UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Una propuesta híbrida para el criptoanálisis RSA

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Noelia Marí Salvador

Tutor: Damián López Rodríguez

Curso 2017/2018

RESUMEN

Dada la importancia actual de los algoritmos de cifrado de clave pública en protocolos criptográficos y por lo tanto de los problemas que se utilizan para proporcionar la seguridad computacional necesaria, la factorización de números enteros es un problema interesante ya que no se conoce algoritmo eficiente para resolverla y tampoco se conoce con certeza la clasificación del problema.

El objetivo de este trabajo consiste en proponer un algoritmo de factorización híbrido para el criptoanálisis RSA. Para este fin, se partirá de los algoritmos de Dixon y de Pollard Rho.

La hipótesis de este estudio de basa en modificar el algoritmo de Dixon, incorporando características del algoritmo de Pollard Rho, para intentar conseguir los valores correctos para la factorización con menor costo temporal.

Las conclusiones de este documento se basarán en la rapidez y la efectividad del nuevo algoritmo comparado con los otros dos utilizados.

Palabras clave: RSA, criptoanálisis, Dixon, Pollard Rho, factorización.

ABSTRACT

Because of the current importance of public-key algorithms in cryptographic protocols and, hence, of the problems used to provide the computational security needed, integer factorization is an interesting problem, because neither an efficient algorithm is known to solve it nor the problem's classification isn't known.

The goal of this work is to propose a new hybrid factorization algorithm for RSA cryptanalysis. To achieve this, we will consider Dixon's and Pollard's Rho algorithms.

The hypothesis of this study is based on modifying Dixon's algorithm, incorporating characteristics of Pollard Rho's algorithm, in order to obtain the the factorization with lower temporal cost.

Conclusions will be based on compared efficiency of the algorithms.

Keywords: RSA, cryptanalysis, Dixon, Pollard Rho, factorization.

RESUM

Donada la importància actual dels algoritmes de xifrat de clau pública en protocols criptogràfics i per tant dels problemes que s'utilitzen per a proporcionar la seguretat computacional necessària, la factorització de nombres sencers és un problema interessant ja que no es coneix algoritme eficient per a resoldre-la i tampoc es coneix amb certesa la classificació del problema.

L'objectiu d'este treball consisteix a proposar un algoritme de factorització híbrid per al criptoanàlisis RSA. Per a este fi, es partirà dels algoritmes de Dixon i de Pollard Rho.

La hipòtesi d'este estudi de basa a modificar l'algoritme de Dixon, incorporant característiques de l'algoritme de Pollard Rho, per a intentar aconseguir els valors correctes per a la factorització amb menor cost temporal.

Les conclusions d'este document es basaran en la rapidesa i l'efectivitat del nou algoritme comparat amb els altres dos utilitzats.

Paraules clau: RSA, criptoanàlisis, Dixon, Pollard Rho, factorització.

TABLA DE CONTENIDOS

TABLA DE ALGORITMOS

INTRODUCCIÓN

LA CRIPTOGRAFÍA

Puede parecer que el concepto de criptografía es moderno y relativamente nuevo, ahora están en pleno auge las criptomonedas como el Bitcoin o el Ether, y no hace mucho del escándalo de la red de vigilancia mundial que Snowden filtró [7]. Sin embargo, la criptografía data de hace más de 4.000 años, y su nombre origina del griego antiguo: *krypto*, «oculto», y *graphos*, «escribir»; lo que quiere decir, escritura oculta.

Desde la Antigua Roma con Julio César y su característico cifrado César, pasando por la Segunda Guerra Mundial y la famosa máquina Enigma y actualmente con las criptomonedas, la criptografía ha jugado, y lo sigue haciendo, un papel importante en la historia de la humanidad y su relevancia ha ido aumentando conforme ha aumentado el volumen de información manejada. Es precisamente debido a esto, que los métodos de encriptación han cambiado; algunos se han ido quedando obsoletos y otros han evolucionado.

Tradicionalmente, los criptógrafos han considerado el diseño de métodos basados en la clave privada, también conocidos como algoritmos de clave simétrica (de hecho fue así hasta bien entrado el siglo XX). Estos métodos consideran la misma clave para el cifrado y el descifrado de los mensajes. Los sistemas de clave simétrica presentaban muchas ventajas, por ejemplo, algunas de ellas: era fácil de usar, la criptografía de clave simétrica es rápida y la vigencia de las claves era mucho mayor comparada con la actualidad. El problema de estos sistemas respecto a la vigencia de las claves aparece cuando el tráfico de comunicaciones es tan elevado que hace inviable la distribución de claves. Supone un peligro para la encriptación que muchas personas deban conocer una misma clave, también, si el tráfico de comunicaciones aumentan y las claves son distintas, supone la dificultad de almacenar tantas claves como actores en la comunicación.

Este sistema resultó útil entonces ya que, tanto el volumen de información que necesitaba cifrarse, como el número de interlocutores que necesitaban el acceso a una comunicación cifrada era muy limitado. Por lo tanto, eso requería la existencia de pocos canales seguros para la distribución de claves.

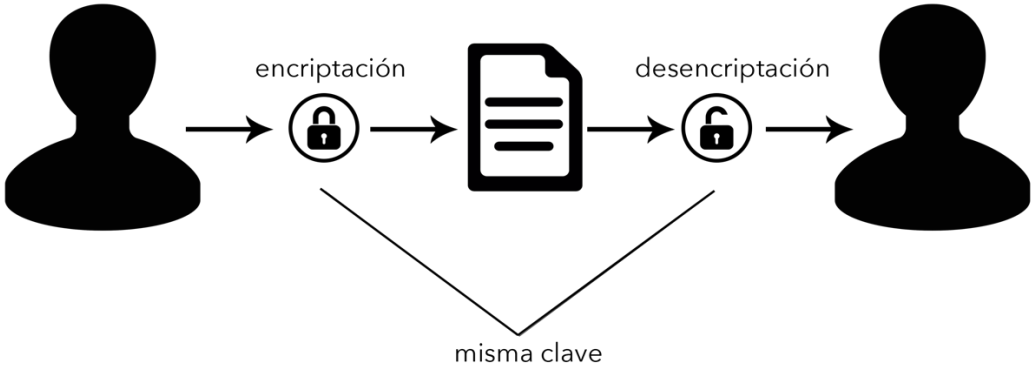


Ilustración 1. Esquema de clave simétrica

Desde el siglo XIX ya se considera que la seguridad no depende del secreto del diseño de los métodos, como Auguste Kerckhoffs, un lingüista y criptógrafo holandés, enunció en el segundo de sus seis principios relativos a las características deseables en los criptosistemas: "La efectividad del sistema no debe depender de que su diseño permanezca en secreto", esto es, que la seguridad de un criptosistema debe recaer exclusivamente en la clave, independientemente del algoritmo utilizado. Debido a esto, la distribución de las claves siempre ha sido el punto más importante del algoritmo de clave privada.

Un ejemplo de cómo los algoritmos han ido evolucionando es el algoritmo 3DES, el cual emplea 3 claves y 3 ejecuciones del algoritmo más común en sistemas de clave privada, el DES. Actualmente, siguen apareciendo nuevos protocolos para intentar paliar el problema de la distribución de claves en los algoritmos de clave simétrica.

En los años 70 del siglo pasado, Whitfield Diffie y Martin Hellman hicieron público un artículo [1] que cambió completamente el funcionamiento de los criptosistemas de entonces, ya que proponían un método de intercambio de claves que no requería un canal seguro. Hoy en día, el algoritmo Diffie-Hellman se sigue utilizando en muchas aplicaciones que implican comunicación entre diferentes partes, como por ejemplo WhatsApp. Además de la citada aplicación de mensajería, Diffie-Hellman también está presente en la red para el anonimato Tor. Esta red emplea el protocolo Diffie-Hellman sobre una conexión TLS previamente establecida, para intercambiar claves de sesión entre el cliente y los nodos de la red.

Debido a este resultado fue posible la creación de los sistemas de clave pública (o clave asimétrica), en los que nos vamos a centrar en este proyecto. En los sistemas de clave pública, la clave está formada por dos componentes distintas: la componente

pública y la componente privada. La efectividad de estos algoritmos depende de problemas matemáticos conocidos como funciones de un solo sentido, que requieren poca potencia de cálculo para llevarse a cabo, pero suponen demasiado coste para calcular a la inversa. Es aquí donde aparece el criptosistema RSA [2], el criptosistema más utilizado hasta el momento y en el que nos vamos a centrar en este estudio.

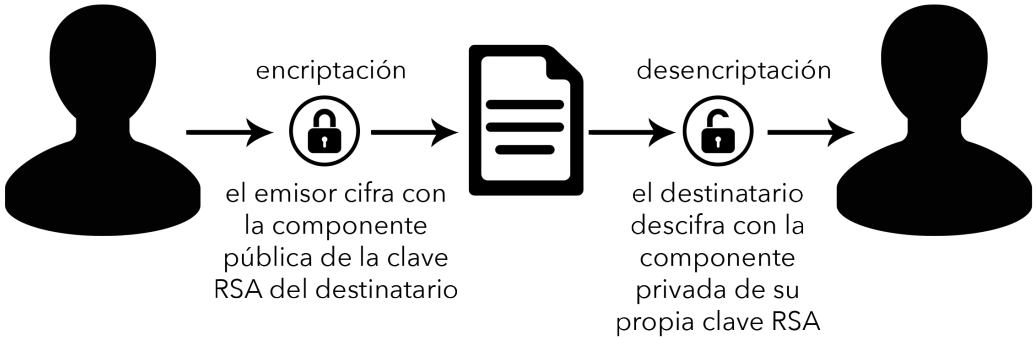


Ilustración 2. Esquema de clave asimétrica

Como hemos comentado antes, y por el principio de Kerckhoffs, la clave es el elemento que marca la seguridad de cualquier algoritmo en criptografía. Por lo tanto, el tamaño de la clave es muy importante. Se habla de seguridad computacional cuando el hecho de descifrar la clave es tan costoso que no merece la pena llevarlo a cabo. Por ejemplo, si se tratase un problema el cual es NP-completo o uno para el que todavía no se dispone de un algoritmo eficiente para él, no importa el hardware utilizado, el coste es inabordable partir de cierto tamaño de problema.

Los problemas habitualmente utilizados para garantizar la seguridad computacional son los dos siguientes:

- El logaritmo discreto: el cálculo del logaritmo discreto no es una tarea sencilla de computar, mientras que la inversa de este, la exponenciación discreta , sí resulta en una tarea simple (por ejemplo: el cálculo de 3^{1359} módulo n, siendo n un número de gran tamaño). En la mayoría de los casos, la complejidad en un caso normal resulta tan difícil como en el peor de los casos.

- La factorización: consiste en descomponer un entero n en sus factores primos. Como caso general, no existe un algoritmo efectivo para resolver el problema. Aunque sin llegar a ser completamente efectivos, existen algoritmos de

factorización que para casos particulares resultan muy eficientes, como se muestra más adelante en este documento.

En ambos casos, el tamaño del problema viene determinado por el numero de bits necesarios para representar en binario el número que se considera en el problema.

Las claves mayormente utilizadas actualmente en criptosistemas de clave asimétrica son de 1024 bits, lo que supone un coste muy elevado de computación. Sin embargo, si se consiguiera un algoritmo extremadamente potente y cuyo coste fuese razonable para la factorización, en principio, no habría ningún problema en aumentar el tamaño de las claves para garantizar la seguridad.

Criptosistema RSA

En 1977, Ron Rivest, Adi Shamir y Leonard Adleman desarrollaron uno de los primeros criptosistemas de clave pública, divulgado en su artículo de 1978 [2]. Este sistema se basa en la propuesta de Whitfield Diffie y Martin E. Hellman. El algoritmo Diffie-Hellman basa sus seguridad en el cálculo del algoritmo discreto. Rivest, Shamir y Adleman, en su artículo, consideran el producto de dos primos como valor modular, lo que les permite obtener dos valores *e* y *d* tales que:

$$x^{ed} \bmod n = x$$

En este punto, se muestra una de las problemáticas anteriormente mencionadas, esta operación, la exponenciación discreta, es fácil de resolver, no como su inversa, el logaritmo discreto. Esto implica que si se quiere averiguar x, se debería resolver el logaritmo discreto de la ecuación, pese al gran costo computacional de la factorización de *n*, resulta más sencillo factorizar *n* que obtener *e* a partir de *d*, o bien *d* a partir de *e*.

La implementación del algoritmo de generación de claves RSA depende de una elección a priori de dos grandes números primos *p* y *q*, que se multiplican para obtener *n*.

$$n = p \cdot q$$

A continuación, se calcula la función φ de Euler:

$$\varphi(n) = (p - 1)(q - 1)$$

La función φ es importante porque proporciona el tamaño del grupo multiplicativo de enteros módulo n, número relativamente primos con respecto a *n*.

Posteriormente se calculan dos parámetros, uno público y uno privado, *e* y *d*, que servirán como exponentes para el cifrado y para el descifrado respectivamente. Para esto, se escoge uno de los parámetros y el otro se calcula satisfaciendo:

$$e \cdot d = 1 + k \cdot \varphi(n)$$

para un cierto entero *k*. Siendo *e* menor a φ(n) y coprimo con este mismo. Este cálculo satisface la congruencia:

$$e \cdot d \equiv 1 \pmod{\varphi(n)}$$

Esto significa:

$$d = e^{-1} \pmod{\varphi(n)}$$

$$e = d^{-1} \pmod{\varphi(n)}$$

De este algoritmo obtenemos el componente público

$$K_{pb} = (n, e)$$

y el componente privado

$$K_{pr} = d$$

La clave resultante se obtiene considerando ambas componentes:

$$<K_{pb}, K_{pr}>$$

Como hemos dicho antes, RSA se basa en funciones de un solo sentido: si bien es fácil calcular *N* multiplicando *p* y *q*, es difícil factorizar el producto. Lo que impide obtener eficientemente la K_{pr} a partir de la K_{pb}.

Los algoritmos de cifrado y de descifrado de RSA son muy sencillos y muy parecidos entre sí.

Algoritmo de cifrado RSA

Entrada:

- K_{pb}^{B}= (n, e): componente pública del destinatario
- *x*: mensaje a cifrar

Salida: *y*: mensaje cifrado

1: $y = x^{e} \pmod{n}$

2: **Devuelve** y

Algoritmo de descifrado RSA

Entrada:

- K_{pr}^{D}= (n, d): componente privada del destinatario

- *y*: mensaje a descifrar

Salida: *x*: mensaje descifrado

1: $x = y^d \pmod n$

2: **Devuelve** x

CORRECCIÓN

Para demostrar que en, en efecto, el proceso de cifrado y de descifrado es correcto se muestra la traza de la corrección [5]. Dados un mensaje *x* y una clave RSA:

Ya que $e \cdot d \equiv 1 \pmod{\varphi(n)}$, existe un entero *k* tal que $e \cdot d = 1 + k \cdot \varphi(n)$. Si mcd(x, p) = 1, entonces por el teorema de Fermat

$$x^{p-1} \equiv 1 \pmod p$$

Elevando ambas partes de esta congruencia a la potencia k(q-1)

$$(x^{p-1})^{k(q-1)} \equiv 1^{k(q-1)} \pmod p$$

$$x^{k(p-1)(q-1)} \equiv 1 \pmod p$$

Y multiplicando ambas partes por *x*

$$x^{1+k(p-1)(q-1)} \equiv x \pmod p$$

Por otro lado, si mcd(x, p) = p, esta última congruencia sigue siendo válida puesto que ambos lados son congruentes con 0 (mod *p*). Por esto, en cualquier caso

$$x^{ed} \equiv x \pmod p$$

Por el mismo argumento

$$x^{ed} \equiv x \pmod q$$

Debido a que p y q son distintos primos

$$x^{ed} \equiv x \pmod n$$

Por lo tanto,

$$y^d \equiv (x^e)^d \equiv x \pmod{n}$$

Estos cálculos se basan en el teorema chino del resto, usado para reducir operaciones con número extremadamente grandes mediante la conversión a congruencias. Pudiendo ser transportados los cálculos de $\mathbf{Z}_n$ a $\mathbf{Z}_p \cdot \mathbf{Z}_q$, donde n = p·q. Pero, en este contexto, permite probar que como

$$x \equiv a \pmod{p}$$

y además:

$$x \equiv a \pmod{q}$$

entonces, podemos concluir, utilizando el teorema chino del resto, que:

$$x \equiv a \pmod{p \cdot q}$$

Ataques a RSA

Al ser un sistema desarrollado en 1977, RSA ha sido objeto de numerosos análisis y ese es uno de sus puntos fuertes: no se ha encontrado ningún ataque que comprometa realmente la seguridad del sistema.

La mayoría de los ataques conocidos hasta la fecha aprovechan las propias características de la implementación del criptosistema. Por lo general, explotan su uso indebido, la mala elección del componente privado *d* o el componente público *e*, la relación entre mensajes encriptados, etc.

A continuación se detallan algunos de los ataques conocidos:

Cifrado cíclico

El cifrado cíclico consiste en cifrar repetidas veces el criptograma interceptado con la clave pública del destinatario hasta volver a obtener el criptograma, cosa que tarde o temprano ocurrirá y que nos revelará el mensaje en claro. Pongamos un ejemplo, siendo n = 52.841 y el criptograma interceptado $c^e = 1.855^7$.

$$c_1 = c^e \pmod n = \boxed{1.855}^7 \pmod{52.841} = 23.797$$
$$c_2 = c_1^e \pmod n = 23.797^7 \pmod{52.841} = 25.334$$
$$c_3 = c_2^e \pmod n = 25.334^7 \pmod{52.841} = 12.508$$
$$c_4 = c_3^e \pmod n = 12.508^7 \pmod{52.841} = 49.343$$
$$c_5 = c_4^e \pmod n = 49.343^7 \pmod{52.841} = 33.708$$
$$\dots$$
$$c_{39} = c_{38}^e \pmod n = 10.494^7 \pmod{52.841} = 46.640$$
$$c_{40} = c_{39}^e \pmod n = 46.640^7 \pmod{52.841} = 17.225$$
$$c_{41} = c_{40}^e \pmod n = \boxed{17.225}^7 \pmod{52.841} = \boxed{1.855}$$

Tal y como se muestra en la traza, se ha logrado llegar al mensaje en claro, 17.225, en 41 iteraciones.

Sin embargo, ya que la realización de este ciclo es extremadamente elevada, estos ataques no se consideran una amenaza real para la seguridad del criptosistema.

Canal lateral

Ataque basado en información obtenida gracias a la propia implementación física de un sistema informático, en lugar de basarse en puntos débiles del algoritmo implementado. El principio subyacente es que los efectos físicos causados por el funcionamiento de un criptosistema pueden proporcionar información útil.

En muchas de las implementaciones de estos métodos que se han propuesto, el ataque de canal lateral se basa en la medición precisa de información sobre una pieza de hardware que realiza operaciones RSA, como por ejemplo la exponenciación modular. De este modo, por ejemplo, los picos de energía consumidos por la CPU durante el paso de un algoritmo sin una exponenciación modular pueden diferir de aquellos resultantes durante el paso de un algoritmo con ella.

Estas mediciones permiten localizar y monitorizar cada uno de los pasos de la exponenciación modular que implica la K_{pr}, lo que conduce a obtener cada bit individual de la clave en cada paso.

Factorización

La factorización es el problema que fija la cota en la seguridad en los cifrados de clave pública. Cualquier problema de seguridad en el criptosistema RSA se puede solventar. Sin embargo, la factorización es un problema del que no se puede saber la talla. Si se consigue factorizar *n*, en el caso general , se obtiene la clave privada RSA.

Como se ha comentado anteriormente, para casos generales no existe ningún algoritmo efectivo. En cambio, para casos particulares sí que los hay muy efectivos. Más adelante, en el apartado "Algoritmos de factorización" se presentarán los algoritmos de factorización más interesantes para casos particulares.

Objetivos

En este trabajo se presenta un nuevo algoritmo como una combinación de los algoritmos ya existentes de Dixon y de Pollard Rho. De manera general, el objetivo final que se ha marcado durante la realización del proyecto ha sido el desarrollo de un algoritmo derivado de Dixon, pero incorporando características obtenidas del algoritmo de Pollard Rho, con el objeto de estudiar el comportamiento comparado entre los tres algoritmos para un conjunto de números de prueba.

El estudio de este proyecto pretende profundizar en ambos algoritmos y comprobar si, en efecto, se puede conseguir un nuevo algoritmo con distintos resultados para los mismos valores. Partiendo de la hipótesis de que es posible mejorar el comportamiento de Dixon empleando la función de pseudoaleatoriedad, utilizada en el algoritmo de Pollard Rho, en lugar de la elección aleatoria de números de partida.

Los resultados se basarán en la comparación de los tres algoritmos citados. Midiendo el tiempo de respuesta y el coste computacional, este último será determinado por el número de exponenciaciones modulares realizadas en las distintas ejecuciones de cada algoritmo.

De este modo, se obtiene una medida independiente del hardware o la implementación utilizada, ya que considera la operación de mayor complejidad empleada en los algoritmos. Además, de este modo se disocia, en los algoritmos de Dixon y la propuesta que aquí se hace, el coste derivado de la resolución de un sistema de ecuaciones módulo 2, que puede resolverse utilizando otras técnicas.

ALGORITMO DE FERMAT

Este algoritmo busca representar un número impar *n* como una diferencia de dos cuadrados. Para así devolver el producto de una suma por una diferencia.

Este algoritmo es extremadamente eficiente cuando ambos factores *p* y *q* son muy cercanos entre sí. Suponiendo que estos, en efecto, sean próximos, la raíz de *n* es una aproximación de ambos. Por esto, el algoritmo inicializa *a* al techo de la raíz cuadrada de *n*, ya que se considera un buen punto de partida.

Además, partiendo del parámetro *a* ya calculado, $a^2 - n$ es siempre cercano a un número positivo muy reducido, próximo a 0, lo que significa que si resulta en un cuadrado perfecto es muy sencillo su cálculo.

Algoritmo de Fermat

Entrada: un número impar *n*

Salida: el producto de una suma por una diferencia

1: a = $\lceil \sqrt{n} \rceil$

Dentro de un bucle while True:

2: Si $a^2 - n$ no es un cuadrado perfecto:

2.1: a = a +1

3: Si $a^2 - n$ es un cuadrado perfecto:

3.1: $b^2 = a^2 - n$

3.2: **Devuelve** (a + b)(a - b)

En el peor de los caso este algoritmo devuelve *n*, lo que significa que *n* es un número primo.

Ejemplo 1:

Siendo n = 5959

a = 78

Como $78^2 - 5959$ no es un cuadrado perfecto:

a = 78 + 1 = 79

Hacemos lo mismo hasta que $a^2 - n$ es un cuadrado perfecto, cuando:

$a = 80$

$a^2 - n = 441$

$b = \sqrt{441} = 21$

Devuelve $(80 + 21) \cdot (80 - 21)$

ALGORITMO DE POLLARD P – 1

Este algoritmo se considera de propósito especial debido a que puede usarse para encontrar cualquier factor primo de un número compuesto *n* en el cual la factorización de bien (*p* - 1) o (*q* - 1) cuenten con primos pequeños.

Existen varias formas de abordar las construcción de un múltiplo de *p – 1*. Una de ellas, es calculando el factorial, ya que esta función puede ser calculada eficientemente de forma incremental.

Algoritmo de Pollard p - 1

Entrada: Un número entero positivo compuesto n

Salida: Un factor de n

1: Escoger un valor *a* aleatorio tal que $2 \leq a \leq n - 1$

2: Si $1 < mcd(a, n) < n$

2.1: **Devolver** mcd(a, n)

3: k = 2

En un bucle while True:

4: $a = a^k \bmod n$

5: $d = mcd(a - 1, n)$

6: Si $1 < d < n$:

6.1: **Devolver** d

7: Si d = n:

6.1: **Devolver** False

8: k + +

Ejemplo 2:

Siendo n = 2987

Supongamos que se escoge como valor aleatorio *a* = 3

Siendo *i* el número de iteración:

i = 0:

```
                k = 2, a^k! = 9, mcd(9 - 1, 2987) = 1
        i = 1:
                k = 3, a^k! = 729, mcd(729 - 1, 2987) = 1

        i = 2:
                k = 4, a^k! = 285, mcd(285 - 1, 2987) = 1
        i = 3:
                k = 5, a^k! = 2675, mcd(2675 - 1, 2987) = 1
        i = 4:
                k = 6, a^k! = 2171, mcd(2171 - 1, 2987) = 1
        i = 5:
                k = 7, a^k! = 175, mcd(175 – 1, 2987) = 29

Como mcd(174, 2987) = 29:
        Devuelve p = 29
```

ALGORITMO DE POLLARD RHO

El algoritmo rho de Pollard, descrito por Pollard en 1975, considera que uno de los factores de *n* es pequeño, en esta hipótesis se basa el algoritmo y su eficiencia. Debido a que se enfoca en un pequeño factor de *n*, se considera un algoritmo de factorización de propósito especial.

El algoritmo considera dos índices *a* y *b*, inicializados a un entero *k* y *F* una función pseudoaleatoria que hace avanzar a *b* el doble de rápido que a *a*, es decir, por cada iteración de la primera de las copias de la secuencia, la segunda hace dos iteraciones. Esto simula el recorrido del conjunto de forma que, de acuerdo con la paradoja del cumpleaños, puede encontrarse elementos que cumplen la condición después de $\sqrt{n}$ iteraciones.

El siguiente paso es calcula el máximo común divisor de la resta de ambos índices y *n*. Si dicho mcd resulta mayor a 1 y menor a *n*, hemos hallado un factor no trivial de *n*.

En caso de que el mcd resultase ser igual a *n*, el método ha resultado en fracaso, tan solo habría que cambiar la función pseudoaleatoria y probar de nuevo.

Algoritmo de Pollard Rho

Entrada: Un número entero positivo compuesto n

Salida: Un factor de n

1: Inicializar las variables *a* y *b* a 2

En un bucle while True:

2: $a = a^2 + 1 \bmod n$

3: $b = b^2 + 1 \bmod n$

4: $b = b^2 + 1 \bmod n$

5: $p = \text{mcd}(a - b, n)$

6: Si $1 < p < n$, **devuelve** p

7: Si $p = n$, **devuelve** n

Ejemplo 3:
Siendo n = 39617
En la primera iteración:
a = 5
b = 26
mcd (|5-26|, 39617) = 1
En la segunda iteración:
a = 26
b = 22543
mcd (|26-22543|, 39617) = 1
En la tercera iteración:
a = 677
b = 37403
mcd (|677-37403|, 39617) = 1
En la cuarta iteración:
a = 22543
b = 34307
mcd (|22543-34307|, 39617) = 173
Devuelve p = 173

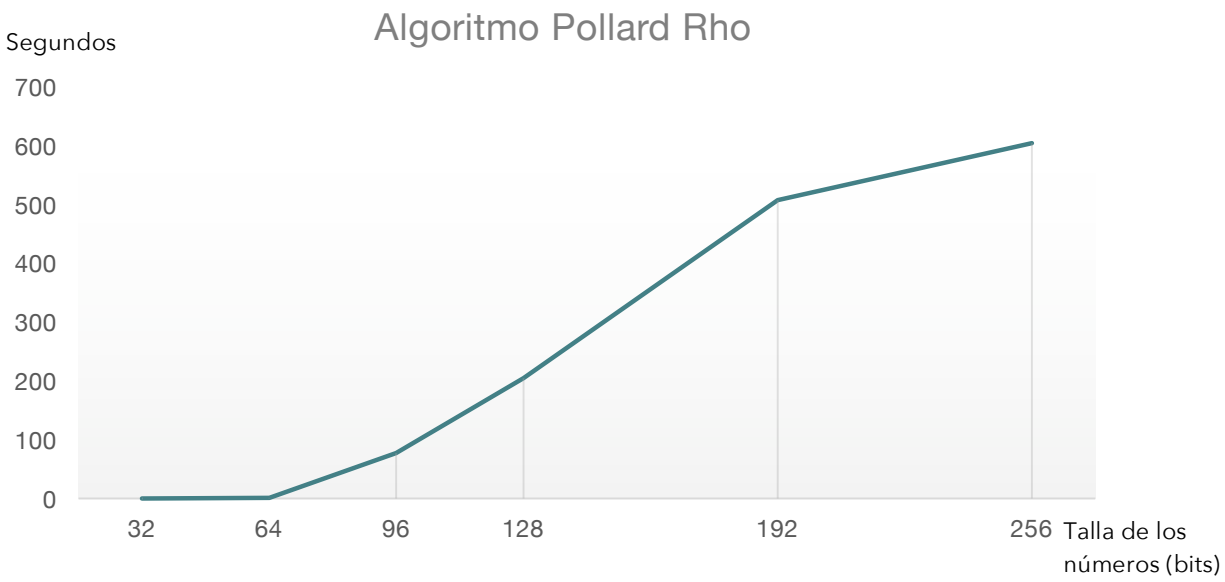
Comportamiento experimental

Este es uno de los algoritmos que consideramos en este trabajo como base para la propuesta de un nuevo método de factorización.

Para realizar un estudio comparado se han generado aleatoriamente números de seis tamaños distintos: 32 bits, 64 bits, 96 bits, 128 bits, 192 bits y 256 bits. Donde cada uno de ellos se compone de los primos p y q, exactamente del mismo tamaño. Por ejemplo, para la generación de un número a factorizar de 32 bits, se han generado dos número primos de 16 bits, para que el número resultante sean el producto de estos dos últimos números.

Para la generación de estos primos de ha utilizado el algoritmo de Miller-Rabin, incorporado en los métodos de Mathematica. Miller-Rabin es el test de primalidad más utilizado en la práctica, por ello es el estándar en la generación de números primos en aplicaciones como Mathematica.

De cada tamaño se han generado 500 números de acuerdo al procedimiento descrito. Esta cantidad se considera suficiente para obtener representatividad estadística. El mismo conjunto de número se ha utilizado posteriormente para las pruebas realizadas con los algoritmos de Dixon y el algoritmo híbrido.



El comportamiento del algoritmo es bueno, no mostrando indicativos del coste real del problema. Esto se debe a la naturaleza heurística del algoritmo y su buen comportamiento experimental al conjunto de números factorizados.

Este algoritmo, al ser un heurístico que considera que un factor p de $n = p \cdot q$ es pequeño, se comporta peor dado número mayores. En la experimentación realizada, el algoritmo de Pollard Rho no ha resultado satisfactorio para la talla de 256 bits en 121 ocasiones. Para que estos fallos se paliasen y, así, obtener la factorización de los números en cuestión, tan solo ha hecho falta modificar la función pseudoaleatoria cambiando la exponenciación, cuando el código original eleva el parámetro al cuadrado y le suma 1, la modificación eleva el parámetro al cubo y le añade 1.

ALGORITMO DE DIXON

En el siglo XX, Maurice Kraitchik propuso que se podían encontrar números *x* e *y* tales que

$$x^2 \equiv y^2 \pmod{n}$$

teniendo así una alta probabilidad de que mcd(*x* − *y*, n) fuera un factor no trivial de N. En esta idea se basa el algoritmo que propuso Dixon ya que trata de encontrar enteros *x* e *y* que cumplan la congruencia anterior.

Supongamos que el número a factorizar es *N* y la base de primos contenga los *i* primeros números primos, tal que B = {p_1, …, p_i} empezando por 2, elegimos un número entero aleatorio *a*, calculamos d(a) = a^2 (mod *n*) y procedemos a realizar un intento rápido de factorización de *d(a)*, utilizando exclusivamente los primos en B, considerando así *d(a)* como B-smooth. Sin profundizar en el intento, *d(a)* no es B-smooth, probamos con un número *a* diferente. Seguimos haciendo lo mismo hasta que tengamos suficientes números *d(a)*. En este caso necesitamos más de *i* números *a* para los cuales hemos encontrado la factorización de *d(a)*.

Si *d(a)* ha sido factorizado completamente, entonces podemos escribirlo como d(a) = $p_1^{e_1} * \cdots * p_i^{e_i}$. La factorización de *d(a)* puede ser representada por el vector v(a) = (e_1, . . . , e_i).

Si ahora todas las entradas del vector *v(a)* son pares, entonces tenemos que *d(a)* es un cuadrado perfecto y por lo tanto hemos encontrado dos números

$$x = a \text{ e } y = \sqrt{d(a)}$$

tales que

$$x^2 \equiv y^2 \pmod{n}$$

Ejemplo 4:

Siendo n = 2491 y b = 10

B = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

Suponiendo el valor *a* elegido aleatoriamente es a = 347

d(a) = 841

v(a) = [0, 0, 0, 0, 0, 0, 0, 0, 0, 2]

Como todas las entradas del vector (a) son pares:

x = 347

y = $\sqrt{841}$ = 29

Devuelve p = mcd (x-y, n) = 53

Aunque esto no ocurriera, podemos encontrar una suma de distintos vectores *v(a)* que contenga todas las entradas pares. Para ello, definimos los vectores w(a) = (b_1, . . . , b_i) con b_j = 0, si e_j es par o b_j = 1, si e_j es impar y sobre dichos vectores *w(a)* se aplica eliminación gaussiana, obteniendo así el subconjunto de vectores *v(a)* cuya suma resulta un vector con coordenadas pares.

Llegados a este punto ya tenemos un subconjunto de números *d(a)* tales que el producto será un cuadrado perfecto. Por lo que tenemos una congruencia de la forma $d(a_1)*d(a_2)* \cdots *d(a_i) \equiv a^2{}_1*a^2{}_2* \cdots *a^2{}_i \pmod{n}$ donde ambos lados son cuadrados perfectos, por lo que tendríamos nuevamente números *x* e *y* tales que $x^2 \equiv y^2 \pmod{n}$. En caso de no obtener un factor no trivial de *N* a partir de *x* e *y*, habría que buscar otro subconjunto de vectores *v(a)* que para los que los vector *w(a)* correspondientes sean linealmente independientes.

Ejemplo 5:

Siendo n = 84923 y b = 4

B = [2, 3, 5, 7]

Suponiendo el valor *a* elegido aleatoriamente es a_0 = 513

$d(a_0)$ = 8400

$v(a_0)$ = [4, 1, 2, 1]

Como no todas las entradas del vector $v(a_0)$ son pares seguimos buscando números *a*:

a_1 = 537

$d(a_1)$ = 33600

$v(a_1)$ = [6, 1, 2, 1]

Ahora tenemos dos vectores *v* que combinados dan como resultado un vector con todos los componentes pares:

$v(a_{0,1})$ = [10, 2, 4, 2]

x = 513 * 537 = 275481

y = $\sqrt{8400 * 33600}$ = 16800

Devuelve p = mcd (x-y, n) = 163

A continuación se muestra el algoritmo:

Algoritmo de Dixon

Entrada: Un número entero positivo compuesto n

Salida: Un factor de n

1: Elección de un tamaño para la base de factores primos

2: Elección de un número de vectores resultantes para el cálculo (denominado *éxitos*)

En un bucle, mientras que un contador *i* sea menor a *éxitos*:

3: Elección de un número aleatorio *a*

4: Factorización y almacenamiento de a_i según la base de factores primos

5: Almacenamiento del vector resultante módulo 2

Una vez terminado el bucle:

6: Si existe un vector donde todos los componentes son 0:

6.1: x = *a*

6.2: y = $\sqrt{d(a)}$

7: Si no existe:

7.1: Se hace una combinación de los vectores módulo 2 mediante la función XOR, hasta obtener uno de todo 0 y se almacenan su índices *i*:

7.2: x = $\prod_0^i a_i$

7.3: y = $\sqrt{\prod_0^i d(a_i)}$

8: **Devuelve** mcd (x – y, n)

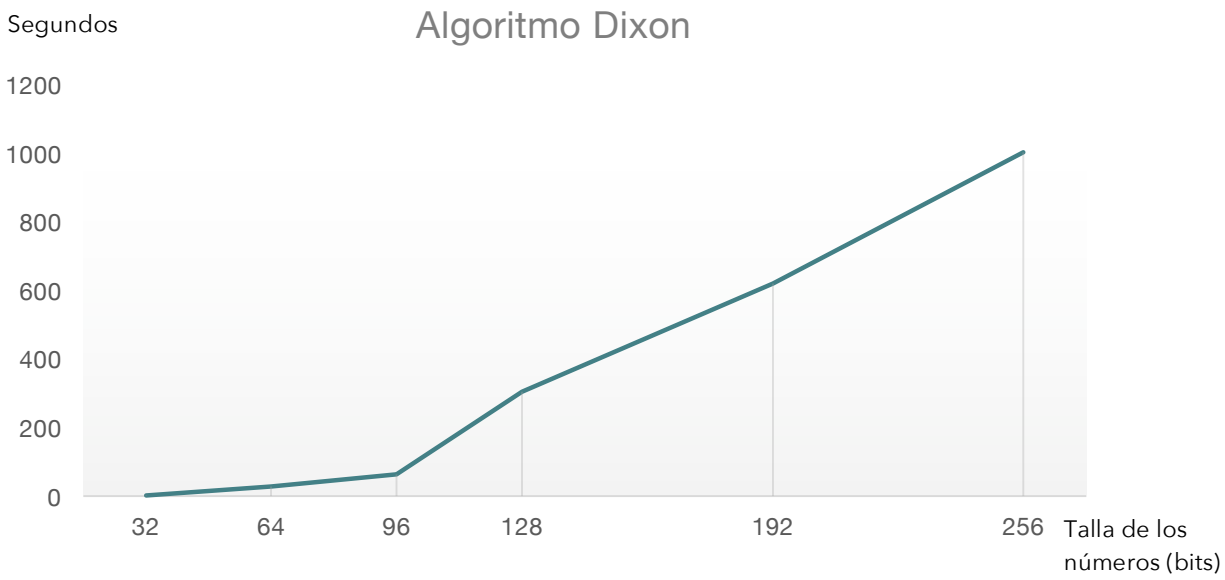
De este algoritmo cabe destacar su importancia, puesto que es el germen del algoritmo de la criba cuadrática.

Carl Pomerance, un matemático estadounidense especializado en teoría de números, propuso en 1981 un algoritmo llamado criba cuadrática que extendía las ideas de Dixon y de Kraitchik. Básicamente se trata de una optimización del método de Dixon, ya que la búsqueda ya no es aleatoria, sino que comienza desde la raíz de *n*.

Selecciona los valores de *a* cerca de la raíz cuadrada de *n* tal que a^2 mod *n* sea un número pequeño, lo que aumenta en gran medida la posibilidad de obtener un número B-*smooth*.

Comportamiento experimental

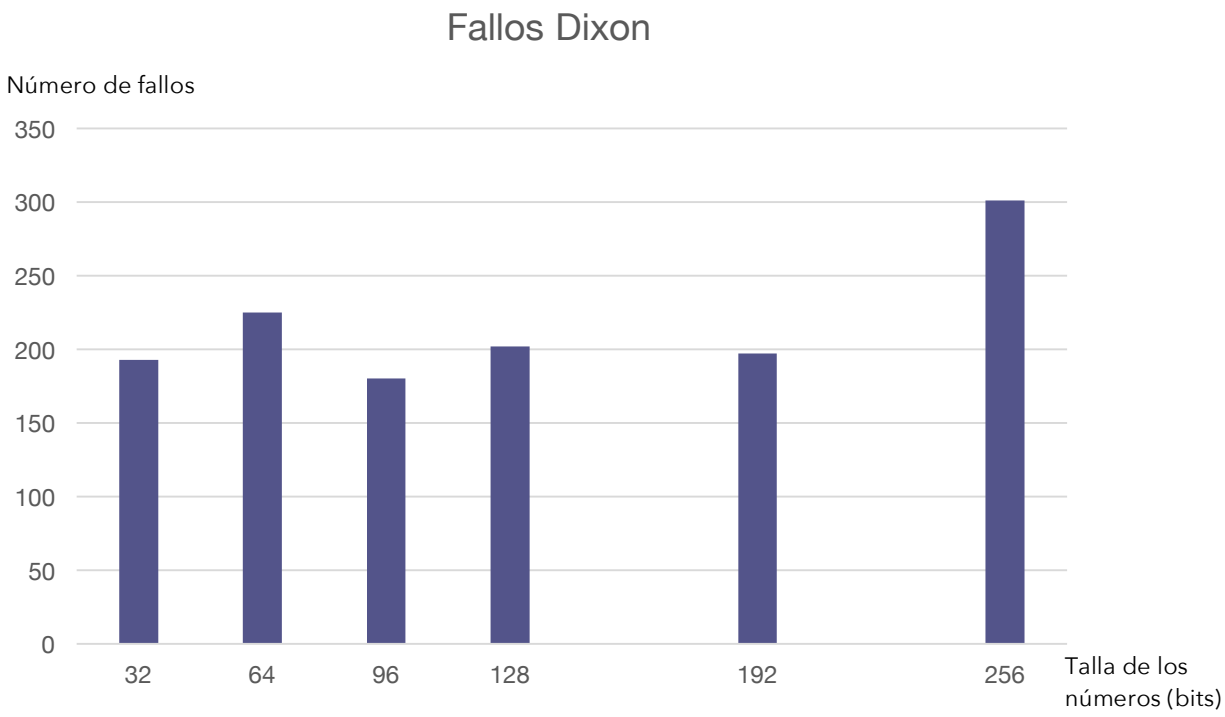
En el algoritmo de Dixon, se ha utilizado el mismo conjunto de números obtenidos para probar el algoritmo de Pollard Rho. Como se puede observar en la gráfica de abajo, sigue un crecimiento exponencial para el conjunto de números analizado.



Su tiempo medio de respuesta para cada talla, aunque no es extremadamente grande, es mucho mayor al del algoritmo de Pollard Rho.

Además, al tratar de un algoritmo con la elección del número de partida aleatoria, hay casos en los que el algoritmo puede derivar en fallo. Este caso, no significa que el algoritmo no sirva para factorizar ese número, solo que la elección el número aleatorio escogido no ha sido satisfactoria, por lo que, muy probablemente, una segunda ejecución de la factorización sea efectiva. En la experimentación realizada, este comportamiento se ha dado un total de 1.298 veces de las 3.000 ejecuciones.

A continuación se muestra una gráfica acerca de los fallos:



Observando la gráfica, se puede comprobar que el número de fallos no sigue ninguna distribución ordenada. Esto es algo esperado, debido a la naturaleza aleatoria del parámetro *a* en el algoritmo de Dixon.

ALGORITMO HÍBRIDO

Este algoritmo es una variación del algoritmo de Dixon, cambiando el parámetro que escoge los números de partida, que se denominará *a* en este documento. Para esto, se conjunta con la función pseudoaleatoria empleada en Pollard Rho. De manera que la elección de número de partida deja de ser aleatoria para seguir una secuencia pseudoaleatoria.

La hipótesis se basa en que del nuevo criterio se consiguen números seleccionados pseudoaleatoriamente, buscando obtener así un recorrido más racional de la secuencia de números. En esto mismo se basa la hipótesis, en que esta modificación permite un recorrido más eficiente del conjunto, conduciendo a la localización de valores *B-smooth* con menor costo temporal.

En Dixon, los números de partida son aleatorios.

$$a = \text{numero aleatorio entre 2 y } n - 1$$

Sin embargo, en el algoritmo híbrido, en la primera iteración

$$a = \sqrt{n}$$

A partir de la segunda iteración *a* se determina por la función pseudoaleatoria de Pollard Rho, obteniendo como parámetro la antigua *a*.

$$a = a^2 + 1 \pmod{n}$$

Algoritmo híbrido

Entrada: Un número entero positivo compuesto n

Salida: Un factor de n

1: Elección de un tamaño para la base de factores primos

2: Elección de un número de vectores resultantes para el cálculo (denominado *éxitos*)

3: $a = \lceil \sqrt{n} \rceil$

En un bucle, mientras que un contador *i* sea menor a *éxitos*:

4: Factorización y almacenamiento de a_i según la base de factores primos

5: Almacenamiento del vector resultante módulo 2

6: $a = a^2 + 1 \pmod n$

Una vez terminado el bucle:

7: Si existe un vector donde todos los componentes son 0:

7.1: $x = a$

7.2: $y = \sqrt{d(a)}$

8: Si no existe:

8.1: Se hace una combinación de los vectores módulo 2 mediante la función XOR, hasta obtener uno de todo 0 y se almacenan su índices *i*:

8.2: $x = \prod_0^i a_i$

8.3: $y = \sqrt{\prod_0^i d(a_i)}$

9: **Devuelve** mcd (x – y, n)

Ejemplo 6:

Siendo n = 84923, b = 4 y éxitos = 3

B = [2, 3, 5, 7]

El primer valor de *a* corresponde a $a_0 = 292$, pero como no es *B-smooth*, continúa el bucle $a = a^2 + 1 \pmod n$ hasta llegar al valor $a_0 = 48871$

$d(a_0) = 189$

$v(a_0) = [0, 3, 0, 1]$

Seguimos buscando valores a_i *B-smooth* hasta alcanzar el número de éxitos:

a_1 = 76012
$d(a_1)$ = 2916
$v(a_1)$ = [2, 6, 0, 0]
a_2 = 62968
$d(a_2)$ = 84000
$v(a_2)$ = [5, 1, 3, 1]

Ordenamos los vectores módulo 2 en una matriz triangular, donde el primer valor es el índice del número *a* al que corresponde:

[2 [1, 1, 1, 1]]
[0 [0, 1, 0, 1]]
[1 [0, 0, 0, 0]]

Como tenemos un vector de todo 0, ya tenemos una solución:

x = a_1 = 76012
y = $\sqrt{d(a_1)}$ = 54

Devuelve p = mcd (x-y, n) = 163

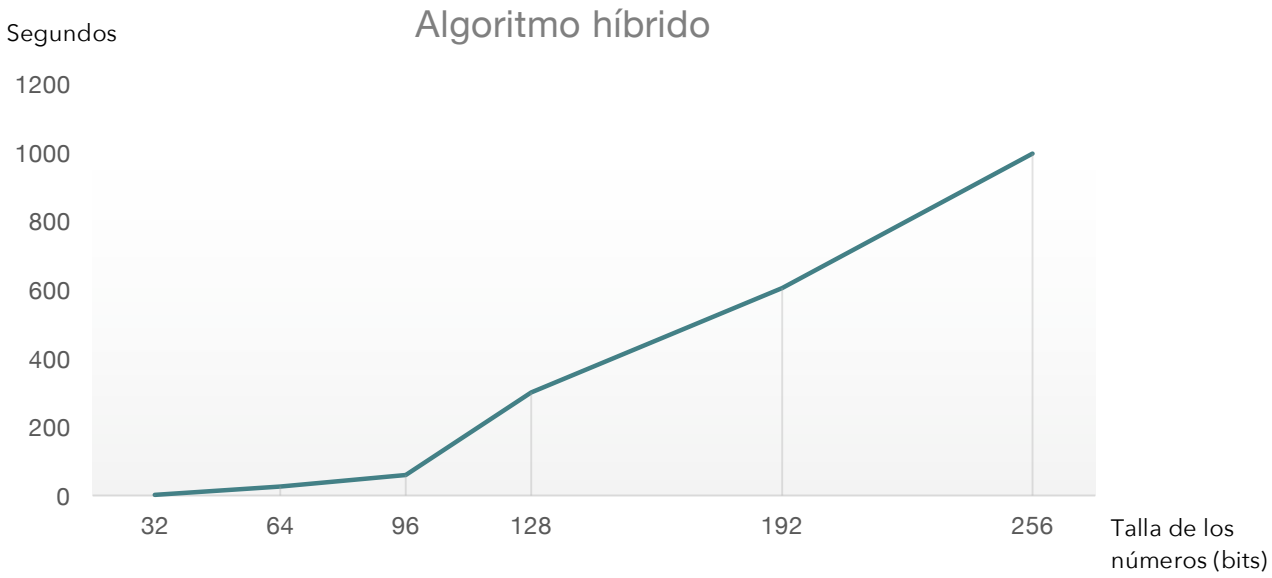
COMPORTAMIENTO EXPERIMENTAL

Antes de hablar del comportamiento experimental, se va a comentar las expectativas de este.

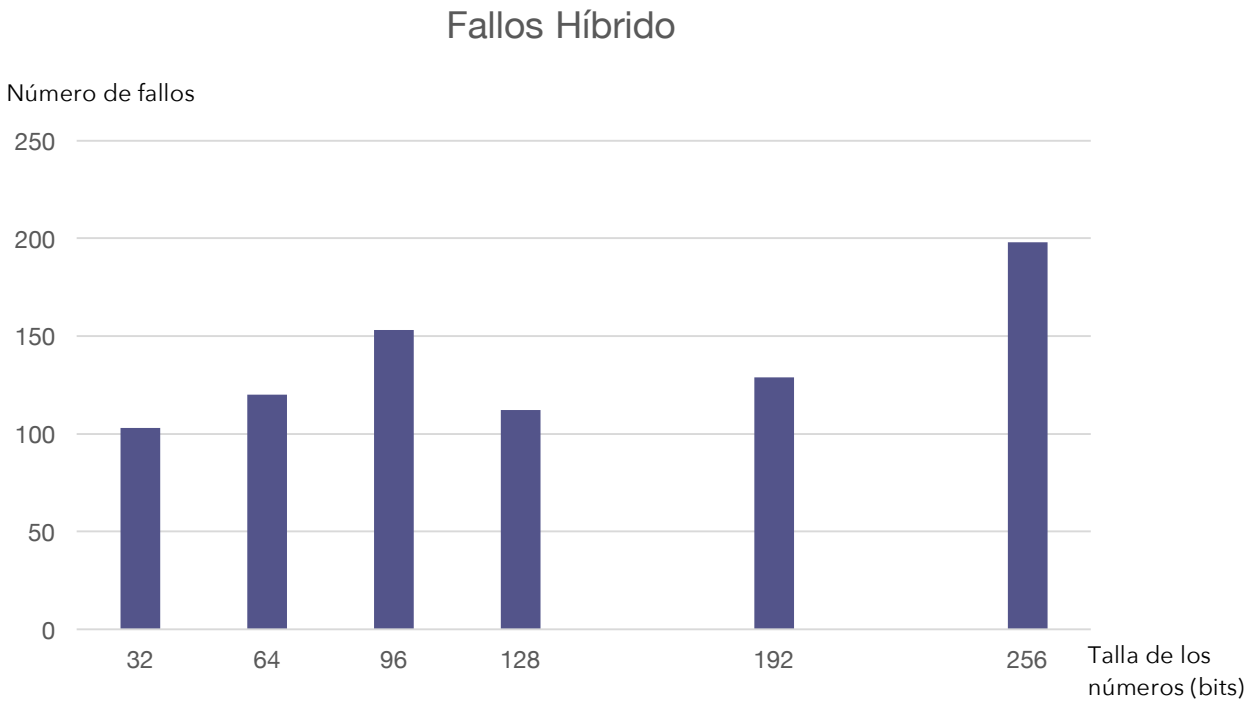
La hipótesis en la que se basa este proyecto, como ya se ha comentado anteriormente, es que un recorrido pseudoaleatorio del conjunto de números a factorizar aceleraría el algoritmo comparado con la selección de números aleatorios. Lo que se esperaba de este algoritmo era que fuera más eficiente que el de Dixon y que el de Pollard Rho.

Como se puede comprobar en la gráfica, su comportamiento ha seguido un crecimiento exponencial, al igual que el algoritmo de Dixon.

La media de los tiempos de factorización para cada talla de números es bastante elevada, pese a que la evolución del recorrido del conjunto de números sigue cierto orden siendo pseudoaleatoria.

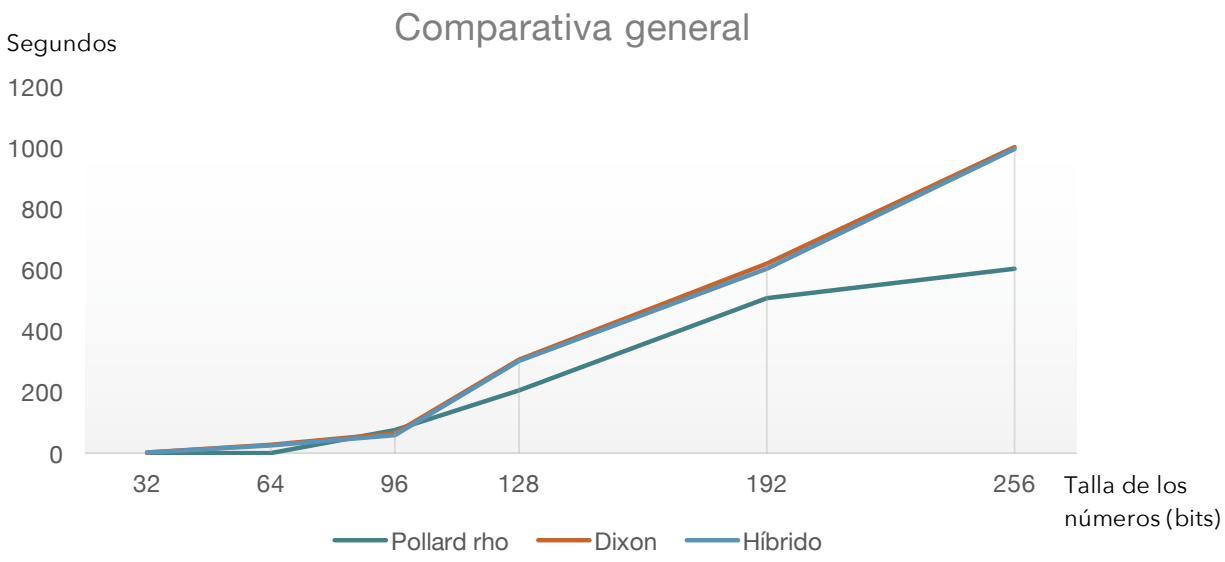


A continuación se muestra una gráfica sobre los fallos en el algoritmo híbrido:



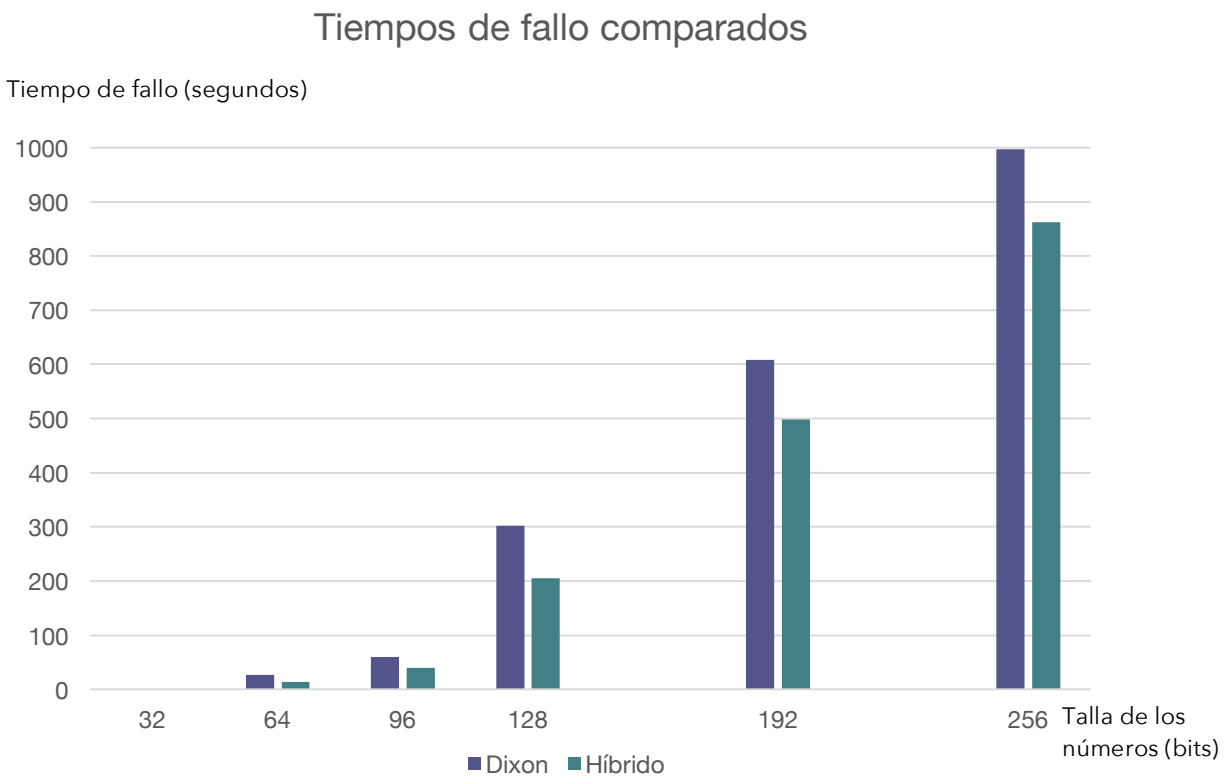
RESULTADOS COMPARADOS

Se comenzará explicando una gráfica donde se comparan los tres algoritmos juntos, según el tiempo medido en segundos, en el eje Y, y el tamaño en bits de los números, en el eje X. Cada línea representa un algoritmo distinto. Cada punto representa la media de tiempo de la factorización de los 500 números de dicho grupo.

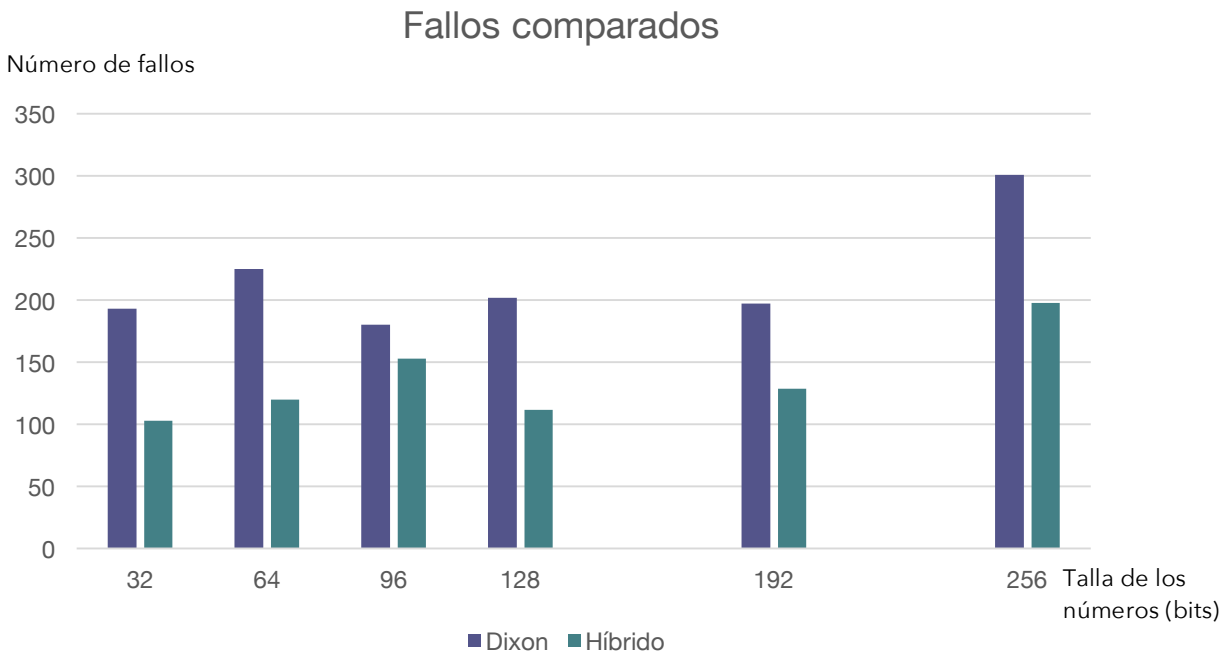


No obstante, sí que se han podido observar diferencia experimentales en la comparativa de los algoritmos de Dixon y el híbrido.

- Mientras que el algoritmo de Dixon ha resultado en fallo 1.298 veces en el total de las pruebas realizadas, el algoritmo híbrido lo ha hecho en un total de 996 veces.
- Si el resultado derivaba en fallo, el algoritmo híbrido lo ha hecho en menos tiempo que el algoritmo de Dixon, como se muestra en la siguiente gráfica:



En esta gráfica se representa los tiempos de ejecución de los algoritmos de Dixon y el híbrido cuando la ejecución falla.



En esta última gráfica se muestra la comparativa del número de fallos en ambos algoritmos. Como se puede observar, existen diferencias entre el número de fallos del algoritmo de Dixon y el número de fallos del algoritmo híbrido.

En efecto, aunque no se muestra en los tiempos de ejecución, el recorrido pseudoaleatorio propuesto sí ha supuesto una diferencia respecto al recorrido completamente aleatorio de Dixon, provocando un menor número de fallos en cada una de las tallas.

TECNOLOGÍAS UTILIZADAS

Para realizar este estudio se han hecho uso de algunas tecnologías de programación y de ejecución.

- Python: ha sido el lenguaje con el que se han desarrollado los algoritmos. Se trata de un lenguaje de programación sencillo, de fácil legibilidad y muy semejante a Java. Apareció en 1991, pero sigue estando a la orden del día. Este pasado mes de junio se publicó una nueva versión del lenguaje.

- Wolfram Mathematica: herramienta con la que ha sido posible generar los números a factorizar. Para ello, se ha hecho uso de la función *RandomPrime* para generar, por cada número factorizado, dos números primos de la mitad de la talla del número a factorizar. De esta manera nos aseguramos que todo el conjunto de número se trata de números $p \cdot q$. Esta aplicación se hace servir de Miller-Rabin para la generación de números primos.

- Atom: editor de código fuente en el se ha llevado a cabo la programación de los algoritmos. Es de código abierto, apareció en febrero de 2014 y tiene paquetes que se le pueden instalar para que el entorno se adapte al lenguaje utilizado en cada desarrollo.

- Terminal del ordenador: mediante ella se han ejecutado los diferentes algoritmos, llevando así a cabo las pruebas.

CONCLUSIONES

En el estudio llevado a cabo, el algoritmo de Pollard Rho, ha resultado ser el más eficiente de los tres algoritmos comparados, para el conjunto de números factorizados en este proyecto.

Su respuesta a la factorización de números de tallas reducidas, hasta 96 bits, ha sido extremadamente eficiente, siendo el tiempo medio de las factorizaciones del conjunto de números menor a 100 segundos.

Lamentablemente, la hipótesis presentada en este trabajo ha demostrado ser errónea, debido a que el coste temporal del nuevo algoritmo creado y el coste del algoritmo de Dixon han resultado prácticamente iguales en este estudio.

La realización de este proyecto ha sido muy interesante, puesto que los resultados obtenidos no han sido los esperados. Se consideraba que un recorrido más ordenado iba a aportar diferentes resultados respecto a un recorrido completamente aleatorio.

REFERENCIAS

[1] **New Directions in Cryptography**, Whitfield Diffie and Martin E. Hellman, 1976.

[2] **Una introducción a la criptografía de clave pública**, Wolfgang Willems e Ismael Gutiérrez García, 2008.

[3] **Handbook of Applied Cryptography**, A. Menezes, P. van Oorschot, and S. Vanstone, *CRC Press*, 1996.

[4] **A Method for Obtaining Digital Signatures and Public-Key Cryptosystems**, A. Shamir R.L. Rivest and L. Adleman, *Magazine Communications of the ACM*, 1978.

[5] **Network Security Essentials Applications and Standards**, William Stallings, 2010.

[6] **Applied Cryptanalysis: Breaking Ciphers in the Real World**, Mark Stamp, Richard M. Low, 2007.

[7] https://wikileaks.org/