lemma. It is used in some proofs of the uniqueness of prime factorizations.

**Primality of one**

Most early Greeks did not even consider 1 to be a number,[3] and so they did not consider it a prime. In the 19th century however, many mathematicians did consider the number 1 a prime. For example, Derrick Norman Lehmer's list of primes up to 10,006,721, reprinted as late as 1956, started with 1 as its first prime. Henri Lebesgue is said to be the last professional mathematician to call 1 prime. Although a large body of mathematical work is also valid when calling 1 a prime, the above fundamental theorem of arithmetic does not hold as stated. For example, the number 15 can be factored as \(3 \cdot 5\) or \(1 \cdot 3 \cdot 5\). If 1 were admitted as a prime, these two presentations would be considered different factorizations of 15 into prime numbers, so the statement of that theorem would have to be modified. Furthermore, the prime numbers have several properties that the number 1 lacks, such as the relationship of the number to its corresponding value of Euler's totient function or the sum of divisors function.[4][5]
History

There are hints in the surviving records of the ancient Egyptians that they had some knowledge of prime numbers: the Egyptian fraction expansions in the Rhind papyrus, for instance, have quite different forms for primes and for composites. However, the earliest surviving records of the explicit study of prime numbers come from the Ancient Greeks. Euclid's Elements (circa 300 BC) contain important theorems about primes, including the infinitude of primes and the fundamental theorem of arithmetic. Euclid also showed how to construct a perfect number from a Mersenne prime. The Sieve of Eratosthenes, attributed to Eratosthenes, is a simple method to compute primes, although the large primes found today with computers are not generated this way.

After the Greeks, little happened with the study of prime numbers until the 17th century. In 1640 Pierre de Fermat stated (without proof) Fermat's little theorem (later proved by Leibniz and Euler). Fermat conjectured that all numbers of the form $2^n + 1$ are prime (they are called Fermat numbers) and he verified this up to $n = 4$ (or $2^{16} + 1$). However, the very next Fermat number $2^{32} + 1$ is composite (one of its prime factors is 641), as Euler discovered later, and in fact no further Fermat numbers are known to be prime. The French monk Marin Mersenne looked at primes of the form $2^p - 1$, with $p$ a prime. They are called Mersenne primes in his honor.

Euler's work in number theory included many results about primes. He showed the infinite series $1/2 + 1/3 + 1/5 + 1/7 + 1/11 + \ldots$ is divergent. In 1747 he showed that the even perfect numbers are precisely the integers of the form $2^{p-1}(2^p - 1)$, where the second factor is a Mersenne prime.

At the start of the 19th century, Legendre and Gauss independently conjectured that as $x$ tends to infinity, the number of primes up to $x$ is asymptotic to $x/\ln(x)$, where $\ln(x)$ is the natural logarithm of $x$. Ideas of Riemann in his 1859 paper on the zeta-function sketched a program that would lead to a proof of the prime number theorem. This outline was completed by Hadamard and de la Vallée Poussin, who independently proved the prime number theorem in 1896.

Proving a number is prime is not done (for large numbers) by trial division. Many mathematicians have worked on primality tests for large numbers, often restricted to specific number forms. This includes Pépin's test for Fermat numbers (1877), Proth's theorem (around 1878), the Lucas–Lehmer primality test (originated 1856),[6] and the generalized Lucas primality test. More recent algorithms like APRT-CL, ECPP, and AKS work on arbitrary numbers but remain much slower.

For a long time, prime numbers were thought to have extremely limited application outside of pure mathematics.[7] This changed in the 1970s when the concepts of public-key cryptography were invented, in which prime numbers formed the basis of the first algorithms such as the RSA cryptosystem algorithm.
Since 1951 all the largest known primes have been found by computers. The search for ever larger primes has
generated interest outside mathematical circles. The Great Internet Mersenne Prime Search and other distributed
computing projects to find large primes have become popular in the last ten to fifteen years, while mathematicians
continue to struggle with the theory of primes.

**Number of prime numbers**

There are infinitely many prime numbers. Another way of saying this is that the sequence

2, 3, 5, 7, 11, 13, ...

of prime numbers never ends. This statement is referred to as Euclid's theorem in honor of the ancient Greek
mathematician Euclid, since the first known proof for this statement is attributed to him. Many more proofs of the
infinitude of primes are known, including an analytical proof by Euler, Goldbach's proof based on Fermat
numbers,[8] Furstenberg's proof using general topology, and Kummer's elegant proof.

**Euclid's proof**

Euclid's proof (Book IX, Proposition 20[9]) considers any finite set \( S \) of primes. The key idea is to consider the
product of all these numbers plus one:

\[
N = 1 + \prod_{p \in S} p.
\]

Like any other natural number, \( N \) is divisible by at least one prime number (it is possible that \( N \) itself is prime).

None of the primes by which \( N \) is divisible can be members of the finite set \( S \) of primes with which we started,
because dividing \( N \) by any one of these leaves a remainder of 1. Therefore the primes by which \( N \) is divisible are
additional primes beyond the ones we started with. Thus any finite set of primes can be extended to a larger finite set
of primes.

It is often erroneously reported that Euclid begins with the assumption that the set initially considered contains all
prime numbers, leading to a contradiction, or that it contains precisely the \( n \) smallest primes rather than any arbitrary
finite set of primes. Today, the product of the smallest \( n \) primes plus 1 is conventionally called the \( n \)th Euclid
number.

**Euler's analytical proof**

Euler's proof uses the sum of the reciprocals of primes,

\[
S(p) = \frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \cdots + \frac{1}{p}.
\]

This sum becomes bigger than any arbitrary real number provided that \( p \) is big enough.[10] This shows that there are
infinitely many primes, since otherwise this sum would grow only until the biggest prime \( p \) is reached. The growth
of \( S(p) \) is quantified by Mertens' second theorem. For comparison, the sum

\[
\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{n^2} = \sum_{i=1}^{n} \frac{1}{i^2}
\]

does not grow to infinity as \( n \) goes to infinity. In this sense, prime numbers occur more often than squares of natural
numbers. Brun's theorem states that the sum of the reciprocals of twin primes,

\[
\left( \frac{1}{3} + \frac{1}{5} \right) + \left( \frac{1}{5} + \frac{1}{7} \right) + \left( \frac{1}{11} + \frac{1}{13} \right) + \cdots = \sum_{\substack{p \text{ prime,} \\ p+2 \text{ prime}}} \left( \frac{1}{p} + \frac{1}{p+2} \right),
\]

is finite.
Testing primality and integer factorization

There are various methods to determine whether a given number \( n \) is prime. The most basic routine, trial division, is of little practical use because of its slowness. One group of modern primality tests is applicable to arbitrary numbers, while more efficient tests are available for particular numbers. Most such methods only tell whether \( n \) is prime or not. Routines also yielding one (or all) prime factors of \( n \) are called factorization algorithms.

**Trial division**

The most basic method of checking the primality of a given integer \( n \) is called *trial division*. This routine consists of dividing \( n \) by each integer \( m \) that is greater than 1 and less than or equal to the square root of \( n \). If the result of any of these divisions is an integer, then \( n \) is not a prime, otherwise it is a prime. Indeed, if \( n = ab \) is composite (with \( a \) and \( b \neq 1 \)) then one of the factors \( a \) or \( b \) is necessarily at most \( \sqrt{n} \). For example, for \( n = 37 \), the trial divisions are by \( m = 2, 3, 4, 5, \) and 6. None of these numbers divides 37, so 37 is prime. This routine can be implemented more efficiently if a complete list of primes up to \( \sqrt{n} \) is known—then trial divisions need to be checked only for those \( m \) that are prime. For example, to check the primality of 37, only three divisions are necessary \((m = 2, 3, \) and 5\), given that 4 and 6 are composite.

While a simple method, trial division quickly becomes impractical for testing large integers because the number of possible factors grows too rapidly as \( n \) increases. According to the prime number theorem explained below, the number of prime numbers less than \( \sqrt{n} \) is approximately given by \( \sqrt{n} / \ln(\sqrt{n}) \), so the algorithm may need up to this number of trial divisions to check the primality of \( n \). For \( n = 10^{20} \), this number is 450 million—too large for many practical applications.

**Sieves**

An algorithm yielding all primes up to a given limit, such as required in the trial division method, is called a prime number sieve. The oldest example, the sieve of Eratosthenes (see above) is useful for relatively small primes. The modern sieve of Atkin is more complicated, but faster when properly optimized. Before the advent of computers, lists of primes up to bounds like \( 10^7 \) were also used.

**Primality testing versus primality proving**

Modern primality tests for general numbers \( n \) can be divided into two main classes, probabilistic (or "Monte Carlo") and deterministic algorithms. Deterministic algorithms provide a way to tell for sure whether a given number is prime or not. For example, trial division is a deterministic algorithm because, if it performed correctly, it will always identify a prime number as prime and a composite number as composite. Probabilistic algorithms are normally faster, but do not completely prove that a number is prime. These tests rely on testing a given number in a partly random way. For example, a given test might pass all the time if applied to a prime number, but pass only with probability \( p \) if applied to a composite number. If we repeat the test \( n \) times and pass every time, then the probability that our number is composite is \( 1/(1-p)^n \), which decreases exponentially with the number of tests, so we can be as sure as we like (though never perfectly sure) that the number is prime. On the other hand, if the test ever fails, then we know that the number is composite.

A particularly simple example of a probabilistic test is the Fermat primality test, which relies on the fact (Fermat's little theorem) that \( n^\phi(n) \equiv 1 \pmod{p} \) for any \( n \) if \( p \) is a prime number. If we have a number \( b \) that we want to test for primality, then we work out \( n^b \pmod{b} \) for a random value of \( n \) as our test. A flaw with this test is that there are some composite numbers (the Carmichael numbers) that satisfy the Fermat identity even though they are not prime, so the test has no way of distinguishing between prime numbers and Carmichael numbers. Carmichael numbers are substantially rarer than prime numbers, though, so this test can be useful for practical purposes. More powerful extensions of the Fermat primality test, such as the Baillie-PSW, Miller-Rabin, and Solovay-Strassen tests, are guaranteed to fail at least some of the time when applied to a composite number.
Deterministic algorithms do not erroneously report composite numbers as prime. In practice, the fastest such method is known as elliptic curve primality proving. Analyzing its run time is based on heuristic arguments, as opposed to the rigorously proven complexity of the more recent AKS primality test. Deterministic methods are typically slower than probabilistic ones, so the latter ones are typically applied first before a more time-consuming deterministic routine is employed.

The following table lists a number of prime tests. The running time is given in terms of \( n \), the number to be tested and, for probabilistic algorithms, the number \( k \) of tests performed. Moreover, \( \varepsilon \) is an arbitrarily small positive number, and \( \log \) is the logarithm to an unspecified base. The big O notation means that, for example, elliptic curve primality proving requires a time that is bounded by a factor (not depending on \( n \), but on \( \varepsilon \)) times \( \log^{5+\varepsilon}(n) \).

<table>
<thead>
<tr>
<th>Test</th>
<th>Developed in</th>
<th>Type</th>
<th>Running time</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKS primality test</td>
<td>2002</td>
<td>deterministic</td>
<td>( O(\log^{6+\varepsilon}(n)) )</td>
<td></td>
</tr>
<tr>
<td>Elliptic curve primality proving</td>
<td>1977</td>
<td>deterministic</td>
<td>( O(\log^{5+\varepsilon}(n)) ) heuristically</td>
<td></td>
</tr>
<tr>
<td>Baillie-PSW primality test</td>
<td>1980</td>
<td>probabilistic</td>
<td>( O(\log^3 n) )</td>
<td>no known counterexamples</td>
</tr>
<tr>
<td>Miller–Rabin primality test</td>
<td>1980</td>
<td>probabilistic</td>
<td>( O(k \cdot \log^{2+\varepsilon} (n)) )</td>
<td>error probability ( 4^{-k} )</td>
</tr>
<tr>
<td>Solovay–Strassen primality test</td>
<td>1977</td>
<td>probabilistic</td>
<td>( O(k \cdot \log^3 n) )</td>
<td>error probability ( 2^{-k} )</td>
</tr>
<tr>
<td>Fermat primality test</td>
<td></td>
<td>probabilistic</td>
<td>( O(k \cdot \log^{2+\varepsilon} (n)) )</td>
<td>fails for Carmichael numbers</td>
</tr>
</tbody>
</table>

**Special-purpose algorithms and the largest known prime**

In addition to the aforementioned tests applying to any natural number \( n \), a number of much more efficient primality tests is available for special numbers. For example, to run Lucas’ primality test requires the knowledge of the prime factors of \( n − 1 \), while the Lucas–Lehmer primality test needs the prime factors of \( n + 1 \) as input. For example, these tests can be applied to check whether

\[
n! \pm 1 = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \pm 1
\]

are prime. Prime numbers of this form are known as factorial primes. Other primes where either \( p + 1 \) or \( p − 1 \) is of a particular shape include the Sophie Germain primes (primes of the form \( 2p + 1 \) with \( p \) prime), primorial primes, Fermat primes and Mersenne primes, that is, prime numbers that are of the form \( 2^p − 1 \), where \( p \) is an arbitrary prime. The Lucas–Lehmer test is particularly fast for numbers of this form. This is why the largest known prime has almost always been a Mersenne prime since the dawn of electronic computers.

Fermat primes are of the form

\[
F_k = 2^{2^k} + 1,
\]

with \( k \) an arbitrary natural number. They are named after Pierre de Fermat who conjectured that all such numbers \( F_k \) are prime. This was based on the evidence of the first five numbers in this series—3, 5, 17, 257, and 65,537—being prime. However, \( F_5 \) is composite and so are all other Fermat numbers that have been verified as of 2011. A regular \( n \)-gon is constructible using straightedge and compass if and only if

\[
n = 2^i \cdot m
\]

where \( m \) is a product of any number of distinct Fermat primes and \( i \) is any natural number, including zero.
The following table gives the largest known primes of the mentioned types. Some of these primes have been found using distributed computing. In 2009, the Great Internet Mersenne Prime Search project was awarded a US$100,000 prize for first discovering a prime with at least 10 million digits. The Electronic Frontier Foundation also offers $150,000 and $250,000 for primes with at least 100 million digits and 1 billion digits, respectively. Some of the largest primes not known to have any particular form (that is, no simple formula such as that of Mersenne primes) have been found by taking a piece of semi-random binary data, converting it to a number \( n \), multiplying it by \( 256^k \) for some positive integer \( k \), and searching for possible primes within the interval \([256^k n + 1, 256^k (n + 1) - 1]\).

<table>
<thead>
<tr>
<th>Type</th>
<th>Prime</th>
<th>Number of decimal digits</th>
<th>Date</th>
<th>Found by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mersenne prime</td>
<td>( 2^{57,885,161} - 1 )</td>
<td>17,425,170</td>
<td>January 25, 2013</td>
<td>Great Internet Mersenne Prime Search</td>
</tr>
<tr>
<td>not a Mersenne prime (Proth number)</td>
<td>( 19,249 \times 2^{13,018,586} + 1 )</td>
<td>3,918,990</td>
<td>March 26, 2007</td>
<td>Seventeen or Bust</td>
</tr>
<tr>
<td>factorial prime</td>
<td>( 150209! + 1 )</td>
<td>712,355</td>
<td>October 2011</td>
<td>PrimeGrid</td>
</tr>
<tr>
<td>primorial prime</td>
<td>( 1098133# - 1 )</td>
<td>476,311</td>
<td>March 2012</td>
<td>PrimeGrid</td>
</tr>
<tr>
<td>twin primes</td>
<td>( 3756801695685 \times 2^{666669} \pm 1 )</td>
<td>200,700</td>
<td>December 2011</td>
<td>PrimeGrid</td>
</tr>
</tbody>
</table>

**Integer factorization**

Given a composite integer \( n \), the task of providing one (or all) prime factors is referred to as *factorization of \( n \).* Elliptic curve factorization is an algorithm relying on arithmetic on an elliptic curve.

**Distribution**

In 1975, number theorist Don Zagier commented that primes both

> grow like weeds among the natural numbers, seeming to obey no other law than that of chance [but also] exhibit stunning regularity [and] that there are laws governing their behavior, and that they obey these laws with almost military precision.

The distribution of primes in the large, such as the question how many primes are smaller than a given, large threshold, is described by the prime number theorem, but no efficient formula for the \( n \)-th prime is known.

There are arbitrarily long sequences of consecutive non-primes, as for every positive integer \( n \) the \( n \) consecutive integers from \((n + 1)! + 2\) to \((n + 1)! + n + 1\) (inclusive) are all composite (as \((n + 1)! + k\) is divisible by \( k \) for \( k \) between \( 2 \) and \( n + 1 \)).

Dirichlet’s theorem on arithmetic progressions, in its basic form, asserts that linear polynomials

\[ p(n) = a + bn \]

with coprime integers \( a \) and \( b \) take infinitely many prime values. Stronger forms of the theorem state that the sum of the reciprocals of these prime values diverges, and that different such polynomials with the same \( b \) have approximately the same proportions of primes.

The corresponding question for quadratic polynomials is less well-understood.
Formulas for primes

There is no known efficient formula for primes. For example, Mills’ theorem and a theorem of Wright assert that there are real constants \( A > 1 \) and \( \mu \) such that

\[
\left\lfloor A^{3^n} \right\rfloor \quad \text{and} \quad \left\lfloor 2^{n^{2^{\mu}}} \right\rfloor
\]

are prime for any natural number \( n \). Here \( \left\lfloor \cdot \right\rfloor \) represents the floor function, i.e., largest integer not greater than the number in question. The latter formula can be shown using Bertrand’s postulate (proven first by Chebyshev), which states that there always exists at least one prime number \( p \) with \( n < p < 2n - 2 \), for any natural number \( n > 3 \).

However, computing \( A \) or \( \mu \) requires the knowledge of infinitely many primes to begin with.\(^{[11]}\) Another formula is based on Wilson’s theorem and generates the number 2 many times and all other primes exactly once.

There is no non-constant polynomial, even in several variables, that takes only prime values. However, there is a set of Diophantine equations in 9 variables and one parameter with the following property: the parameter is prime if and only if the resulting system of equations has a solution over the natural numbers. This can be used to obtain a single formula with the property that all its positive values are prime.

Number of prime numbers below a given number

The prime counting function \( \pi(n) \) is defined as the number of primes not greater than \( n \). For example \( \pi(11) = 5 \), since there are five primes less than or equal to 11. There are known algorithms to compute exact values of \( \pi(n) \) faster than it would be possible to compute each prime up to \( n \). The prime number theorem states that \( \pi(n) \) is approximately given by

\[
\pi(n) \approx \frac{n}{\ln n},
\]

in the sense that the ratio of \( \pi(n) \) and the right hand fraction approaches 1 when \( n \) grows to infinity. This implies that the likelihood that a number less than \( n \) is prime is (approximately) inversely proportional to the number of digits in \( n \). A more accurate estimate for \( \pi(n) \) is given by the offset logarithmic integral

\[
\text{Li}(n) = \int_2^n \frac{dt}{\ln t}.
\]

The prime number theorem also implies estimates for the size of the \( n \)-th prime number \( p_n \) (i.e., \( p_1 = 2, p_2 = 3 \), etc.): up to a bounded factor, \( p_n \) grows like \( n \log(n) \).\(^{[12]}\) In particular, the prime gaps, i.e. the differences \( p_n - p_{n-1} \) of two consecutive primes, become arbitrarily large. This latter statement can also be seen in a more elementary way by noting that the sequence \( n! + 2, n! + 3, \ldots, n! + n \) (for the notation \( n! \) read factorial) consists of \( n - 1 \) composite numbers, for any natural number \( n \).
**Arithmetic progressions**

An arithmetic progression is the set of natural numbers that give the same remainder when divided by some fixed number \( q \) called modulus. For example,

\[
3, 12, 21, 30, 39, \ldots
\]

is an arithmetic progression modulo \( q = 9 \). Except for 3, none of these numbers is prime, since \( 3 + 9n = 3(1 + 3n) \) so that the remaining numbers in this progression are all composite. (In general terms, all prime numbers above \( q \) are of the form \( q\#n + m \), where \( 0 < m < q\# \), and \( m \) has no prime factor \( \leq q \).) Thus, the progression

\[
a, a + q, a + 2q, a + 3q, \ldots
\]

can have infinitely many primes only when \( a \) and \( q \) are coprime, i.e., their greatest common divisor is one. If this necessary condition is satisfied, *Dirichlet's theorem on arithmetic progressions* asserts that the progression contains infinitely many primes. The picture below illustrates this with \( q = 9 \): the numbers are "wrapped around" as soon as a multiple of 9 is passed. Primes are highlighted in red. The rows (=progressions) starting with \( a = 3, 6, \) or \( 9 \) contain at most one prime number. In all other rows (\( a = 1, 2, 4, 5, 7, \) and \( 8 \)) there are infinitely many prime numbers. What is more, the primes are distributed equally among those rows in the long run—the density of all primes congruent \( a \) modulo 9 is \( 1/6 \).

The Green–Tao theorem shows that there are arbitrarily long arithmetic progressions consisting of primes. An odd prime \( p \) is expressible as the sum of two squares, \( p = x^2 + y^2 \), exactly if \( p \) is congruent 1 modulo 4 (Fermat's theorem on sums of two squares).

**Prime values of quadratic polynomials**

Euler noted that the function

\[
n^2 + n + 41
\]

gives prime numbers for \( 0 \leq n < 40 \),\(^{13}\)\(^{14}\) a fact leading into deep algebraic number theory, more specifically Heegner numbers. For bigger \( n \), it does take composite values. The Hardy-Littlewood conjecture F makes an asymptotic prediction about the density of primes among the values of quadratic polynomials (with integer coefficients \( a, b, \) and \( c \))

\[
f(n) = ax^2 + bx + c
\]

in terms of \( \text{Li}(n) \) and the coefficients \( a, b, \) and \( c \). However, progress has proved hard to come by: no quadratic polynomial (with \( a \neq 0 \)) is known to take infinitely many prime values. The Ulam spiral depicts all natural numbers in a spiral-like way. Surprisingly, prime numbers cluster on certain diagonals and not others, suggesting that some quadratic polynomials take prime values more often than other ones.
Open questions

Zeta function and the Riemann hypothesis

The Riemann zeta function $\zeta(s)$ is defined as an infinite sum

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s},$$

where $s$ is a complex number with real part bigger than 1. It is a consequence of the fundamental theorem of arithmetic that this sum agrees with the infinite product

$$\prod_{\text{prime}} \frac{1}{1 - p^{-s}}.$$

The zeta function is closely related to prime numbers. For example, the aforementioned fact that there are infinitely many primes can also be seen using the zeta function: if there were only finitely many primes then $\zeta(1)$ would have a finite value. However, the harmonic series $1 + 1/2 + 1/3 + 1/4 + \ldots$ diverges (i.e., exceeds any given number), so there must be infinitely many primes. Another example of the richness of the zeta function and a glimpse of modern algebraic number theory is the following identity (Basel problem), due to Euler,

$$\zeta(2) = \prod_{\text{prime}} \frac{1}{1 - p^{-2}} = \frac{\pi^2}{6}.$$

The reciprocal of $\zeta(2), 6/\pi^2$, is the probability that two numbers selected at random are relatively prime.\[15\]

The unproven Riemann hypothesis, dating from 1859, states that except for $s = -2, -4, \ldots$, all zeroes of the $\zeta$-function have real part equal to 1/2. The connection to prime numbers is that it essentially says that the primes are as regularly distributed as possible. Wikipedia:Please clarify From a physical viewpoint, it roughly states that the irregularity in the distribution of primes only comes from random noise. From a mathematical viewpoint, it roughly states that the asymptotic distribution of primes (about $x/\log x$ of numbers less than $x$ are primes, the prime number theorem) also holds for much shorter intervals of length about the square root of $x$ (for intervals near $x$). This hypothesis is generally believed to be correct. In particular, the simplest assumption is that primes should have no significant irregularities without good reason.

Other conjectures

In addition to the Riemann hypothesis, many more conjectures revolving about primes have been posed. Often having an elementary formulation, many of these conjectures have withstood a proof for decades: all four of Landau's problems from 1912 are still unsolved. One of them is Goldbach's conjecture, which asserts that every even integer $n$ greater than 2 can be written as a sum of two primes. As of February 2011[1], this conjecture has been verified for all numbers up to $n = 2 \cdot 10^{17}$. Weaker statements than this have been proven, for example Vinogradov's theorem says that every sufficiently large odd integer can be written as a sum of three primes. Chen's theorem says that every sufficiently large even number can be expressed as the sum of a prime and a semiprime, the product of two primes. Also, any even integer can be written as the sum of six primes. The branch of number theory studying such questions is called additive number theory.

Other conjectures deal with the question whether an infinity of prime numbers subject to certain constraints exists. It is conjectured that there are infinitely many Fibonacci primes[16] and infinitely many Mersenne primes, but not Fermat primes.\[17\] It is not known whether or not there are an infinite number of Wieferich primes and of prime Euclid numbers.
A third type of conjectures concerns aspects of the distribution of primes. It is conjectured that there are infinitely many twin primes, pairs of primes with difference 2 (twin prime conjecture). Polignac's conjecture is a strengthening of that conjecture, it states that for every positive integer \( n \), there are infinitely many pairs of consecutive primes that differ by \( 2n \).\[^{[18]}\] It is conjectured there are infinitely many primes of the form \( n^2 + 1 \). These conjectures are special cases of the broad Schinzel's hypothesis H. Brocard's conjecture says that there are always at least four primes between the squares of consecutive primes greater than 2. Legendre's conjecture states that there is a prime number between \( n^2 \) and \( (n + 1)^2 \) for every positive integer \( n \). It is implied by the stronger Cramér's conjecture.

### Applications

For a long time, number theory in general, and the study of prime numbers in particular, was seen as the canonical example of pure mathematics, with no applications outside of the self-interest of studying the topic. In particular, number theorists such as British mathematician G. H. Hardy prided themselves on doing work that had absolutely no military significance.\[^{[19]}\] However, this vision was shattered in the 1970s, when it was publicly announced that prime numbers could be used as the basis for the creation of public key cryptography algorithms. Prime numbers are also used for hash tables and pseudorandom number generators.

Some rotor machines were designed with a different number of pins on each rotor, with the number of pins on any one rotor either prime, or coprime to the number of pins on any other rotor. This helped generate the full cycle of possible rotor positions before repeating any position.

The International Standard Book Numbers work with a check digit, which exploits the fact that 11 is a prime.

### Arithmetic modulo a prime and finite fields

*Modular arithmetic* modifies usual arithmetic by only using the numbers

\[
\{0, 1, 2, \ldots, n - 1\},
\]

where \( n \) is a fixed natural number called modulus. Calculating sums, differences and products is done as usual, but whenever a negative number or a number greater than \( n - 1 \) occurs, it gets replaced by the remainder after division by \( n \). For instance, for \( n = 7 \), the sum \( 3 + 5 \) is 1 instead of 8, since 8 divided by 7 has remainder 1. This is referred to by saying “3 + 5 is congruent to 1 modulo 7” and is denoted

\[
3 + 5 \equiv 1 \pmod{7}.
\]

Similarly, \( 6 + 1 \equiv 0 \pmod{7} \), \( 2 - 5 \equiv 4 \pmod{7} \), since \( -3 + 7 = 4 \), and \( 3 \cdot 4 \equiv 5 \pmod{7} \) as 12 has remainder 5.

Standard properties of addition and multiplication familiar from the integers remain valid in modular arithmetic. In the parlance of abstract algebra, the above set of integers, which is also denoted \( \mathbb{Z}/n\mathbb{Z} \), is therefore a commutative ring for any \( n \). Division, however, is not in general possible in this setting. For example, for \( n = 6 \), the equation

\[
3 \cdot x \equiv 2 \pmod{6},
\]

a solution \( x \) of which would be an analogue of 2/3, cannot be solved, as one can see by calculating \( 3 \cdot 0, 3 \cdot 1 \) modulo 6. The distinctive feature of prime numbers is the following: division is possible in modular arithmetic if and only if \( n \) is a prime. Equivalently, \( n \) is prime if and only if all integers \( m \) satisfying \( 2 \leq m \leq n - 1 \) are coprime to \( n \), i.e. their only common divisor is one. Indeed, for \( n = 7 \), the equation

\[
3 \cdot x \equiv 2 \pmod{7},
\]

has a unique solution, \( x = 3 \). Because of this, for any prime \( p \), \( \mathbb{Z}/p\mathbb{Z} \) (also denoted \( \mathbb{F}_p \)) is called a field or, more specifically, a finite field since it contains finitely many, namely \( p \), elements.

A number of theorems can be derived from inspecting \( \mathbb{F}_p \) in this abstract way. For example, Fermat's little theorem, stating

\[
a^{p-1} \equiv 1 \pmod{p}
\]

for any integer \( a \) not divisible by \( p \), may be proved using these notions. This implies
\[ \sum_{a=1}^{p-1} a^{p-1} \equiv (p-1) \cdot 1 \equiv -1 \pmod{p}. \]

Giuga's conjecture says that this equation is also a sufficient condition for \( p \) to be prime. Another consequence of Fermat's little theorem is the following: if \( p \) is a prime number other than 2 and 5, \( \frac{1}{p} \) is always a recurring decimal, whose period is \( p-1 \) or a divisor of \( p-1 \). The fraction \( \frac{1}{p} \) expressed likewise in base \( q \) (rather than base 10) has similar effect, provided that \( p \) is not a prime factor of \( q \). Wilson's theorem says that an integer \( p > 1 \) is prime if and only if the factorial \( (p-1)! + 1 \) is divisible by \( p \). Moreover, an integer \( n > 4 \) is composite if and only if \( (n-1)! \) is divisible by \( n \).

**Other mathematical occurrences of primes**

Many mathematical domains make great use of prime numbers. An example from the theory of finite groups are the Sylow theorems: if \( G \) is a finite group and \( p^n \) is the highest power of the prime \( p \) that divides the order of \( G \), then \( G \) has a subgroup of order \( p^n \). Also, any group of prime order is cyclic (Lagrange's theorem).

**Public-key cryptography**

Several public-key cryptography algorithms, such as RSA and the Diffie–Hellman key exchange, are based on large prime numbers (for example 512 bit primes are frequently used for RSA and 1024 bit primes are typical for Diffie–Hellman.). RSA relies on the assumption that it is much easier (i.e., more efficient) to perform the multiplication of two (large) numbers \( x \) and \( y \) than to calculate \( x \) and \( y \) (assumed coprime) if only the product \( xy \) is known. The Diffie–Hellman key exchange relies on the fact that there are efficient algorithms for modular exponentiation, while the reverse operation the discrete logarithm is thought to be a hard problem.

**Prime numbers in nature**

Inevitably, some of the numbers that occur in nature are prime. There are, however, relatively few examples of numbers that appear in nature because they are prime.

One example of the use of prime numbers in nature is as an evolutionary strategy used by cicadas of the genus *Magicicada*. These insects spend most of their lives as grubs underground. They only pupate and then emerge from their burrows after 13 or 17 years, at which point they fly about, breed, and then die after a few weeks at most. The logic for this is believed to be that the prime number intervals between emergences make it very difficult for predators to evolve that could specialize as predators on *Magicicadas*. If *Magicicadas* appeared at a non-prime number intervals, say every 12 years, then predators appearing every 2, 3, 4, 6, or 12 years would be sure to meet them. Over a 200-year period, average predator populations during hypothetical outbreaks of 14- and 15-year cicadas would be up to 2% higher than during outbreaks of 13- and 17-year cicadas. Though small, this advantage appears to have been enough to drive natural selection in favour of a prime-numbered life-cycle for these insects.

There is speculation that the zeros of the zeta function are connected to the energy levels of complex quantum systems.

**Generalizations**

The concept of prime number is so important that it has been generalized in different ways in various branches of mathematics. Generally, "prime" indicates minimality or indecomposability, in an appropriate sense. For example, the prime field is the smallest subfield of a field \( F \) containing both 0 and 1. It is either \( \mathbb{Q} \) or the finite field with \( p \) elements, whence the name. Often a second, additional meaning is intended by using the word prime, namely that any object can be, essentially uniquely, decomposed into its prime components. For example, in knot theory, a prime knot is a knot that is indecomposable in the sense that it cannot be written as the knot sum of two nontrivial knots. Any knot can be uniquely expressed as a connected sum of prime knots. Prime models and prime 3-manifolds are
other examples of this type.

**Prime elements in rings**

Prime numbers give rise to two more general concepts that apply to elements of any commutative ring \( R \), an algebraic structure where addition, subtraction and multiplication are defined: prime elements and irreducible elements. An element \( p \) of \( R \) is called prime element if it is neither zero nor a unit (i.e., does not have a multiplicative inverse) and satisfies the following requirement: given \( x \) and \( y \) in \( R \) such that \( p \) divides the product \( xy \), then \( p \) divides \( x \) or \( y \). An element is irreducible if it cannot be written as a product of two ring elements that are not units. In the ring \( \mathbb{Z} \) of integers, the set of prime elements equals the set of irreducible elements, which is

\[
{\ldots, -11, -7, -5, -3, -2, 2, 3, 5, 7, 11, \ldots}.
\]

In any ring \( R \), any prime element is irreducible. The converse does not hold in general, but does hold for unique factorization domains.

The fundamental theorem of arithmetic continues to hold in unique factorization domains. An example of such a domain is the Gaussian integers \( \mathbb{Z}[i] \), that is, the set of complex numbers of the form \( a + bi \) where \( i \) denotes the imaginary unit and \( a \) and \( b \) are arbitrary integers. Its prime elements are known as Gaussian primes. Not every prime (in \( \mathbb{Z} \)) is a Gaussian prime: in the bigger ring \( \mathbb{Z}[i] \), 2 factors into the product of the two Gaussian primes \((1 + i)\) and \((1 - i)\). Rational primes (i.e. prime elements in \( \mathbb{Z} \)) of the form \( 4k + 3 \) are Gaussian primes, whereas rational primes of the form \( 4k + 1 \) are not.

**Prime ideals**

In ring theory, the notion of number is generally replaced with that of ideal. Prime ideals, which generalize prime elements in the sense that the principal ideal generated by a prime element is a prime ideal, are an important tool and object of study in commutative algebra, algebraic number theory and algebraic geometry. The prime ideals of the ring of integers are the ideals \((0), (2), (3), (5), (7), (11), \ldots\) The fundamental theorem of arithmetic generalizes to the Lasker–Noether theorem, which expresses every ideal in a Noetherian commutative ring as an intersection of primary ideals, which are the appropriate generalizations of prime powers.

Prime ideals are the points of algebro-geometric objects, via the notion of the spectrum of a ring.[22] Arithmetic geometry also benefits from this notion, and many concepts exist in both geometry and number theory. For example, factorization or ramification of prime ideals when lifted to an extension field, a basic problem of algebraic number theory, bears some resemblance with ramification in geometry. Such ramification questions occur even in number-theoretic questions solely concerned with integers. For example, prime ideals in the ring of integers of quadratic number fields can be used in proving quadratic reciprocity, a statement that concerns the solvability of quadratic equations

\[
x^2 \equiv p \pmod{q},
\]

where \( x \) is an integer and \( p \) and \( q \) are (usual) prime numbers.[23] Early attempts to prove Fermat's Last Theorem climaxed when Kummer introduced regular primes, primes satisfying a certain requirement concerning the failure of unique factorization in the ring consisting of expressions

\[
a_0 + a_1 \zeta + \cdots + a_{p-1} \zeta^{p-1},
\]

where \( a_0, \ldots, a_{p-1} \) are integers and \( \zeta \) is a complex number such that \( \zeta^p = 1 \).[24]
Valuations

Valuation theory studies certain functions from a field \( K \) to the real numbers \( \mathbb{R} \) called valuations.[25] Every such valuation yields a topology on \( K \), and two valuations are called equivalent if they yield the same topology. A prime of \( K \) (sometimes called a place of \( K \)) is an equivalence class of valuations. For example, the \( p \)-adic valuation of a rational number \( q \) is defined to be the integer \( v_p(q) \), such that

\[
q = p^{v_p(q)} \frac{r}{s},
\]

where both \( r \) and \( s \) are not divisible by \( p \). For example, \( v_3(18/7) = 2 \). The \( p \)-adic norm is defined as[26]

\[
|q|_p := p^{-v_p(q)}.
\]

In particular, this norm gets smaller when a number is multiplied by \( p \), in sharp contrast to the usual absolute value (also referred to as the infinite prime). While completing \( \mathbb{Q} \) (roughly, filling the gaps) with respect to the absolute value yields the field of real numbers, completing with respect to the \( p \)-adic norm \( |\cdot|_p \) yields the field of \( p \)-adic numbers.[27] These are essentially all possible ways to complete \( \mathbb{Q} \), by Ostrowski's theorem. Certain arithmetic questions related to \( \mathbb{Q} \) or more general global fields may be transferred back and forth to the completed (or local) fields. This local-global principle again underlines the importance of primes to number theory.

In the arts and literature

Prime numbers have influenced many artists and writers. The French composer Olivier Messiaen used prime numbers to create ametrical music through "natural phenomena". In works such as \textit{La Nativité du Seigneur} (1935) and \textit{Quatre études de rythme} (1949–50), he simultaneously employs motifs with lengths given by different prime numbers to create unpredictable rhythms: the primes 41, 43, 47 and 53 appear in the third étude, "Neumes rythmiques". According to Messiaen this way of composing was "inspired by the movements of nature, movements of free and unequal durations".

In his science fiction novel \textit{Contact}, NASA scientist Carl Sagan suggested that prime numbers could be used as a means of communicating with aliens, an idea that he had first developed informally with American astronomer Frank Drake in 1975.[28] In the novel \textit{The Curious Incident of the Dog in the Night-Time} by Mark Haddon, the narrator arranges the sections of the story by consecutive prime numbers.[29]

Many films, such as \textit{Cube}, \textit{Sneakers}, \textit{The Mirror Has Two Faces} and \textit{A Beautiful Mind} reflect a popular fascination with the mysteries of prime numbers and cryptography.[30] Prime numbers are used as a metaphor for loneliness and isolation in the Paolo Giordano novel \textit{The Solitude of Prime Numbers}, in which they are portrayed as "outsiders" among integers.

Notes

[5] "Why is the number one not prime?" (http://primes.utm.edu/notes/faq/one.html)
[7] For instance, Beiler writes that number theorist Ernst Kummer loved his ideal numbers, closely related to the primes, "because they had not soiled themselves with any practical applications", and Katz writes that Edmund Landau, known for his work on the distribution of primes, "loathed practical applications of mathematics", and for this reason avoided subjects such as geometry that had already shown themselves to be useful. . .
Prime number

References


[10] , Section 1.6, Theorem 1.13
[12] , Section 4.6, Theorem 4.7
[13] Hua (2009), "
[14] See list of values (http://www.wolframalpha.com/input/?i=evaluate+x^2+for+x+from+0+to+40), calculated by Wolfram Alpha
[17] E.g., see
[18] . p. 112
[19] "No one has yet discovered any warlike purpose to be served by the theory of numbers or relativity, and it seems unlikely that anyone will do so for many years."
[20] , Section II.1, p. 90
[22] Shafarevich, Basic Algebraic Geometry volume 2 (Schemes and Complex Manifolds), p. 5, section V.1
[23] Neukirch, Algebraic Number theory, p. 50, Section I.8
[24] Neukirch, Algebraic Number theory, p. 38, Section I.7
[26] Some sources also put \( \sqrt{n} \).
[27] Gouvea: p-adic numbers: an introduction, Chapter 3, p. 43
• Lehmer, D. H. (1909), *Factor table for the first ten millions containing the smallest factor of every number not divisible by 2, 3, 5, or 7 between the limits 0 and 10017000*, Washington, D.C.: Carnegie Institution of Washington

Further references


External links

• Caldwell, Chris, The Prime Pages at primes.utm.edu (http://primes.utm.edu/).
• Prime Numbers (http://www.bbc.co.uk/programmes/p003hyf5) on *In Our Time* at the BBC.
• An Introduction to Analytic Number Theory, by Ilan Vardi and Cyril Banderier (http://www.maths.ex.ac.uk/~mwatkins/zeta/vardi.html)
• Plus teacher and student package: prime numbers (http://plus.maths.org/issue49/package/index.html) from Plus, the free online mathematics magazine produced by the Millennium Mathematics Project at the University of Cambridge.

Prime number generators and calculators

• Prime Number Checker (http://www.had2know.com/academics/prime-composite.html) identifies the smallest prime factor of a number.
• Fast Online primality test (http://www.alpertron.com.ar/ECM.HTM) makes use of the Elliptic Curve Method (up to thousand-digits numbers, requires Java).
• Prime Number Generator (http://publicliterature.org/tools/prime_number_generator) generates a given number of primes above a given start number.
• Huge database of prime numbers (http://www.bigprimes.net/)
• Prime Numbers up to 1 trillion (http://www.primos.mat.br/indexen.html)
Miller–Rabin primality test

The Miller–Rabin primality test or Rabin–Miller primality test is a primality test: an algorithm which determines whether a given number is prime, similar to the Fermat primality test and the Solovay–Strassen primality test. Its original version, due to Gary L. Miller, is deterministic, but the determinism relies on the unproven generalized Riemann hypothesis; Michael O. Rabin modified it to obtain an unconditional probabilistic algorithm.

Concepts

Just like the Fermat and Solovay–Strassen tests, the Miller–Rabin test relies on an equality or set of equalities that hold true for prime values, then checks whether or not they hold for a number that we want to test for primality.

First, a lemma about square roots of unity in the finite field $\mathbb{Z}/p\mathbb{Z}$, where $p$ is prime and $p > 2$. Certainly 1 and $-1$ always yield 1 when squared modulo $p$; call these trivial square roots of 1. There are no nontrivial square roots of 1 modulo $p$ (a special case of the result that, in a field, a polynomial has no more zeroes than its degree). To show this, suppose that $x$ is a square root of 1 modulo $p$. Then:

\[
x^2 \equiv 1 \pmod{p} \\
(x - 1)(x + 1) \equiv 0 \pmod{p}.
\]

In other words, prime $p$ divides the product $(x - 1)(x + 1)$. By Euclid's lemma it divides one of the factors $x - 1$ or $x + 1$, implying that $x$ is congruent to either 1 or $-1$ modulo $p$.

Now, let $n$ be prime with $n > 2$. It follows that $n - 1$ is even and we can write it as $2^s d$, where $s$ and $d$ are positive integers ($d$ is odd). For each $a$ in $\left(\mathbb{Z}/n\mathbb{Z}\right)^*$, either

\[
a^d \equiv 1 \pmod{n}
\]

or

\[
a^{2^r d} \equiv -1 \pmod{n}
\]

for some $0 \leq r \leq s - 1$.

To show that one of these must be true, recall Fermat's little theorem, that for a prime number $n$:

\[
a^{n-1} \equiv 1 \pmod{n}.
\]

By the lemma above, if we keep taking square roots of $a^{n-1}$, we will get either 1 or $-1$. If we get $-1$ then the second equality holds and it is done. If we never get $-1$, then when we have taken out every power of 2, we are left with the first equality.

The Miller–Rabin primality test is based on the contrapositive of the above claim. That is, if we can find an $a$ such that

\[
a^d \not\equiv 1 \pmod{n}
\]

and

\[
a^{2^r d} \not\equiv -1 \pmod{n}
\]

for all $0 \leq r \leq s - 1$, then $n$ is not prime. We call $a$ a witness for the compositeness of $n$ (sometimes misleadingly called a strong witness, although it is a certain proof of this fact). Otherwise $a$ is called a strong liar, and $n$ is a strong probable prime to base $a$. The term "strong liar" refers to the case where $n$ is composite but nevertheless the equations hold as they would for a prime.

Every odd composite $n$ has many witnesses $a$, however, no simple way of generating such an $a$ is known. The solution is to make the test probabilistic: we choose a non-zero $a$ in $\mathbb{Z}/n\mathbb{Z}$ randomly, and check whether or not it is a witness for the compositeness of $n$. If $n$ is composite, most of the choices for $a$ will be witnesses, and the test will detect $n$ as composite with high probability. There is, nevertheless, a small chance that we are unlucky and hit an $a$ which is a strong liar for $n$. We may reduce the probability of such error by repeating the test for several
Miller-Rabin primality test independently chosen \( a \).

It is crucial to choose random bases \( a \), as, a priori, we don’t know the distribution of witnesses and liars among the numbers 1, 2, ..., \( n - 1 \). In particular, Arnault gave a 397-digit composite number for which all bases \( a \) less than 307 are strong liars; in particular, this number was reported to be prime by the Maple `isprime()` function, which incorrectly implemented the Miller–Rabin test by checking the specific bases 2,3,5,7, and 11 instead of checking random bases.

**Example**

Suppose we wish to determine if \( n = 221 \) is prime. We write \( n - 1 = 220 \) as \( 2^2 \cdot 55 \), so that we have \( s = 2 \) and \( d = 55 \).

We randomly select a number \( a \) such that \( a < n \), say \( a = 174 \). We proceed to compute:

- \( a^{2^0 \cdot 55} \mod n = 174^{55} \mod 221 = 47 \neq 1, n - 1 \)
- \( a^{2^1 \cdot 55} \mod n = 174^{110} \mod 221 = 220 = n - 1 \)

Since \( 220 \equiv -1 \mod n \), either 221 is prime, or 174 is a strong liar for 221. We try another random \( a \), this time choosing \( a = 137 \):

- \( a^{2^0 \cdot 55} \mod n = 137^{55} \mod 221 = 188 \neq 1, n - 1 \)
- \( a^{2^1 \cdot 55} \mod n = 137^{110} \mod 221 = 205 \neq n - 1 \)

Hence 137 is a witness for the compositeness of 221, and 174 was in fact a strong liar. Note that this tells us nothing about the factors of 221 (which are 13 and 17). However, the example with 341 in the next section shows how these calculations can sometimes produce a factor of \( n \).

**Algorithm and running time**

The algorithm can be written in pseudocode as follows:

```
Input: \( n > 3 \), an odd integer to be tested for primality;
Input: \( k \), a parameter that determines the accuracy of the test
Output: composite if \( n \) is composite, otherwise probably prime
write \( n - 1 \) as \( 2^s \cdot d \) with \( d \) odd by factoring powers of 2 from \( n - 1 \)
WitnessLoop: repeat \( k \) times:
    pick a random integer \( a \) in the range \([2, n - 2]\)
    \( x \leftarrow a^d \mod n \)
    if \( x = 1 \) or \( x = n - 1 \) then do next WitnessLoop
    repeat \( s - 1 \) times:
        \( x \leftarrow x^2 \mod n \)
        if \( x = 1 \) then return composite
        if \( x = n - 1 \) then do next WitnessLoop
    return composite
return probably prime
```

Using modular exponentiation by repeated squaring, the running time of this algorithm is \( O(k \log^3 n) \), where \( k \) is the number of different values of \( a \) that we test; thus this is an efficient, polynomial-time algorithm. FFT-based multiplication can push the running time down to \( O(k \log^2 n \log \log n \log \log \log n) = O(k \log^2 n) \).

If we insert Greatest common divisor (GCD) calculations into the above algorithm, we can sometimes obtain a factor of \( n \) instead of merely determining that \( n \) is composite. In particular, if \( n \) is a probable prime base \( a \) but not a strong probable prime base \( a \), then either GCD\((a^d \mod n - 1, n)\) or (for some \( r \) in the above range) GCD\((a^{2r \cdot d} \mod n - 1, n)\) will produce a (not necessarily prime) factor of \( n \); see page 1402 of. If factoring is a goal, these GCDs can be inserted into the above algorithm at little additional computational cost.
For example, consider $n = 341$. We have $n - 1 = 85$. Then $2^{85} \mod 341 = 32$. This tells us that $n$ is not a strong probable prime base 2, so we know $n$ is composite. If we take a GCD at this stage, we can get a factor of 341: $\text{GCD}((2^{85} \mod 341) - 1, 341) = 31$. This works because 341 is a pseudoprime base 2, but is not a strong pseudoprime base 2.

In the case that the algorithm returns "composite" because $x = 1$, it has also discovered that $d2^r$ is (an odd multiple of) the order of $a$—a fact which can (as in Shor's algorithm) be used to factorize $n$, since $n$ then divides

$$a^{d2^r} - 1 = (a^{d2^{r-1}} - 1)(a^{d2^{r-1}} + 1)$$

but not either factor by itself. The reason Miller–Rabin does not yield a probabilistic factorization algorithm is that if

$$a^{n-1} \not\equiv 1 \pmod{n}$$

(i.e., $n$ is not a pseudoprime to base $a$) then no such information is obtained about the period of $a$, and the second "return composite" is taken.

**Accuracy of the test**

The more bases $a$ we test, the better the accuracy of the test. It can be shown that for any odd composite $n$, at least 3/4 of the bases $a$ are witnesses for the compositeness of $n$. The Miller–Rabin test is strictly stronger than the Solovay–Strassen primality test in the sense that for every composite $n$, the set of strong liars for $n$ is a subset of the set of Euler liars for $n$, and for many $n$, the subset is proper. If $n$ is composite then the Miller–Rabin primality test declares $n$ probably prime with a probability at most $4^{-k}$. On the other hand, the Solovay–Strassen primality test declares $n$ probably prime with a probability at most $2^{-k}$.

It is important to note that in many common applications of this algorithm, we are not interested in the error bound described above. The above error bound is the probability of a composite number being declared as a probable prime after $k$ rounds of testing. We are often instead interested in the probability that, after passing $k$ rounds of testing, the number being tested is actually a composite number. Formally, if we call the event of declaring $n$ a probable prime after $k$ rounds of Miller–Rabin $Y_k$, and we call the event that $n$ is composite $X$, then the above bound gives us $P(Y_k | X)$, whereas we are interested in $P(X | Y_k)$. Bayes' theorem gives us a way to relate these two conditional probabilities, namely

$$P(X | Y_k) = \frac{P(Y_k | X)P(X)}{P(Y_k | X)P(X) + P(Y_k | \overline{X})P(\overline{X})}.$$ 

This tells us that the probability that we are often interested in is related not just to the $4^{-k}$ bound above, but also probabilities related to the density of prime numbers in the region near $n$.

In addition, for large values of $n$, on average the probability that a composite number is declared *probably prime* is significantly smaller than $4^{-k}$. Damgård, Landrock and Pomerance compute some explicit bounds and provide a method to make a reasonable selection for $k$ for a desired error bound. Such bounds can, for example, be used to *generate* probable primes; however, they should not be used to *verify* primes with unknown origin, since in cryptographic applications an adversary might try to send you a pseudoprime in a place where a prime number is required. In such cases, only the error bound of $4^{-k}$ can be relied upon.
Deterministic variants of the test

The Miller–Rabin algorithm can be made deterministic by trying all possible \( a \) below a certain limit. The problem in general is to set the limit so that the test is still reliable.

If the tested number \( n \) is composite, the strong liars \( a \) coprime to \( n \) are contained in a proper subgroup of the group \((\mathbb{Z}/n\mathbb{Z})^*\), which means that if we test all \( a \) from a set which generates \((\mathbb{Z}/n\mathbb{Z})^*\), one of them must be a witness for the compositeness of \( n \). Assuming the truth of the generalized Riemann hypothesis (GRH), it is known that the group is generated by its elements smaller than \( O((\log n)^2) \), which was already noted by Miller. The constant involved in the Big O notation was reduced to 2 by Eric Bach. This leads to the following conditional primality testing algorithm:

\[
\text{Input: } n > 1, \text{ an odd integer to test for primality.} \\
\text{Output: } \text{composite if } n \text{ is composite, otherwise prime} \\
\text{write } n-1 \text{ as } 2^s \cdot d \text{ by factoring powers of 2 from } n-1 \\
\text{repeat for all } a \in [2, \min(n - 1, \lfloor 2(\ln n)^2 \rfloor)]: \\
\quad \text{if } a^d \not\equiv 1 \pmod{n} \text{ and } a^{2^r \cdot d} \not\equiv -1 \pmod{n} \text{ for all } r \in [0, s - 1] \\
\quad \text{then return } \text{composite} \\
\text{return } \text{prime} 
\]

The running time of the algorithm is \( \tilde{O}((\log n)^4) \) (with FFT-based multiplication). The full power of the generalized Riemann hypothesis is not needed to ensure the correctness of the test: as we deal with subgroups of even index, it suffices to assume the validity of GRH for quadratic Dirichlet characters.

This algorithm is not used in practice, as it is much slower than the randomized version of the Miller–Rabin test. For theoretical purposes, it was superseded by the AKS primality test, which does not rely on unproven assumptions.

When the number \( n \) to be tested is small, trying all \( a < 2(\ln n)^2 \) is not necessary, as much smaller sets of potential witnesses are known to suffice. For example, Pomerance, Selfridge and Wagstaff and Jaeschke have verified that

- if \( n < 1,373,653 \), it is enough to test \( a = 2 \) and 3;
- if \( n < 9,080,191 \), it is enough to test \( a = 31 \) and 73;
- if \( n < 4,759,123,141 \), it is enough to test \( a = 2, 7, \) and 61;
- if \( n < 1,122,004,669,633 \), it is enough to test \( a = 2, 13, 23, \) and 1662803;
- if \( n < 2,152,302,898,747 \), it is enough to test \( a = 2, 3, 5, 7, \) and 11;
- if \( n < 3,474,749,660,383 \), it is enough to test \( a = 2, 3, 5, 7, 11, \) and 13;
- if \( n < 341,550,071,728,321 \), it is enough to test \( a = 2, 3, 5, 7, 11, 13, \) and 17.

Other criteria of this sort exist and these results give very fast deterministic primality tests for numbers in the appropriate range, without any assumptions.

There is a small list of potential witnesses for every possible input size (at most \( n \) values for \( n \)-bit numbers). However, no finite set of bases is sufficient for all composite numbers. Alford, Granville, and Pomerance have shown that there exist infinitely many composite numbers \( n \) whose smallest compositeness witness is at least \( (\ln n)^{1/(3 \ln \ln \ln n)} \). They also argue heuristically that the smallest number \( w \) such that every composite number below \( n \) has a compositeness witness less than \( w \) should be of order \( \Theta(\log n \log \log n) \).
Notes

External links

• Interactive Online Implementation of the Deterministic Variant (http://gandraxa.com/miller_rabin_primality_test.xml) (stepping through the algorithm step-by-step)
• Applet (German) (http://www.informationsuebertragung.ch/indexAlgorithmenRabinMiller.html)
• Miller-Rabin primality test in C# (http://stackoverflow.com/questions/4236673/sample-code-for-fast-primality-testing-in-c-sharp/4236870#4236870)
• Miller-Rabin primality test in JavaScript using arbitrary precision arithmetic (http://www.javascripter.net/math/primes/millerrabinprimalitytest.htm)

Modular arithmetic

In mathematics, modular arithmetic (sometimes called clock arithmetic) is a system of arithmetic for integers, where numbers "wrap around" upon reaching a certain value—the modulus.

The modern approach to modular arithmetic was developed by Carl Friedrich Gauss in his book Disquisitiones Arithmeticae, published in 1801.

A familiar use of modular arithmetic is in the 12-hour clock, in which the day is divided into two 12-hour periods. If the time is 7:00 now, then 8 hours later it will be 3:00. Usual addition would suggest that the later time should be 7 + 8 = 15, but this is not the answer because clock time "wraps around" every 12 hours; in 12-hour time, there is no "15 o'clock". Likewise, if the clock starts at 12:00 (noon) and 21 hours elapse, then the time will be 9:00 the next day, rather than 33:00. Since the hour number starts over after it reaches 12, this is arithmetic modulo 12. 12 is congruent not only to 12 itself, but also to 0, so the time called "12:00" could also be called "0:00", since 0 ≡ 12 mod 12.

Congruence relation

Modular arithmetic can be handled mathematically by introducing a congruence relation on the integers that is compatible with the operations of the ring of integers: addition, subtraction, and multiplication. For a positive integer n, two integers a and b are said to be congruent modulo n, written:

\[ a \equiv b \pmod{n}, \]

if their difference a − b is an integer multiple of n (or n divides a − b). The number n is called the modulus of the congruence.

For example,

\[ 38 \equiv 14 \pmod{12} \]

because 38 − 14 = 24, which is a multiple of 12.

The same rule holds for negative values:
Modular arithmetic

-8 ≡ 7 (mod 5)
2 ≡ -3 (mod 5)
-3 ≡ -8 (mod 5)

Equivalently, \( a \equiv b \pmod{n} \) can also be thought of as asserting that the remainders of the division of both \( a \) and \( b \) by \( n \) are the same. For instance:

38 ≡ 14 (mod 12)

because both 38 and 14 have the same remainder 2 when divided by 12. It is also the case that \( 38 - 14 = 24 \) is an integer multiple of 12, which agrees with the prior definition of the congruence relation.

A remark on the notation: Because it is common to consider several congruence relations for different moduli at the same time, the modulus is incorporated in the notation. In spite of the ternary notation, the congruence relation for a given modulus is binary. This would have been clearer if the notation \( a \equiv_n b \) had been used, instead of the common traditional notation.

The properties that make this relation a congruence relation (respecting addition, subtraction, and multiplication) are the following.

If

\[ a_1 \equiv b_1 \pmod{n} \]

and

\[ a_2 \equiv b_2 \pmod{n}, \]

then:

- \( a_1 + a_2 \equiv b_1 + b_2 \pmod{n} \)
- \( a_1 - a_2 \equiv b_1 - b_2 \pmod{n} \)

It should be noted that the above two properties would still hold if the theory were expanded to include all real numbers, that is if \( a_1, a_2, b_1, b_2, n \) were not necessarily all integers. The next property, however, would fail if these variables were not all integers:

- \( a_1 a_2 \equiv b_1 b_2 \pmod{n} \).

Remainders

The notion of modular arithmetic is related to that of the remainder in Euclidean division. The operation of finding the remainder is sometimes referred to as the modulo operation and we may see \( 2 = 14 \pmod{12} \). The difference is in the use of congruency, indicated by "\( \equiv \)" and equality indicated by "\( = \)". Equality implies specifically the "common residue", the least non-negative member of an equivalence class. When working with modular arithmetic, each equivalence class is usually represented by its common residue, for example \( 38 \equiv 2 \pmod{12} \) which can be found using long division. It follows that, while it is correct to say \( 38 \equiv 14 \pmod{12} \), and \( 2 \equiv 14 \pmod{12} \), it is incorrect to say \( 38 \equiv 14 \pmod{12} \) (with "\( = \)" rather than "\( \equiv \)").

The difference is clearest when dividing a negative number, since in that case remainders are negative. Hence to express the remainder we would have to write \(-5 \equiv -17 \pmod{12}\), rather than \(7 \equiv -17 \pmod{12}\), since equivalence can only be said of common residues with the same sign.

In computer science, it is the remainder operator that is usually indicated by either "\%t" (e.g. in C, Java, JavaScript, Perl and Python) or "mod" (e.g. in Pascal, BASIC, SQL, Haskell, ABAP), with exceptions (e.g. Excel). These operators are commonly pronounced as "mod", but it is specifically a remainder that is computed (since in C++ a negative number will be returned if the first argument is negative, and in Python a negative number will be returned if the second argument is negative). The function \( \text{modulo} \) instead of \( \text{mod} \), like \( 38 \equiv 14 \pmod{12} \) is sometimes used to indicate the common residue rather than a remainder (e.g. in Ruby). For details of the specific operations
Modular arithmetic

defined in different languages, see the modulo operation page.
Parentheses are sometimes dropped from the expression, e.g. \( 38 \equiv 14 \mod 12 \) or \( 2 = 14 \mod 12 \), or placed around the divisor e.g. \( 38 \equiv 14 \mod (12) \). Notation such as \( 38(\mod 12) \) has also been observed, but is ambiguous without contextual clarification.

**Functional representation of the remainder operation**

The remainder operation can be represented using the floor function. If \( b \equiv a \pmod{n} \), where \( n > 0 \), then if the remainder \( b \) is calculated

\[
b = a - \left\lfloor \frac{a}{n} \right\rfloor \times n,
\]

where \( \left\lfloor \frac{a}{n} \right\rfloor \) is the largest integer less than or equal to \( \frac{a}{n} \), then

\[
a \equiv b \pmod{n} \text{ and,}
\]

\[
0 \leq b < n.
\]

If instead a remainder \( b \) in the range \(-n \leq b < 0\) is required, then

\[
b = a - \left\lfloor \frac{a}{n} \right\rfloor \times n - n.
\]

**Residue systems**

Each residue class modulo \( n \) may be represented by any one of its members, although we usually represent each residue class by the smallest nonnegative integer which belongs to that class (since this is the proper remainder which results from division). Note that any two members of different residue classes modulo \( n \) are incongruent modulo \( n \). Furthermore, every integer belongs to one and only one residue class modulo \( n \).

The set of integers \( \{0, 1, 2, \ldots, n - 1\} \) is called the least residue system modulo \( n \). Any set of \( n \) integers, no two of which are congruent modulo \( n \), is called a complete residue system modulo \( n \).

It is clear that the least residue system is a complete residue system, and that a complete residue system is simply a set containing precisely one representative of each residue class modulo \( n \). The least residue system modulo 4 is \( \{0, 1, 2, 3\} \). Some other complete residue systems modulo 4 are:

- \( \{1,2,3,4\} \)
- \( \{13,14,15,16\} \)
- \( \{-2,-1,0,1\} \)
- \( \{-13,4,17,18\} \)
- \( \{-5,0,6,21\} \)
- \( \{27,32,37,42\} \)

Some sets which are not complete residue systems modulo 4 are:

- \( \{-5,0,6,22\} \) since 6 is congruent to 22 modulo 4.
- \( \{5,15\} \) since a complete residue system modulo 4 must have exactly 4 incongruent residue classes.
Reduced residue systems

Any set of \( \varphi(n) \) integers that are relatively prime to \( n \) and that are mutually incongruent modulo \( n \), where \( \varphi(n) \) denotes Euler's totient function, is called a reduced residue system modulo \( n \). The example above, \{5,15\} is an example of a reduced residue system modulo 4.

Congruence classes

Like any congruence relation, congruence modulo \( n \) is an equivalence relation, and the equivalence class of the integer \( a \), denoted by \( \overline{a}_n \), is the set \( \{ \ldots, a-2n, a-n, a, a+n, a+2n, \ldots \} \). This set, consisting of the integers congruent to \( a \) modulo \( n \), is called the congruence class or residue class or simply residue of the integer \( a \), modulo \( n \). When the modulus \( n \) is known from the context, that residue may also be denoted \( [a] \).

Integers modulo \( n \)

The set of all congruence classes of the integers for a modulus \( n \) is called the set of integers modulo \( n \), and is denoted \( \mathbb{Z}/n\mathbb{Z} \). The notation \( \mathbb{Z}_n \) is, however, not recommended because it can be confused with the set of \( n \)-adic integers. The set is defined as follows.

\[ \mathbb{Z}/n\mathbb{Z} = \{ \overline{a}_n | a \in \mathbb{Z} \} . \]

When \( n \neq 0 \), \( \mathbb{Z}/n\mathbb{Z} \) has \( n \) elements, and can be written as:

\[ \mathbb{Z}/n\mathbb{Z} = \{ \overline{0}_n, \overline{1}_n, \overline{2}_n, \ldots, \overline{n-1}_n \} . \]

When \( n = 0 \), \( \mathbb{Z}/n\mathbb{Z} \) does not have zero elements; rather, it is isomorphic to \( \mathbb{Z} \), since \( \overline{a}_0 = \{a\} \).

We can define addition, subtraction, and multiplication on \( \mathbb{Z}/n\mathbb{Z} \) by the following rules:

- \( \overline{a}_n + \overline{b}_n = \overline{(a+b)}_n \)
- \( \overline{a}_n - \overline{b}_n = \overline{(a-b)}_n \)
- \( \overline{a}_n \overline{b}_n = \overline{(ab)}_n \)

The verification that this is a proper definition uses the properties given before.

In this way, \( \mathbb{Z}/n\mathbb{Z} \) becomes a commutative ring. For example, in the ring \( \mathbb{Z}/24\mathbb{Z} \), we have

\[ \overline{12}_{24} + \overline{21}_{24} = \overline{9}_{24} \]

as in the arithmetic for the 24-hour clock.

The notation \( \mathbb{Z}/n\mathbb{Z} \) is used, because it is the factor ring of \( \mathbb{Z} \) by the ideal \( n\mathbb{Z} \) containing all integers divisible by \( n \), where \( 0\mathbb{Z} \) is the singleton set \( \{0\} \). Thus \( \mathbb{Z}/n\mathbb{Z} \) is a field when \( n\mathbb{Z} \) is a maximal ideal, that is, when \( n \) is prime.

In terms of groups, the residue class \( \overline{a}_n \) is the coset of \( a \) in the quotient group \( \mathbb{Z}/n\mathbb{Z} \), a cyclic group.\(^1\)

The set \( \mathbb{Z}/n\mathbb{Z} \) has a number of important mathematical properties that are foundational to various branches of mathematics.

Rather than excluding the special case \( n = 0 \), it is more useful to include \( \mathbb{Z}/0\mathbb{Z} \) (which, as mentioned before, is isomorphic to the ring \( \mathbb{Z} \) of integers), for example when discussing the characteristic of a ring.

When \( n \) is prime, the set of integers modulo \( n \) form a finite field.
Applications

Modular arithmetic is referenced in number theory, group theory, ring theory, knot theory, abstract algebra, computer algebra, cryptography, computer science, chemistry and the visual and musical arts.

It is one of the foundations of number theory, touching on almost every aspect of its study, and provides key examples for group theory, ring theory and abstract algebra.

Modular arithmetic is often used to calculate checksums that are used within identifiers - International Bank Account Numbers (IBANs) for example make use of modulo 97 arithmetic to trap user input errors in bank account numbers. In cryptography, modular arithmetic directly underpins public key systems such as RSA and Diffie-Hellman, as well as providing finite fields which underlie elliptic curves, and is used in a variety of symmetric key algorithms including AES, IDEA, and RC4.

In computer algebra, modular arithmetics is commonly used to limit the size of integer coefficients in intermediate calculations and data. It is used in polynomial factorization, a problem for which all known efficient algorithms use modular arithmetic. It is used by the most efficient implementations of polynomial greatest common divisor, exact linear algebra and Gröbner basis algorithms over the integers and the rational numbers.

In computer science, modular arithmetic is often applied in bitwise operations and other operations involving fixed-width, cyclic data structures. The modulo operation, as implemented in many programming languages and calculators, is an application of modular arithmetic that is often used in this context. XOR is the sum of 2 bits, modulo 2.

In chemistry, the last digit of the CAS registry number (a number which is unique for each chemical compound) is a check digit, which is calculated by taking the last digit of the first two parts of the CAS registry number times 1, the next digit times 2, the next digit times 3 etc., adding all these up and computing the sum modulo 10.

In music, arithmetic modulo 12 is used in the consideration of the system of twelve-tone equal temperament, where octave and enharmonic equivalency occurs (that is, pitches in a 1:2 or 2:1 ratio are equivalent, and C-sharp is considered the same as D-flat).

The method of casting out nines offers a quick check of decimal arithmetic computations performed by hand. It is based on modular arithmetic modulo 9, and specifically on the crucial property that 10 \equiv 1 \pmod{9}.

Arithmetic modulo 7 is used in algorithms that determine the day of the week for a given date. In particular, Zeller's congruence and the doomsday algorithm make heavy use of modulo-7 arithmetic.

More generally, modular arithmetic also has application in disciplines such as law (see e.g., apportionment), economics, (see e.g., game theory) and other areas of the social sciences, where proportional division and allocation of resources plays a central part of the analysis.

Computational complexity

Since modular arithmetic has such a wide range of applications, it is important to know how hard it is to solve a system of congruences. A linear system of congruences can be solved in polynomial time with a form of Gaussian elimination, for details see linear congruence theorem. Algorithms, such as Montgomery reduction, also exist to allow simple arithmetic operations, such as multiplication and exponentiation modulo \( n \), to be performed efficiently on large numbers.

Solving a system of non-linear modular arithmetic equations is NP-complete.
Notes

[1] Sengadir T.,

References

• Maarten Bullynck " Modular Arithmetic before C.F. Gauss. Systematisations and discussions on remainder problems in 18th century Germany" (http://www.kuttaka.org/Gauss_Modular.pdf)

External links

• In this modular art (http://britton.disted.camosun.bc.ca/modart/jbmodart.htm) article, one can learn more about applications of modular arithmetic in art.
• Weisstein, Eric W., " Modular Arithmetic (http://mathworld.wolfram.com/ModularArithmetic.html)" , MathWorld.
• An article (http://mersennewiki.org/index.php/modular_arithmetic) on modular arithmetic on the GIMPS wiki
• Modular Arithmetic and patterns in addition and multiplication tables (http://www.cut-the-knot.org/blue/Modulo.shtml)
• Whitney Music Box (http://wheelof.com/whitney)—an audio/video demonstration of integer modular math

Automated modular arithmetic theorem provers:

• Spear (http://www.domagoj-babic.com/index.php/ResearchProjects/Spear)
• AAProver (http://www.lenherr.name/~thomas/ma/aaprover.page) - Simple C++ framework easy to use in applications, supporting (among others) all integer operators present in languages such as C/C++/Java and arbitrary bit-width.
Modular exponentiation

Modular exponentiation is a type of exponentiation performed over a modulus. It is particularly useful in computer science, especially in the field of cryptography.

A "modular exponentiation" calculates the remainder when a positive integer \( b \) (the base) raised to the \( e \)-th power (the exponent), \( b^e \), is divided by a positive integer \( m \), called the modulus. In symbols, this is, given base \( b \), exponent \( e \), and modulus \( m \), the modular exponentiation \( c \) is:

\[
c \equiv b^e \pmod{m}
\]

For example, given \( b = 5 \), \( e = 3 \), and \( m = 13 \), the solution, \( c = 8 \), is the remainder of dividing \( 5^3 \) by 13.

If \( b \), \( e \), and \( m \) are non-negative, and \( b < m \), then a unique solution \( c \) exists with the property \( 0 \leq c < m \).

Modular exponentiation can be performed with a negative exponent \( e \) by finding the modular multiplicative inverse \( d \) of \( b \) modulo \( m \) using the extended Euclidean algorithm. That is:

\[
c \equiv b^e \equiv d | e| \pmod{m}
\]
where \( e < 0 \) and \( b \cdot d \equiv 1 \pmod{m} \)

Modular exponentiation problems similar to the one described above are considered easy to do, even when the numbers involved are enormous. On the other hand, computing the discrete logarithm - that is, the task of finding the exponent \( e \) if given \( b \), \( c \), and \( m \) - is believed to be difficult. This one way function behavior makes modular exponentiation a candidate for use in cryptographic algorithms.

**Straightforward method**

The most straightforward method of calculating a modular exponent is to calculate \( b^e \) directly, then to take this number modulo \( m \). Consider trying to compute \( c \), given \( b = 4 \), \( e = 13 \), and \( m = 497 \):

\[
c \equiv 4^{13} \pmod{497}
\]

One could use a calculator to compute \( 4^{13} \); this comes out to 67,108,864. Taking this value modulo 497, the answer \( c \) is determined to be 445.

Note that \( b \) is only one digit in length and that \( e \) is only two digits in length, but the value \( b^e \) is 8 digits in length.

In strong cryptography, \( b \) is often at least 256 binary digits (77 decimal digits). Consider \( b = 5 \times 10^{76} \) and \( e = 17 \), both of which are perfectly reasonable values. In this example, \( b \) is 77 digits in length and \( e \) is 2 digits in length, but the value \( b^e \) is 1,304 decimal digits in length. Such calculations are possible on modern computers, but the sheer magnitude of such numbers causes the speed of calculations to slow considerably. As \( b \) and \( e \) increase even further to provide better security, the value \( b^e \) becomes unwieldy.

The time required to perform the exponentiation depends on the operating environment and the processor. The method described above requires \( O(e) \) multiplications to complete.

**Memory-efficient method**

A second method to compute modular exponentiation requires more operations than the first method. Because the required memory is substantially less, however, operations take less time than before. The end result is that the algorithm is faster.

This algorithm makes use of the fact that, given two integers \( a \) and \( b \), the following two equations are equivalent:

\[
c \equiv (a \cdot b) \pmod{m}
\]
\[
c \equiv (a \cdot (b \pmod{m})) \pmod{m}
\]

The algorithm is as follows:

1. Set \( c = 1, e' = 0 \).
2. Increase \( e' \) by 1.
3. Set $c \equiv (b \cdot c) \pmod{m}$.
4. If $e' < e$, goto step 2. Else, $c$ contains the correct solution to $c \equiv b^e \pmod{m}$.

Note that in every pass through step 3, the equation $c \equiv b^{e'} \pmod{m}$ holds true. When step 3 has been executed $e$ times, then, $c$ contains the answer that was sought. In summary, this algorithm basically counts up $e'$ by ones until $e'$ reaches $e$, doing a multiply by $b$ and the modulo operation each time it adds one (to ensure the results stay small).

The example $b = 4$, $e = 13$, and $m = 497$ is presented again. The algorithm passes through step 3 thirteen times:

- $e' = 1. c = (1 * 4) \ mod 497 = 4\ mod 497 = 4.$
- $e' = 2. c = (4 * 4) \ mod 497 = 16\ mod 497 = 16.$
- $e' = 3. c = (16 * 4) \ mod 497 = 64\ mod 497 = 64.$
- $e' = 4. c = (64 * 4) \ mod 497 = 256\ mod 497 = 256.$
- $e' = 5. c = (256 * 4) \ mod 497 = 1024\ mod 497 = 30.$
- $e' = 6. c = (30 * 4) \ mod 497 = 120\ mod 497 = 120.$
- $e' = 7. c = (120 * 4) \ mod 497 = 480\ mod 497 = 480.$
- $e' = 8. c = (480 * 4) \ mod 497 = 1920\ mod 497 = 429.$
- $e' = 9. c = (429 * 4) \ mod 497 = 1716\ mod 497 = 225.$
- $e' = 10. c = (225 * 4) \ mod 497 = 900\ mod 497 = 403.$
- $e' = 11. c = (403 * 4) \ mod 497 = 1612\ mod 497 = 121.$
- $e' = 12. c = (121 * 4) \ mod 497 = 484\ mod 497 = 484.$
- $e' = 13. c = (484 * 4) \ mod 497 = 1936\ mod 497 = 445.$

The final answer for $c$ is therefore 445, as in the first method.

Like the first method, this requires $O(e)$ multiplications to complete. However, since the numbers used in these calculations are much smaller than the numbers used in the first algorithm's calculations, the computation time decreases by a factor of at least $O(e)$ in this method.

In pseudocode, this method can be performed the following way:

```plaintext
function modular_pow(base, exponent, modulus)
    c := 1
    for e_prime = 1 to exponent
        c := (c * base) mod modulus
    return c
```

**Right-to-left binary method**

A third method drastically reduces the number of operations to perform modular exponentiation, while keeping the same memory footprint as in the previous method. It is a combination of the previous method and a more general principle called exponentiation by squaring (also known as binary exponentiation).

First, it is required that the exponent $e$ be converted to binary notation. That is, $e$ can be written as:

$$e = \sum_{i=0}^{n-1} a_i 2^i$$

In such notation, the length of $e$ is $n$ bits. $a_i$ can take the value 0 or 1 for any $i$ such that $0 \leq i < n - 1$. By definition, $a_{n-1} = 1$.

The value $b^e$ can then be written as:

$$b^e = b^{\sum_{i=0}^{n-1} a_i 2^i} = \prod_{i=0}^{n-1} (b^{2^i})^{a_i}$$

The solution $c$ is therefore:
Modular exponentiation

\[ c = \prod_{i=0}^{n-1} \left( b^{2^i} \right)^{a_i} \pmod{m} \]

The following is an example in pseudocode based on Applied Cryptography by Bruce Schneier.\(^1\) The inputs \( \text{base} \), \( \text{exponent} \), and \( \text{modulus} \) correspond to \( b \), \( e \), and \( m \) in the equations given above.

```plaintext
function modular_pow(base, exponent, modulus)
    result := 1
    while exponent > 0
        if (exponent mod 2 == 1):
            result := (result * base) mod modulus
        exponent := exponent >> 1
        base = (base * base) mod modulus
    return result
```

Note that upon entering the loop for the first time, the code variable \( \text{base} \) is equivalent to \( b \). However, the repeated squaring in the third line of code ensures that at the completion of every loop, the variable \( \text{base} \) is equivalent to \( b^{2^i} \pmod{m} \), where \( i \) is the number of times the loop has been iterated. (This makes \( i \) the next working bit of the binary exponent \( \text{exponent} \), where the least-significant bit is \( \text{exponent}_0 \).)

The first line of code simply carries out the multiplication in \( \prod_{i=0}^{n-1} \left( b^{2^i} \right)^{a_i} \pmod{m} \). If \( a_i \) is zero, no code executes since this effectively multiplies the running total by one. If \( a_i \) instead is one, the variable \( \text{base} \) (containing the value of the original base) is simply multiplied in.

Example: \( \text{base} = 4 \), \( \text{exponent} = 13 \), and \( \text{modulus} = 497 \). Note that \( \text{exponent} \) is 1101 in binary notation. Because \( \text{exponent} \) is four binary digits in length, the loop executes only four times:

- Upon entering the loop for the first time, variables \( \text{base} = 4 \), \( \text{exponent} = 1101 \) (binary), and \( \text{result} = 1 \). Because the right-most bit of \( \text{exponent} \) is 1, \( \text{result} \) is changed to be \( (1 * 4) \pmod{497} \), or 4. \( \text{exponent} \) is right-shifted to become 110 (binary), and \( \text{base} \) is squared to be \( (4 * 4) \pmod{497} \), or 16.
- The second time through the loop, the right-most bit of \( \text{exponent} \) is 0, causing \( \text{result} \) to retain its present value of 4. \( \text{exponent} \) is right-shifted to become 11 (binary), and \( \text{base} \) is squared to be \( (16 * 16) \pmod{497} \), or 256.
- The third time through the loop, the right-most bit of \( e \) is 1. \( \text{result} \) is changed to be \( (4 * 256) \pmod{497} \), or 30. \( \text{exponent} \) is right-shifted to become 1, and \( \text{base} \) is squared to be \( (256 * 256) \pmod{497} \), or 429.
- The fourth time through the loop, the right-most bit of \( \text{exponent} \) is 1. \( \text{result} \) is changed to be \( (30 * 429) \pmod{497} \), or 445. \( \text{exponent} \) is right-shifted to become 0, and \( \text{base} \) is squared to be \( (429 * 429) \pmod{497} \), or 151.

The loop then terminates since \( \text{exponent} \) is zero, and the result 445 is returned. This agrees with the previous two algorithms.

The running time of this algorithm is \( O(\log \text{exponent}) \). When working with large values of \( \text{exponent} \), this offers a substantial speed benefit over both of the previous two algorithms.
Reversible and quantum modular exponentiation

In quantum computing, modular exponentiation appears as the bottleneck of Shor's algorithm, where it must be computed by a circuit consisting of reversible gates, which can be further broken down into quantum gates appropriate for a specific physical device. Furthermore, in Shor's algorithm it is possible to know the base and the modulus of exponentiation at every call, which enables various circuit optimizations.\(^2\)

In programming languages

Because modular exponentiation is an important operation in computer science, and there are efficient algorithms (see above) that are much faster than simply exponentiating and then taking the remainder, many programming languages and arbitrary-precision integer libraries have a dedicated function to perform modular exponentiation:

- Python's built-in `pow()` (exponentiation) function \(^3\) takes an optional third argument which is the number to modulo by
- Java's `java.math.BigInteger` class has a `modPow()` \(^4\) method to perform modular exponentiation
- Perl's `Math::BigInt` module has a `bmodpow()` method \(^5\) to perform modular exponentiation
- Go's `big.Int` type contains an `Exp()` (exponentiation) method \(^6\) whose third parameter, if non-nil, is the number to modulo by
- PHP's BC Math library has a `bcpowmod()` function \(^7\) to perform modular exponentiation
- The GNU Multiple Precision Arithmetic Library (GMP) library contains a `mpz_powm()` function \(^8\) to perform modular exponentiation
- Custom Function `@PowerMod()` \(^9\) for FileMaker Pro (with 1024-bit RSA encryption example)

References

[5] http://perldoc.perl.org/Math/BigInt.html#bmodpow%28%29


External links

- Fast Modular Exponentiation Java Applet (http://www.math.umn.edu/~garrett/crypto/a01/FastPow.html) - University of Minnesota Math Department
RSA (algorithm)

<table>
<thead>
<tr>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designers</td>
</tr>
<tr>
<td>First published</td>
</tr>
<tr>
<td>Certification</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cipher detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key sizes</td>
</tr>
<tr>
<td>Rounds</td>
</tr>
</tbody>
</table>

A 768 bit key has been broken

RSA is an algorithm for public-key cryptography that is based on the presumed difficulty of factoring large integers, the factoring problem. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described the algorithm in 1977. Clifford Cocks, an English mathematician, had developed an equivalent system in 1973, but it wasn't declassified until 1997.

A user of RSA creates and then publishes the product of two large prime numbers, along with an auxiliary value, as their public key. The prime factors must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the prime factors can feasibly decode the message. Whether breaking RSA encryption is as hard as factoring is an open question known as the RSA problem.

History

The RSA algorithm was publicly described in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman at MIT; the letters RSA are the initials of their surnames, listed in the same order as on the paper.[1]

MIT was granted U.S. Patent 4,405,829 [2] for a "Cryptographic communications system and method" that used the algorithm in 1983. The patent would have expired on September 21, 2000 (the term of patent was 17 years at the time), but the algorithm was released to the public domain by RSA Security on September 6, 2000, two weeks earlier.[3] Since a paper describing the algorithm had been published in August 1977, prior to the December 1977 filing date of the patent application, regulations in much of the rest of the world precluded patents elsewhere and only the US patent was granted. Had Cocks' work been publicly known, a patent in the US might not have been possible, either.

From the DWPI's abstract of the patent,
The system includes a communications channel coupled to at least one terminal having an encoding device and to at least one terminal having a decoding device. A message-to-be-transferred is enciphered to ciphertext at the encoding terminal by encoding the message as a number M in a predetermined set. That number is then raised to a first predetermined power (associated with the intended receiver) and finally computed. The remainder or residue, C, is... computed when the exponentiated number is divided by the product of two predetermined prime numbers (associated with the intended receiver).

Clifford Cocks, an English mathematician working for the UK intelligence agency GCHQ, described an equivalent system in an internal document in 1973 but, given the relatively expensive computers needed to implement it at the time, it was mostly considered a curiosity and, as far as is publicly known, was never deployed. His discovery, however, was not revealed until 1998 due to its top-secret classification, and Rivest, Shamir, and Adleman devised RSA independently of Cocks' work.

**Operation**

The RSA algorithm involves three steps: key generation, encryption and decryption.

**Key generation**

RSA involves a **public key** and a **private key**. The public key can be known by everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key. The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers p and q.
   - For security purposes, the integers p and q should be chosen at random, and should be of similar bit-length. Prime integers can be efficiently found using a primality test.
2. Compute \( n = pq \).
   - n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
3. Compute \( \phi(n) = \phi(p)\phi(q) = (p-1)(q-1) \), where \( \phi \) is Euler's totient function.
4. Choose an integer e such that 1 < e < \( \phi(n) \) and \( \gcd(e, \phi(n)) = 1 \); i.e., e and \( \phi(n) \) are coprime.
   - e is released as the public key exponent.
   - e having a short bit-length and small Hamming weight results in more efficient encryption — most commonly \( 2^{16} + 1 = 65,537 \). However, much smaller values of e (such as 3) have been shown to be less secure in some settings.
5. Determine d as \( d^{-1} \equiv e \pmod{\phi(n)} \), i.e., d is the multiplicative inverse of e (modulo \( \phi(n) \)).
   - This is more clearly stated as solve for d given \( dë = 1 \pmod{\phi(n)} \)
   - This is often computed using the extended Euclidean algorithm.
   - d is kept as the private key exponent.

The **public key** consists of the modulus n and the public (or encryption) exponent e. The **private key** consists of the modulus n and the private (or decryption) exponent d, which must be kept secret. p, q, and \( \phi(n) \) must also be kept secret because they can be used to calculate d.

- An alternative, used by PKCS#1, is to choose d matching \( de \equiv 1 \pmod{\lambda} \) with \( \lambda = \text{lcm}(p-1, q-1) \), where lcm is the least common multiple. Using \( \lambda \) instead of \( \phi(n) \) allows more choices for d. \( \lambda \) can also be defined using the Carmichael function, \( \lambda(n) \).
- The ANSI X9.31 standard prescribes, IEEE 1363 describes, and PKCS#1 allows, that p and q match additional requirements: being strong primes, and being different enough that Fermat factorization fails.
Encryption

Alice transmits her public key \((n, e)\) to Bob and keeps the private key secret. Bob then wishes to send message \(M\) to Alice.

He first turns \(M\) into an integer \(m\), such that \(0 \leq m < n\) by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext \(c\) corresponding to

\[ c \equiv m^e \pmod{n}. \]

This can be done quickly using the method of exponentiation by squaring. Bob then transmits \(c\) to Alice.

Decryption

Alice can recover \(m\) from \(c\) by using her private key exponent \(d\) via computing

\[ m \equiv c^d \pmod{n}. \]

Given \(m\), she can recover the original message \(M\) by reversing the padding scheme.

(In practice, there are more efficient methods of calculating \(c^d\) using the precomputed values below.)

Using the Chinese remainder algorithm

For efficiency many popular crypto libraries (like OpenSSL, Java and .NET) use the following optimization for decryption and signing based on the Chinese remainder theorem. The following values are precomputed and stored as part of the private key:

- \(p\) and \(q\): the primes from the key generation,
- \(d_P = d \pmod{p-1}\),
- \(d_Q = d \pmod{q-1}\),
- \(q_{\text{inv}} = q^{-1} \pmod{p}\).

These values allow the recipient to compute the exponentiation \(m = c^d \pmod{pq}\) more efficiently as follows:

\[ m_1 = c^{d_P} \pmod{p}, \]
\[ m_2 = c^{d_Q} \pmod{q}, \]
\[ h = q_{\text{inv}}(m_1 - m_2) \pmod{p} \text{ (if } m_1 < m_2 \text{ then some libraries compute } h \text{ as } q_{\text{inv}}(m_1 + p - m_2) \pmod{p}), \]
\[ m = m_2 + hq. \]

This is more efficient than computing \(m = c^d \pmod{pq}\) even though two modular exponentiations have to be computed. The reason is that these two modular exponentiations both use a smaller exponent and a smaller modulus.

A working example

Here is an example of RSA encryption and decryption. The parameters used here are artificially small, but one can also use OpenSSL to generate and examine a real keypair.

1. Choose two distinct prime numbers, such as
   \(p = 61\) and \(q = 53\).
2. Compute \(n = pq\) giving
   \(n = 61 \times 53 = 3233\).
3. Compute the totient of the product as \(\varphi(n) = (p - 1)(q - 1)\) giving
   \(\varphi(3233) = (61 - 1)(53 - 1) = 3120\).
4. Choose any number \(1 < e < 3120\) that is coprime to 3120. Choosing a prime number for \(e\) leaves us only to check that \(e\) is not a divisor of 3120.
   Let \(e = 17\).
5. Compute \( d \), the modular multiplicative inverse of \( e \pmod{\varphi(n)} \) yielding
\[
d = 2753.
\]
The public key is \((n = 3233, e = 17)\). For a padded plaintext message \( m \), the encryption function is
\[
c(m) = m^{17} \pmod{3233}.
\]
The private key is \((n = 3233, d = 2753)\). For an encrypted ciphertext \( c \), the decryption function is \( c^{2753} \pmod{3233} \).
\[
m(c) = c^{2753} \pmod{3233}.
\]
For instance, in order to encrypt \( m = 65 \), we calculate
\[
c \equiv 65^{17} \pmod{3233} \equiv 2790.
\]
To decrypt \( c = 2790 \), we calculate
\[
m \equiv 2790^{2753} \pmod{3233} \equiv 65.
\]
Both of these calculations can be computed efficiently using the square-and-multiply algorithm for modular exponentiation. In real-life situations the primes selected would be much larger; in our example it would be trivial to factor \( n, 3233 \) (obtained from the freely available public key) back to the primes \( p \) and \( q \). Given \( e \), also from the public key, we could then compute \( d \) and so acquire the private key.

Practical implementations use the Chinese remainder theorem to speed up the calculation using modulus of factors \( \pmod{pq} \) using \( \pmod{p} \) and \( \pmod{q} \).

The values \( d_p, d_q, \text{ and } q_{\text{inv}} \), which are part of the private key are computed as follows:

- \( d_p = d \pmod{(p - 1)} = 2753 \pmod{(61 - 1)} = 53 \)
- \( d_q = d \pmod{(q - 1)} = 2753 \pmod{(53 - 1)} = 49 \)
- \( q_{\text{inv}} = q^{-1} \pmod{p} = 53^{-1} \pmod{61} = 38 \) (Hence:
\[
q_{\text{inv}} \times q \pmod{p} = 38 \times 53 \pmod{61} = 1
\]
Here is how \( d_p, d_q, \text{ and } q_{\text{inv}} \) are used for efficient decryption. (Encryption is efficient by choice of public exponent \( e \))

- \( m_1 = c^{d_p} \pmod{p} = 2790^{53} \pmod{61} = 4 \)
- \( m_2 = c^{d_q} \pmod{q} = 2790^{49} \pmod{53} = 12 \)
- \( h = (q_{\text{inv}} \times (m_1 - m_2)) \pmod{p} = (38 \times -8) \pmod{61} = 1 \)
- \( m = m_2 + h \times q = 12 + 1 \times 53 = 65 \) (same as above but computed more efficiently)

**Attacks against plain RSA**

There are a number of attacks against plain RSA as described below.

- When encrypting with low encryption exponents (e.g., \( e = 3 \)) and small values of the \( m \), (i.e., \( m < n^{1/e} \)) the result of \( m^e \) is strictly less than the modulus \( n \). In this case, ciphertexts can be easily decrypted by taking the \( e \)th root of the ciphertext over the integers.

- If the same clear text message is sent to \( e \) or more recipients in an encrypted way, and the receivers share the same exponent \( e \), but different \( p, q \), and therefore \( n \), then it is easy to decrypt the original clear text message via the Chinese remainder theorem. Johan Håstad noticed that this attack is possible even if the cleartexts are not equal, but the attacker knows a linear relation between them.\[4\] This attack was later improved by Don Coppersmith.\[5\]

- Because RSA encryption is a deterministic encryption algorithm (i.e., has no random component) an attacker can successfully launch a chosen plaintext attack against the cryptosystem, by encrypting likely plaintexts under the public key and test if they are equal to the ciphertext. A cryptosystem is called semantically secure if an attacker cannot distinguish two encryptions from each other even if the attacker knows (or has chosen) the corresponding plaintexts. As described above, RSA without padding is not semantically secure.

- RSA has the property that the product of two ciphertexts is equal to the encryption of the product of the respective plaintexts. That is \( m_1^e m_2^e \equiv (m_1 m_2)^e \pmod{n} \). Because of this multiplicative property a chosen-ciphertext attack is
possible. E.g., an attacker, who wants to know the decryption of a ciphertext $c \equiv m^e \pmod{n}$ may ask the holder of the private key to decrypt an unsuspicious-looking ciphertext $c' \equiv cr^e \pmod{n}$ for some value $r$ chosen by the attacker. Because of the multiplicative property $c'$ is the encryption of $mr \pmod{n}$. Hence, if the attacker is successful with the attack, he will learn $mr \pmod{n}$ from which he can derive the message $m$ by multiplying $mr$ with the modular inverse of $r$ modulo $n$.

**Padding schemes**

To avoid these problems, practical RSA implementations typically embed some form of structured, randomized padding into the value $m$ before encrypting it. This padding ensures that $m$ does not fall into the range of insecure plaintexts, and that a given message, once padded, will encrypt to one of a large number of different possible ciphertexts.

Standards such as PKCS#1 have been carefully designed to securely pad messages prior to RSA encryption. Because these schemes pad the plaintext $m$ with some number of additional bits, the size of the un-padded message $M$ must be somewhat smaller. RSA padding schemes must be carefully designed so as to prevent sophisticated attacks which may be facilitated by a predictable message structure. Early versions of the PKCS#1 standard (up to version 1.5) used a construction that appears to make RSA semantically secure. However, at Eurocrypt 2000, Coron et al. showed that for some types of messages, this padding does not provide a high enough level of security. Furthermore, at Crypto 1998, Bleichenbacher showed that this version is vulnerable to a practical adaptive chosen ciphertext attack. Later versions of the standard include Optimal Asymmetric Encryption Padding (OAEP), which prevents these attacks. As such, OAEP should be used in any new application, and PKCS#1 v1.5 padding should be replaced wherever possible. The PKCS#1 standard also incorporates processing schemes designed to provide additional security for RSA signatures (e.g., the Probabilistic Signature Scheme for RSA/RSA-PSS).

**Signing messages**

Suppose Alice uses Bob's public key to send him an encrypted message. In the message, she can claim to be Alice but Bob has no way of verifying that the message was actually from Alice since anyone can use Bob's public key to send him encrypted messages. In order to verify the origin of a message, RSA can also be used to sign a message.

Suppose Alice wishes to send a signed message to Bob. She can use her own private key to do so. She produces a hash value of the message, raises it to the power of $d$ (modulo $n$) (as she does when decrypting a message), and attaches it as a "signature" to the message. When Bob receives the signed message, he uses the same hash algorithm in conjunction with Alice's public key. He raises the signature to the power of $e$ (modulo $n$) (as he does when encrypting a message), and compares the resulting hash value with the message's actual hash value. If the two agree, he knows that the author of the message was in possession of Alice's private key, and that the message has not been tampered with since.

Secure padding schemes such as RSA-PSS are as essential for the security of message signing as they are for message encryption. Two US patents on PSS were granted (USPTO 6266771 and USPTO 70360140); however, these patents expired on 24 July 2009 and 25 April 2010, respectively. Use of PSS no longer seems to be encumbered by patents. Note that using different RSA key-pairs for encryption and signing is potentially more secure.\[6\][7]
Security and practical considerations

Integer factorization and RSA problem

The security of the RSA cryptosystem is based on two mathematical problems: the problem of factoring large numbers and the RSA problem. Full decryption of an RSA ciphertext is thought to be infeasible on the assumption that both of these problems are hard, i.e., no efficient algorithm exists for solving them. Providing security against partial decryption may require the addition of a secure padding scheme.\[citation needed\]

The RSA problem is defined as the task of taking \( e \)th roots modulo a composite \( n \): recovering a value \( m \) such that \( c \equiv m^e \mod n \), where \((n, e)\) is an RSA public key and \( c \) is an RSA ciphertext. Currently the most promising approach to solving the RSA problem is to factor the modulus \( n \). With the ability to recover prime factors, an attacker can compute the secret exponent \( d \) from a public key \((n, e)\), then decrypt \( c \) using the standard procedure. To accomplish this, an attacker factors \( n \) into \( p \) and \( q \), and computes \((p − 1)(q − 1)\) which allows the determination of \( d \) from \( e \). No polynomial-time method for factoring large integers on a classical computer has yet been found, but it has not been proven that none exists. See integer factorization for a discussion of this problem. Rivest, Shamir and Adleman note that Miller has shown that – assuming the Extended Riemann Hypothesis – finding \( d \) from \( n \) and \( e \) is as hard as factoring \( n \) into \( p \) and \( q \) (up to a polynomial time difference).\[8\] However, Rivest, Shamir and Adleman note (in section IX / D of their paper) that they have not found a proof that inverting RSA is equally hard as factoring. As of 2010\[9\], the largest (known) number factored by a general-purpose factoring algorithm was 768 bits long (see RSA-768), using a state-of-the-art distributed implementation. RSA keys are typically 1024 to 2048 bits long. Some experts believe that 1024-bit keys may become breakable in the near future (though this is disputed); few see any way that 4096-bit keys could be broken in the foreseeable future. Therefore, it is generally presumed that RSA is secure if \( n \) is sufficiently large. If \( n \) is 300 bits or shorter, it can be factored in a few hours on a personal computer, using software already freely available. Keys of 512 bits have been shown to be practically breakable in 1999 when RSA-155 was factored by using several hundred computers and are now factored in a few weeks using common hardware.\[10\] Exploits using 512-bit code-signing certificates that may have been factored were reported in 2011.\[11\] A theoretical hardware device named TWIRL and described by Shamir and Tromer in 2003 called into question the security of 1024-bit keys. It is currently recommended that \( n \) be at least 2048 bits long.\[12\] In 1994, Peter Shor showed that a quantum computer (if one could ever be practically created for the purpose) would be able to factor in polynomial time, breaking RSA; see Shor's algorithm.

Faulty key generation

Finding the large primes \( p \) and \( q \) is usually done by testing random numbers of the right size with probabilistic primality tests which quickly eliminate virtually all non-primes.

Numbers \( p \) and \( q \) should not be 'too close', lest the Fermat factorization for \( n \) be successful, if \( p − q \), for instance is less than \( 2n^{1/4} \) (which for even small 1024-bit values of \( n \) is \( 3 \times 10^{77} \)) solving for \( p \) and \( q \) is trivial. Furthermore, if either \( p − 1 \) or \( q − 1 \) has only small prime factors, \( n \) can be factored quickly by Pollard's \( p − 1 \) algorithm, and these values of \( p \) or \( q \) should therefore be discarded as well.

It is important that the private key \( d \) be large enough. Michael J. Wiener showed that if \( p \) is between \( q \) and \( 2q \) (which is quite typical) and \( d < n^{1/3}/3 \), then \( d \) can be computed efficiently from \( n \) and \( e \).

There is no known attack against small public exponents such as \( e = 3 \), provided that proper padding is used. However, when no padding is used, or when the padding is improperly implemented, small public exponents have a greater risk of leading to an attack, such as the unpadded plaintext vulnerability listed above. 65537 is a commonly used value for \( e \). This value can be regarded as a compromise between avoiding potential small exponent attacks and still allowing efficient encryptions (or signature verification). The NIST Special Publication on Computer Security (SP 800-78 Rev 1 of August 2007) does not allow public exponents \( e \) smaller than 65537, but does not state a reason
for this restriction.

**Importance of strong random number generation**

A cryptographically strong random number generator, which has been properly seeded with adequate entropy, must be used to generate the primes $p$ and $q$. An analysis comparing millions of public keys gathered from the Internet was carried out in early 2012 by Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung and Christophe Wachter. They were able to factor 0.2% of the keys using only Euclid's algorithm.\[13\][14]

They exploited a weakness unique to cryptosystems based on integer factorization. If $n = pq$ is one public key and $n' = p'q'$ is another, then if by chance $p = p'$, then a simple computation of $\text{gcd}(n,n') = p$ factors both $n$ and $n'$, totally compromising both keys. Lenstra et al. note that this problem can be minimized by using a strong random seed of bit-length twice the intended security level, or by employing a deterministic function to choose $q$ given $p$, instead of choosing $p$ and $q$ independently.

Nadia Heninger was part of a group that did a similar experiment. They used an idea of Daniel J. Bernstein to compute the GCD of each RSA key $n$ against the product of all the other keys $n'$ they had found (a 729 million digit number), instead of computing each $\text{gcd}(n,n')$ separately, thereby achieving a very significant speedup since after one large division the GCD problem is of normal size.

Heninger says in her blog that the bad keys occurred almost entirely in embedded applications, including "firewalls, routers, VPN devices, remote server administration devices, printers, projectors, and VOIP phones" from over 30 manufactures. Heninger explains that the one-shared-prime problem uncovered by the two groups results from situations where the pseudorandom number generator is poorly seeded initially and then reseeded between the generation of the first and second primes. Using seeds of sufficiently high entropy obtained from key stroke timings or electronic diode noise or atmospheric noise from a radio receiver tuned between stations should solve the problem.\[15\]

Strong random number generation is important throughout every phase of public key cryptography. For instance, if a weak generator is used for the symmetric keys that are being distributed by RSA, then an eavesdropper could bypass the RSA and guess the symmetric keys directly.

**Timing attacks**

Kocher described a new attack on RSA in 1995: if the attacker Eve knows Alice's hardware in sufficient detail and is able to measure the decryption times for several known ciphertexts, she can deduce the decryption key $d$ quickly. This attack can also be applied against the RSA signature scheme. In 2003, Boneh and Brumley demonstrated a more practical attack capable of recovering RSA factorizations over a network connection (e.g., from a Secure Socket Layer (SSL)-enabled webserver)\[16\] This attack takes advantage of information leaked by the Chinese remainder theorem optimization used by many RSA implementations.

One way to thwart these attacks is to ensure that the decryption operation takes a constant amount of time for every ciphertext. However, this approach can significantly reduce performance. Instead, most RSA implementations use an alternate technique known as cryptographic blinding. RSA blinding makes use of the multiplicative property of RSA. Instead of computing $c^d \pmod{n}$, Alice first chooses a secret random value $r$ and computes $(r^e c)^d \pmod{n}$. The result of this computation after applying Euler's Theorem is $r c^d \pmod{n}$ and so the effect of $r$ can be removed by multiplying by its inverse. A new value of $r$ is chosen for each ciphertext. With blinding applied, the decryption time is no longer correlated to the value of the input ciphertext and so the timing attack fails.
Adaptive chosen ciphertext attacks

In 1998, Daniel Bleichenbacher described the first practical adaptive chosen ciphertext attack, against RSA-encrypted messages using the PKCS #1 v1 padding scheme (a padding scheme randomizes and adds structure to an RSA-encrypted message, so it is possible to determine whether a decrypted message is valid). Due to flaws with the PKCS #1 scheme, Bleichenbacher was able to mount a practical attack against RSA implementations of the Secure Socket Layer protocol, and to recover session keys. As a result of this work, cryptographers now recommend the use of provably secure padding schemes such as Optimal Asymmetric Encryption Padding, and RSA Laboratories has released new versions of PKCS #1 that are not vulnerable to these attacks.

Side-channel analysis attacks

A side-channel attack using branch prediction analysis (BPA) has been described. Many processors use a branch predictor to determine whether a conditional branch in the instruction flow of a program is likely to be taken or not. Often these processors also implement simultaneous multithreading (SMT). Branch prediction analysis attacks use a spy process to discover (statistically) the private key when processed with these processors.

Simple Branch Prediction Analysis (SBPA) claims to improve BPA in a non-statistical way. In their paper, "On the Power of Simple Branch Prediction Analysis", [17] the authors of SBPA (Onur Acicimez and Cetin Kaya Koc) claim to have discovered 508 out of 512 bits of an RSA key in 10 iterations.

A power fault attack on RSA implementations has been described in 2010. [18] The authors recovered the key by varying the CPU power voltage outside limits; this caused multiple power faults on the server.

Proofs of correctness

Proof using Fermat's little theorem

The proof of the correctness of RSA is based on Fermat's little theorem. This theorem states that if \( p \) is prime and \( a \) does not divide an integer \( a \) then

\[
a^{(p-1)} \equiv 1 \pmod{p},
\]

We want to show that \( m^{ed} \equiv m \pmod{pq} \) for every integer \( m \) when \( p \) and \( q \) are distinct prime numbers and \( e \) and \( d \) are positive integers satisfying

\[
ed \equiv 1 \pmod{(p-1)(q-1)}.
\]

We can write

\[
ed - 1 = h(p - 1)(q - 1).
\]

for some nonnegative integer \( h \).

To check two numbers, like \( m^{ed} \) and \( m \), are congruent mod \( pq \) it suffices (and in fact is equivalent) to check they are congruent mod \( p \) and mod \( q \) separately. (This is part of the Chinese remainder theorem, although it is not the significant part of that theorem.) To show \( m^{ed} \equiv m \pmod{p} \), (mod p), we consider two cases: \( m \equiv 0 \pmod{p} \) and \( m \not\equiv 0 \pmod{p} \). In the first case \( m^{ed} \) is a multiple of \( p \), so \( m^{ed} \equiv 0 \pmod{p} \). In the second case

\[
m^{ed} = m^{(ed-1)}m = m^{h(p-1)(q-1)}m = (m^{p-1})^{h(q-1)}m \equiv 1^{h(q-1)}m \equiv m \pmod{p},
\]

where we used Fermat's little theorem to replace \( m^{p-1} \) mod \( p \) with 1.

The verification that \( m^{ed} \equiv m \pmod{q} \) proceeds in a similar way, treating separately the cases \( m \equiv 0 \pmod{q} \) and \( m \not\equiv 0 \pmod{q} \), using Fermat's little theorem for modulus \( q \) in the second case.

This completes the proof that, for any integer \( m \),

\[
(m^e)^d \equiv m \pmod{pq}.
\]
Proof using Euler's theorem

Although the original paper of Rivest, Shamir, and Adleman used Fermat's little theorem to explain why RSA works, it is common to find proofs that rely instead on Euler's theorem.

We want to show that \( m^{ed} \equiv m \pmod{n} \), where \( n = pq \) is a product of two different prime numbers and \( e \) and \( d \) are positive integers satisfying \( ed = 1 \pmod{\varphi(n)} \). Since \( e \) and \( d \) are positive, we can write \( ed = 1 + h\varphi(n) \) for some nonnegative integer \( h \). Assuming that \( m \) is relatively prime to \( n \), we have

\[
m^{ed} = m^{1+ h\varphi(n)} = m( m^{\varphi(n)})^h \equiv m(1)^h \equiv m \pmod{n},
\]

where the second-last congruence follows from the Euler's theorem.

When \( m \) is not relatively prime to \( n \), the argument just given is invalid. This is highly improbable (only a proportion of \( 1/p + 1/q - 1/(pq) \) numbers have this property), but even in this case the desired congruence is still true. Either \( m \equiv 0 \pmod{p} \) or \( m \equiv 0 \pmod{q} \), and these cases can be treated using the previous proof.

Notes

[8] Gary L. Miller, "Riemann's Hypothesis and Tests for Primality" (http://www.cs.cmu.edu/~gml/Publications/Papers/Mi75.pdf)
[12] Has the RSA algorithm been compromised as a result of Bernstein's Paper? (http://www.rsa.com/rsalabs/node.asp?id=2007) What key size should I be using?
[14] "Ron was wrong, Whit is right" http://eprint.iacr.org/2012/064.pdf

References


External links

• PKCS #1: RSA Cryptography Standard (http://www.rsasecurity.com/rsalabs/node.asp?id=2125) (RSA Laboratories website)
The PKCS #1 standard "provides recommendations for the implementation of public-key cryptography based on the RSA algorithm, covering the following aspects: cryptographic primitives; encryption schemes; signature schemes with appendix; ASN.1 syntax for representing keys and for identifying the schemes".

- Explanation of RSA using colored lamps (http://www.youtube.com/watch?v=vgTtHV04xRI)
- Prime Number Hide-And-Seek: How the RSA Cipher Works (http://www.muppetlabs.com/~breadbox/txt/rsa.html)
- Example of an RSA implementation with PKCS#1 padding (GPL source code) (http://polarssl.org/source_code)
- Kocher's article about timing attacks (http://www.cryptography.com/resources/whitepapers/TimingAttacks.pdf)
- A spreadsheet implementing RSA (https://docs.google.com/spreadsheet/ccc?key=0AmFN4Z5iIFsHdHdFMGxXZkZCd2RnQWZBQnZqSU4UVE#gid=0)
License

Creative Commons Attribution-Share Alike 3.0
creativecommons.org/licenses/by-sa/3.0/