Tema 10 : Algoritmos de Factorización

Autores :

- Dylan Ramsés Cabrera Morales
- Leonardo Peláez Ascención

Trabajo Final de Criptografía

Facultad de Matemática y Computación

Universidad de la Habana

Índice

1. Introducción

La factorización de enteros, el proceso de descomponer un número compuesto en sus factores primos, es un problema fundamental en la teoría de números con profundas implicaciones en el campo de la criptografía. La seguridad de muchos sistemas criptográficos modernos, especialmente el ampliamente utilizado algoritmo RSA, se basa directamente en la dificultad computacional de factorizar números grandes que son producto de dos primos muy grandes (semiprimos). Mientras que la multiplicación de dos números primos es una operación trivial para las computadoras, el proceso inverso de encontrar esos dos primos a partir de su producto se vuelve exponencialmente más difícil a medida que el tamaño del número aumenta.
Este informe tiene como objetivo proporcionar una revisión exhaustiva de los algoritmos de factorización. Se explorará el estado del arte crítico en este campo, describiendo al menos cuatro algoritmos clave y profundizando en sus fundamentos matemático-criptográficos. Además, se presentará una implementación de software de uno de estos algoritmos, con una explicación detallada sobre cómo se abordaría la factorización de números de gran tamaño, específicamente de al menos 2048 bits, en el contexto de las capacidades computacionales actuales y futuras. La comprensión de estos algoritmos es crucial para evaluar la robustez de los esquemas criptográficos actuales y anticipar los desafíos de seguridad emergentes.

2. Estado del Arte Crítico de los Algoritmos de Factorización

El avance en los algoritmos de factorización ha sido un motor constante en la evolución de la criptografía de clave pública. Históricamente, la dificultad de la factorización ha servido como la base de seguridad para sistemas como RSA, impulsando la investigación tanto en el desarrollo de métodos más eficientes como en la creación de claves de mayor longitud para mantener la seguridad.

2.1. Evolución Histórica

Desde los métodos más antiguos como la división por prueba (Trial Division) y el método de Fermat, el campo ha progresado hacia algoritmos más sofisticados como el Rho de Pollard y el Método de la Curva Elíptica (ECM). Estos métodos representaron mejoras significativas en la eficiencia, particularmente para encontrar factores pequeños o medianos. Sin embargo, para números compuestos por dos factores primos grandes de tamaño similar, como los utilizados en RSA, se requirieron algoritmos aún más potentes.
El desarrollo de la Criba Cuadrática (Quadratic Sieve, QS) y, posteriormente, la Criba General del Campo de Números (General Number Field Sieve, GNFS), marcó un hito crucial. El GNFS es reconocido como el algoritmo clásico más eficiente para factorizar enteros mayores de 10^{100}, ofreciendo una complejidad subexponencial. Su capacidad para encontrar números "suaves"(con factores primos pequeños) en campos numéricos, en lugar de solo en enteros, es la clave de su eficiencia superior.

3. Descripción y Fundamentación Matemático-Criptográfica de Algoritmos Clave

A continuación, se describen cuatro algoritmos de factorización de enteros, detallando su funcionamiento y los principios matemáticos subyacentes.

3.1. Algoritmo de División por Prueba (Trial Division)

3.1.1. Descripción

El algoritmo de división por prueba es el método de factorización más simple y directo. Consiste en intentar dividir el número compuesto n por cada entero desde 2 hasta la raíz cuadrada de n. Si se encuentra un divisor, entonces n es compuesto y ese divisor es un factor. Si no se encuentra ningún divisor en este rango, entonces n es un número primo.

3.1.2. Fundamentación Matemático-Criptográfica

La fundamentación matemática de este algoritmo es elemental: si un número compuesto n tiene un factor p, entonces debe tener otro factor q tal que $n = p \cdot q$. Si p y q son ambos mayores que $\sqrt{n}$, entonces su producto $p \cdot q$ sería mayor que n, lo cual es una contradicción. Por lo tanto, al menos uno de los factores primos de n debe ser menor o igual a $\sqrt{n}$. Esto permite limitar el rango de búsqueda de divisores.
La complejidad temporal de la división por prueba es $O(\sqrt{n})$. Aunque es conceptualmente sencillo,

su eficiencia disminuye drásticamente a medida que n aumenta. Por ejemplo, para un número de 2048 bits, $\sqrt{n}$ sería aproximadamente 2^{1024}, una cantidad astronómicamente grande de pruebas, lo que lo hace inviable para números criptográficamente relevantes.

3.2. Método de Factorización de Fermat

3.2.1. Descripción

El método de factorización de Fermat, ideado por Pierre de Fermat, es un algoritmo de factorización diseñado para números enteros impares. Se basa en la idea de representar un número compuesto N como la diferencia de dos cuadrados perfectos: N = x^2 - y^2. Si se encuentra tal representación, entonces N puede factorizarse como (x - y)(x + y).
El algoritmo procede de la siguiente manera:

1. Se calcula un valor inicial para x, que es el entero más pequeño mayor o igual a la raíz cuadrada de N
2. Se calcula b = x^2 - N
3. Se verifica si b es un cuadrado perfecto
4. Si b no es un cuadrado perfecto, se incrementa x en 1 y se repiten los pasos 2 y 3.
5. Una vez que se encuentra un x tal que x^2 - N es un cuadrado perfecto, los factores de N son (x - y) y (x + y).

3.2.2. Fundamentación Matemático-Criptográfica

La eficiencia del método de Fermat depende críticamente de la cercanía de los dos factores primos de N. Si $N = P \cdot Q$ y P, Q están muy cerca el uno del otro (es decir, $P \approx Q$), el algoritmo converge rápidamente porque x (que es aproximadamente (P+Q)/2) estará muy cerca de $\sqrt{N}$, y x^2 - N se convertirá en un cuadrado perfecto y^2 (donde y = (P-Q)/2) con pocas iteraciones.
La complejidad temporal del método de Fermat es $O(N^{1/4})$ en el peor de los casos para una variante específica , pero en la práctica, su rendimiento es muy variable. Si los factores P y Q están muy separados, el número de iteraciones puede ser muy grande, haciendo que el algoritmo sea ineficiente. Para números RSA, donde los factores primos P y Q son grandes y se eligen para que no estén necesariamente cerca (de hecho, se suelen elegir para que estén separados para evitar ataques como este), el método de Fermat no es práctico. Es más adecuado para números que son producto de dos primos cercanos.

3.3. Algoritmo de Rho de Pollard

3.3.1. Descripción

El algoritmo Rho de Pollard, introducido por John Pollard en 1975, es un método de factorización probabilístico diseñado para encontrar factores primos relativamente pequeños de números compuestos grandes. Se basa en la idea de la paradoja del cumpleaños y la detección de ciclos en una secuencia pseudoaleatoria. El algoritmo funciona generando una secuencia de números x_i utilizando una función de iteración, comúnmente f(x) = (x^2 + c) mod n, donde c es una constante y n es el número a factorizar. Se utilizan dos "punteros", uno que se mueve un paso a la vez (la

"tortuga", x) y otro que se mueve dos pasos a la vez (la "liebre", y). Debido a que la secuencia de valores módulo n es finita, eventualmente x e y colisionarán (es decir, $x \equiv y \pmod{p}$ para algún factor p de n), formando un ciclo. Cuando esto ocurre, el máximo común divisor (GCD) de la diferencia absoluta entre x e y y el número n (es decir, $\gcd(|x - y|, n)$) a menudo revelará un factor no trivial de n

3.3.2. Fundamentación Matemático-Criptográfica

La eficacia del algoritmo Rho de Pollard radica en la propiedad de que, si p es un factor primo de n, la secuencia $x_i \pmod{p}$ también formará un ciclo. La paradoja del cumpleaños sugiere que el número esperado de elementos que se deben generar antes de encontrar una colisión es aproximadamente $\sqrt{(\pi p/2)}$. Por lo tanto, la probabilidad de encontrar un factor p es proporcional a $\sqrt{p}$. Esto significa que el algoritmo es particularmente eficiente cuando el número a factorizar tiene al menos un factor primo pequeño.
La complejidad temporal esperada de Pollard's Rho es $O(\sqrt{p})$, donde p es el factor primo más pequeño de n. Para números RSA, donde los dos factores primos son grandes y de tamaño similar, Pollard's Rho no es eficiente porque p sería del orden de $\sqrt{n}$, llevando a una complejidad de $O(n^{1/4})$. Es una mejora sobre la división por prueba, pero aún insuficiente para números de 2048 bits

3.4. Método de la Curva Elíptica (ECM)

3.4.1. Descripción

El Método de la Curva Elíptica (ECM), propuesto por Hendrik Lenstra en 1987, es un algoritmo de factorización subexponencial que utiliza la estructura de grupo de las curvas elípticas sobre campos finitos. Es un algoritmo de propósito especial, lo que significa que es más adecuado para encontrar factores pequeños o medianos (por ejemplo, de 9 a 30 dígitos) de números muy grandes, no necesariamente para semiprimos grandes como los de RSA. A menudo se utiliza como una fase inicial para eliminar factores pequeños de un número antes de aplicar métodos más costosos para factores grandes.
El algoritmo selecciona aleatoriamente una curva elíptica E y un punto P sobre ella, definidos sobre los enteros módulo n. Luego, calcula múltiplos escalares de P (es decir, kP para un k que es un producto de muchos números pequeños, como un factorial o un producto de potencias de primos pequeños). Las operaciones de adición de puntos en la curva elíptica implican divisiones modulares. Si durante una de estas divisiones se intenta dividir por un número que no es invertible módulo n, pero que tiene un GCD no trivial con n, entonces se ha encontrado un factor de n. Esto ocurre si el orden del grupo de la curva elíptica módulo un factor primo p de n es "suave"(es decir, sus factores primos son pequeños).

3.4.2. Fundamentación Matemático-Criptográfica

La ventaja de ECM sobre algoritmos como Pollard's Rho o p-1 es que, al usar múltiples curvas elípticas, la probabilidad de que el orden del grupo de alguna de esas curvas módulo un factor primo p sea suave es mayor que la probabilidad de que p-1 sea suave. Esto se debe a que hay muchas curvas elípticas posibles para un p dado, y sus órdenes de grupo varían, mientras que p-1 es fijo.
La complejidad temporal esperada de ECM para encontrar un factor p de n es $O(\exp(\sqrt{2}\log p \log p \log p))$.
Esta complejidad depende principalmente del tamaño del factor más pequeño p que se busca. Para

números RSA de 2048 bits, cuyos factores primos son de aproximadamente 1024 bits, ECM no es eficiente, ya que la probabilidad de que el orden del grupo sea lo suficientemente suave para un factor de ese tamaño es muy baja.

3.5. Criba General del Campo de Números (GNFS)

3.5.1. Descripción

La Criba General del Campo de Números (GNFS) es el algoritmo clásico de factorización de enteros más eficiente conocido para números muy grandes, específicamente aquellos mayores de 10^{100}. Es un algoritmo de propósito general, lo que significa que puede factorizar cualquier número compuesto que no sea una potencia de un primo (los cuales son triviales de factorizar tomando raíces)
El principio fundamental de GNFS es una mejora sobre los métodos de criba más simples, como la criba cuadrática. En lugar de buscar números suaves del orden de $\sqrt{n}$, GNFS busca números suaves que son subexponenciales en el tamaño de n. Esto se logra realizando cálculos y factorizaciones en campos numéricos algebraicos, lo que introduce una complejidad considerable en el algoritmo.
El método implica la elección de dos polinomios f(x) y g(x) de grados pequeños, con coeficientes enteros, irreducibles sobre los racionales y que tienen una raíz común m cuando se interpretan módulo n. El objetivo es encontrar pares de enteros (a, b) que hagan que dos expresiones relacionadas con los polinomios, r y s, sean simultáneamente "suaves"(tengan solo factores primos pequeños). Una vez que se han encontrado suficientes de estos pares, se utiliza álgebra lineal (por ejemplo, eliminación Gaussiana o algoritmos más eficientes como Block Lanczos) para encontrar combinaciones de estos pares que resulten en cuadrados en los campos numéricos correspondientes. Finalmente, esto permite encontrar dos números x e y tales que $x^2 \equiv y^2$ (mod n), lo que con alta probabilidad revela un factor de n calculando gcd(n, x - y).

3.5.2. Fundamentación Matemático-Criptográfica

La complejidad heurística de GNFS para factorizar un entero n se expresa como $O(e^{c+o(1)} \cdot \log^{1/3} n \cdot \log^{2/3}(\log n))$
Esta es la complejidad más baja conocida para cualquier algoritmo de factorización clásico. La elección de los polinomios es crítica y puede afectar drásticamente el tiempo de ejecución.
El GNFS es el algoritmo utilizado para los récords de factorización de números RSA grandes, como el RSA-768 y el RSA-250. Sin embargo, incluso con la GNFS, la factorización de un número de 2048 bits sigue siendo computacionalmente inviable con los recursos actuales. Se estima que requeriría un esfuerzo computacional masivo, equivalente a miles de años de CPU en máquinas de alto rendimiento, lo que valida la seguridad actual de RSA-2048 frente a ataques clásicos.

4. Implementación de un Software para Factorizar Números

Como se ha explicado, la Criba General del Campo de Números (GNFS) es el algoritmo clásico más eficiente para números de gran tamaño. Sin embargo, una implementación práctica y completa de GNFS para factorizar números de 2048 bits es extremadamente compleja y va mucho más allá de un software de ejemplo. Requiere un conocimiento matemático profundo, algoritmos altamente

optimizados para álgebra lineal de matrices dispersas, y vastos recursos computacionales, a menudo distribuidos.
De manera similar, el algoritmo de Shor, aunque teóricamente más eficiente en un computador cuántico, requiere hardware cuántico tolerante a fallos a gran escala que aún no está disponible para números de 2048 bits.
Dadas estas limitaciones prácticas, el código proporcionado en *PollardRho.ipynb* es una implementación del algoritmo Rho de Pollard. Este algoritmo, aunque no es el más eficiente para factorizar números RSA de 2048 bits (ya que estos no tienen factores primos pequeños), sirve como un ejemplo ilustrativo de cómo se manejan los números grandes y las operaciones modulares en los algoritmos de factorización. Demuestra el uso de aritmética de precisión arbitraria y exponenciación modular, conceptos fundamentales para cualquier algoritmo de factorización que opere con números de gran tamaño.

5. Conclusiones

El análisis de los algoritmos de factorización demuestra que, en el ámbito de la computación clásica, la Criba General del Campo de Números (GNFS) es el método más potente para números grandes. Este algoritmo ha permitido la factorización de números de hasta 768 bits (RSA-768) en 2009 y 829 bits (RSA-250) en 2020. Estos logros representan los límites actuales de la capacidad de factorización clásica. Sin embargo, el módulo RSA-2048, que es el estándar actual para muchas aplicaciones de seguridad, sigue siendo computacionalmente seguro contra ataques clásicos, lo que justifica su uso continuo en la mayoría de los sistemas criptográficos.
Algoritmos como la División por Prueba y el Rho de Pollard son valiosos para encontrar factores pequeños o medianos, pero su complejidad computacional los hace inadecuados para los semiprimos grandes que constituyen los módulos RSA de 2048 bits. El Método de la Curva Elíptica (ECM) es más eficiente para factores medianos, pero tampoco escala adecuadamente para los factores de 1024 bits que se encuentran en un número RSA-2048. La seguridad de RSA se basa en la premisa de que factorizar el producto de dos primos grandes es un problema "difícil"para las computadoras clásicas, lo que implica que el tiempo y los recursos necesarios para realizar la factorización exceden cualquier límite práctico.

Referencias

[1] Lerch, D. (2007). Criptografía de curva elíptica: Ataque de rho de pollard.

[2] Lucena López, M. J. (2010). Criptografía y seguridad en computadores. Universidad de Jaén.

[3] Corena Bossa, J. C., & Rey Salazar, I. M. (2006). Factorización de números enteros grandes usando General Number Field Sieve en paralelo.

[4] https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_de_enteros

[5] https://es.wikipedia.org/wiki/Algoritmo_rho_de_Pollard

[6] https://es.wikipedia.org/wiki/Criba_especial_del_cuerpo_de_n%C3%BAmeros