Primality Testing and Factoring Algorithms, and their Applications in Cryptography

Arham Rakhecha
arrakhecha@gmail.com

Euler Circle

July 15, 2023

Table of Contents

Introduction

Primality Testing and Factoring algorithms

Carl Friedrich Gauss (1777-1855):

Primality Testing and Factoring algorithms

Carl Friedrich Gauss (1777-1855):

The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers to such an extent that it would be superfluous to discuss the problem at length . . . Further, the dignity of the science itself seems to require solution of a problem so elegant and so celebrated.

Asymmetric Cryptography

Asymmetric cryptography, or Public-Key cryptography, is formed on the basis of two key concepts in mathematics: *Primality testing* and *Factoring Algorithms*.

Primality Testing

Fermat's Little Theorem

Fermat's Little Theorem

Let p be a prime which does not divide the integer a, then:

Fermat's Little Theorem

Fermat's Little Theorem

Let p be a prime which does not divide the integer a, then:

$$a^{p-1} \equiv 1 \pmod{p}.$$

Miller-Rabin Test

Miller-Rabin Algorithm

Miller-Rabin Test

Miller-Rabin Algorithm

1. Choose a random integer $a \in [1, N-1]$.

Miller-Rabin Test

Miller-Rabin Algorithm

1. Choose a random integer $a \in [1, N-1]$.
2. Write $N = 2^s t + 1$, with t odd, and compute $b = a^t \bmod N$. If $b \equiv \pm 1 \bmod N$, return true (a is not a witness, N is possibly prime).

Miller-Rabin Test

Miller-Rabin Algorithm

1. Choose a random integer $a \in [1, N-1]$.
2. Write $N = 2^s t + 1$, with t odd, and compute $b = a^t \bmod N$. If $b \equiv \pm 1 \bmod N$, return true (a is not a witness, N is possibly prime).
3. For i from 1 to $s - 1$: Set $b \leftarrow b^2 \bmod N$. If $b \equiv -1 \bmod N$, return true (a is not a witness, N is possibly be prime).

Miller-Rabin Test

Miller-Rabin Algorithm

1. Choose a random integer $a \in [1, N-1]$.
2. Write $N = 2^s t + 1$, with t odd, and compute $b = a^t \bmod N$. If $b \equiv \pm 1 \bmod N$, return true (a is not a witness, N is possibly prime).
3. For i from 1 to $s-1$: Set $b \leftarrow b^2 \bmod N$. If $b \equiv -1 \bmod N$, return true (a is not a witness, N is possibly be prime).
4. Return false (This is because if a is a witness this would mean that N is definitely not prime).

Solovay-Strassen Test

Solovay-Strassen Theorem

Consider an odd integer, which is composite: n. An integer a exists in $\mathbb{Z}_n^*$ which gives us

Solovay-Strassen Test

Solovay-Strassen Theorem

Consider an odd integer, which is composite: n. An integer a exists in $\mathbb{Z}_n^*$ which gives us

$$(a, n) = 1$$

and

Solovay-Strassen Test

Solovay-Strassen Theorem

Consider an odd integer, which is composite: n. An integer a exists in $\mathbb{Z}_n^*$ which gives us

$$(a, n) = 1$$

and

$$a^{(n-1)/2} \not\equiv \left(\frac{a}{n}\right) \bmod n.$$

Agrawal-Kayal-Saxena

AKS Algorithm

1. If $b > 1)$ and $(n = a^b$ for $a \in \mathrm{N}$, return "composite."

Agrawal-Kayal-Saxena

AKS Algorithm

1. If $\mathrm{b} > 1)$ and $(\mathrm{n} = \mathrm{a}^{\mathrm{b}}$ for $\mathrm{a} \in \mathrm{N}$, return "composite."
2. The smallest r for which $O_r(n) > 4\log^2 n$ is true, must be found.

Agrawal-Kayal-Saxena

AKS Algorithm

1. If $\mathrm{b} > 1)$ and $(\mathrm{n} = \mathrm{a}^{\mathrm{b}}$ for $\mathrm{a} \in \mathrm{N}$, return "composite."
2. The smallest r for which $O_r(n) > 4\log^2 n$ is true, must be found.
3. If it is true that $1 < \mathrm{GCD}(\mathrm{a}, \mathrm{n}) < \mathrm{n}$ for every $\mathrm{a} \leq \mathrm{r}$, return "composite."

Agrawal-Kayal-Saxena

AKS Algorithm

1. If $\mathrm{b} > 1)$ and $(\mathrm{n} = \mathrm{a}^{\mathrm{b}}$ for $\mathrm{a} \in \mathrm{N}$, return "composite."
2. The smallest r for which $O_r(n) > 4\log^2 n$ is true, must be found.
3. If it is true that $1 < \mathrm{GCD}(\mathrm{a}, \mathrm{n}) < \mathrm{n}$ for every $\mathrm{a} \leq \mathrm{r}$, return "composite."
4. If it remains true that $\mathrm{n} \leq \mathrm{r}$, return "prime."

Agrawal-Kayal-Saxena

AKS Algorithm

1. If $\mathrm{b} > 1)$ and $(\mathrm{n} = \mathrm{a}^{\mathrm{b}}$ for $\mathrm{a} \in \mathrm{N}$, return "composite."
2. The smallest r for which $O_r(n) > 4\log^2 n$ is true, must be found.
3. If it is true that $1 < \mathrm{GCD}(\mathrm{a}, \mathrm{n}) < \mathrm{n}$ for every $\mathrm{a} \leq \mathrm{r}$, return "composite."
4. If it remains true that $\mathrm{n} \leq \mathrm{r}$, return "prime."
5. For $\mathrm{a} = 1$ to $\lfloor 2\sqrt{\varphi(r)}\log n \rfloor$ do: if it is true that $((x+a)^n \neq x^n + a \pmod{x^r - 1, n})$, return "composite."

Agrawal-Kayal-Saxena

AKS Algorithm

1. If $\mathrm{b} > 1)$ and $(\mathrm{n} = \mathrm{a}^{\mathrm{b}}$ for $\mathrm{a} \in \mathrm{N}$, return "composite."
2. The smallest r for which $O_r(n) > 4\log^2 n$ is true, must be found.
3. If it is true that $1 < \text{GCD}(\mathrm{a}, \mathrm{n}) < \mathrm{n}$ for every $\mathrm{a} \leq \mathrm{r}$, return "composite."
4. If it remains true that $\mathrm{n} \leq \mathrm{r}$, return "prime."
5. For $\mathrm{a} = 1$ to $\lfloor 2\sqrt{\varphi(r)}\log n\rfloor$ do: if it is true that $((x+a)^n \neq x^n + a \pmod{x^r - 1, n})$, return "composite."
6. Return "prime."

Factoring Algorithms

Fermat Factorization

This factoring algorithm is essentially the following concept: if we can interpret N as $a^2 - b^2$, where both a and b are positive integers, we are able to factor N into $(a+b)(a-b)$.

Fermat Factorization Algorithm

This algorithm is for an input of N to generate the factors of N.

Fermat Factorization

This factoring algorithm is essentially the following concept: if we can interpret N as $a^2 - b^2$, where both a and b are positive integers, we are able to factor N into $(a+b)(a-b)$.

Fermat Factorization Algorithm

This algorithm is for an input of N to generate the factors of N.

1. for a from $\lceil \sqrt{N} \rceil$ to N

Fermat Factorization

This factoring algorithm is essentially the following concept: if we can interpret N as $a^2 - b^2$, where both a and b are positive integers, we are able to factor N into $(a+b)(a-b)$.

Fermat Factorization Algorithm

This algorithm is for an input of N to generate the factors of N.

1. for a from $\lceil \sqrt{N} \rceil$ to N
2. $bsqr = a \times a - N$

Fermat Factorization

This factoring algorithm is essentially the following concept: if we can interpret N as $a^2 - b^2$, where both a and b are positive integers, we are able to factor N into $(a+b)(a-b)$.

Fermat Factorization Algorithm

This algorithm is for an input of N to generate the factors of N.

1. for a from $\lceil \sqrt{N} \rceil$ to N
2. $bsqr = a \times a - N$
3. if isSquare($bsqr$) then $b = \sqrt{bsqr}$ $s = a - b$ $v = a + b$ if $s \neq 1$ and $s \neq N$ then output s, v endif
4. endif
5. endfor

Fermat Factorization

This factoring algorithm is essentially the following concept: if we can interpret N as $a^2 - b^2$, where both a and b are positive integers, we are able to factor N into $(a+b)(a-b)$.

Fermat Factorization Algorithm

This algorithm is for an input of N to generate the factors of N.

1. for a from $\lceil\sqrt{N}\rceil$ to N
2. $bsqr = a \times a - N$
3. if isSquare($bsqr$) then $b = \sqrt{bsqr}$ $s = a - b$ $v = a + b$ if $s \neq 1$ and $s \neq N$ then output s, v endif
4. endif
5. endfor

The “isSquare” function takes a square root, and then rounds the value received to an integer, then squares the outcome of that, and then checks if the result is the number that was first inputted

Trial Division and Generalized Trial Division

Trial Division

This algorithm is for an input of N to generate the factors of N.

1. for x from 2 to $\lfloor\sqrt{N}\rfloor$ if x divides N then output $x, N/s$ endif
2. endfor

Generalized Trial Division

In the generalization, we also take numerous values of x, however in this case we use the floor instead of the ceiling, as in the following:

Trial Division and Generalized Trial Division

Trial Division

This algorithm is for an input of N to generate the factors of N.

1. for x from 2 to $\lfloor \sqrt{N} \rfloor$ if x divides N then output $x, N/s$ endif
2. endfor

Generalized Trial Division

In the generalization, we also take numerous values of x, however in this case we use the floor instead of the ceiling, as in the following:

$$x = \lfloor \sqrt{N} \rfloor, \lfloor \sqrt{N} \rfloor \pm 1, \lfloor \sqrt{N} \rfloor \pm 2, \dots .$$

We then see if the greatest common factor between x and N is a proper factor of N, where $\lfloor \sqrt{N} \rfloor$ is the largest integer which is either equal to or less than $\sqrt{N}$.

Quadratic Sieve

Aims to find two numbers which satisfy the following conditions:

$$a^2 \equiv b^2 \pmod{N}$$

and

$$a \not\equiv \pm b \pmod{N}.$$

Pollard's Rho

Pollard's Rho Algorithm

- Choose some x_0, often given the value of 2, and some equation, generally $f(x) = x^2 + 1$.

Pollard's Rho

Pollard's Rho Algorithm

- Choose some x_0, often given the value of 2, and some equation, generally $f(x) = x^2 + 1$.
- Compute $x_1 = f(x_0) \mod N$, $x_2 = f(x_1) \mod N$, $x_3 = f(x_2) \mod N$ and so forth, with general equation $x_(n+1) = f(x_n) \mod N$.

Pollard's Rho

Pollard's Rho Algorithm

- Choose some x_0, often given the value of 2, and some equation, generally $f(x) = x^2 + 1$.
- Compute $x_1 = f(x_0) \mod N$, $x_2 = f(x_1) \mod N$, $x_3 = f(x_2) \mod N$ and so forth, with general equation $x_(n + 1) = f(x_n) \mod N$.
- For subscripts which are even for x_{2y}, we continue with the following: Calculate greatest common factor of $(x_{2y} - x_y)$ and N Continue until greatest common factor is > 1.

Pollard's Rho

Pollard's Rho Algorithm

- Choose some x_0, often given the value of 2, and some equation, generally $f(x) = x^2 + 1$.
- Compute $x_1 = f(x_0) \mod N$, $x_2 = f(x_1) \mod N$, $x_3 = f(x_2) \mod N$ and so forth, with general equation $x_(n+1) = f(x_n) \mod N$.
- For subscripts which are even for x_{2y}, we continue with the following: Calculate greatest common factor of $(x_{2y} - x_y)$ and N Continue until greatest common factor is > 1.
- The greatest common factor is a factor, completing the algorithm.

Example for Pollard's Rho

We will now look at an example of this being used, for $N = 1234$. We set $x_0 = 2$ and $f(x) = x^2 + 1$

$\mathrm{x}_1 = 5$

Example for Pollard's Rho

We will now look at an example of this being used, for $N = 1234$. We set $x_0 = 2$ and $f(x) = x^2 + 1$

$x_1 = 5$
$x_2 = 26$

Example for Pollard's Rho

We will now look at an example of this being used, for $N = 1234$. We set $x_0 = 2$ and $f(x) = x^2 + 1$

$x_1 = 5$
$x_2 = 26 \ \gcd(26 - 5, 1234) = 1$

Example for Pollard's Rho

We will now look at an example of this being used, for $N = 1234$. We set $x_0 = 2$ and $f(x) = x^2 + 1$

$x_1 = 5$
$x_2 = 26 \; \gcd(26 - 5, 1234) = 1$
$x_3 = 677$

Example for Pollard's Rho

We will now look at an example of this being used, for $N = 1234$. We set $x_0 = 2$ and $f(x) = x^2 + 1$

$x_1 = 5$
$x_2 = 26 \ \gcd(26 - 5, 1234) = 1$
$x_3 = 677$
$x_4 = 516$

Example for Pollard's Rho

We will now look at an example of this being used, for $N = 1234$. We set $x_0 = 2$ and $f(x) = x^2 + 1$

$x_1 = 5$
$x_2 = 26 \ \gcd(26 - 5, 1234) = 1$
$x_3 = 677$
$x_4 = 516 \ \gcd(516 - 26, 1234) = 2$

Example for Pollard's Rho

We will now look at an example of this being used, for $N = 1234$. We set $x_0 = 2$ and $f(x) = x^2 + 1$

$\mathrm{x}_1 = 5$
$x_2 = 26 \text{ gcd}(26 - 5, 1234) = 1$
$x_3 = 677$
$x_4 = 516 \text{ gcd}(516 - 26, 1234) = 2$

This gives us the factor 2.

Applications

Applications in Cryptography

Primality testing and factoring algorithms are two key components of modern cryptography. They are utilised primarily in asymmetric, or public-key, cryptography. The concept involves two mathematically related keys, one which is openly distributed and one which is held confidentially.

Rivest-Shamir-Adleman Cryptography

RSA cryptography is a form of asymmetric encryption which relies on the difficulty of factoring large numbers into their prime factors as described above. To generate a key, two large prime numbers, p and q, are chosen. Primality testing is used to verify that p and q are indeed prime. The product of these numbers N is calculated, whilst keeping the values of p and q private.

Thank you!