UNIVERSIDAD DE CANTABRIA

Facultad de Ciencias

Implementación del criptosistema de Cocks

(Implementation of the Cocks cryptosystem)

Trabajo de fin de grado
para acceder al

GRADO EN MATEMÁTICAS

Autora: Denisse Gómez González

Codirector: Domingo Gómez Pérez

Codirectora: Ana Isabel Gómez Pérez

Junio de 2021

Agradecimientos

Me gustaría mostrar mi más sincero agradecimiento a todos aquellos que, directa o indirectamente, han hecho posible la realización de este trabajo, así como a aquellos me han apoyado durante esta etapa académica.

En primer lugar, quiero dar las gracias a mi codirector, Domingo Gómez, por acompañarme y guiarme en este camino. Gracias por haberme dado la oportunidad de realizar este trabajo con la temática que tenía en mente, así como por proporcionarme las pautas y conocimientos necesarios para poder afrontar este trabajo. Gracias también a mí codirectora, Ana Isabel Gómez, por la ayuda y el conocimiento proporcionado en la elaboración de la memoria. Por vuestra implicación y por todo el tiempo que habéis dedicado a ayudarme desde un primer momento, gracias a los dos.

Por otra parte, también quisiera agradecer a todos los profesores de la Facultad de Ciencias que me han acompañado día a día en las aulas. Gracias por vuestra gran labor docente, dedicación y por fomentar el desarrollo de mi curiosidad y conocimiento matemático. Y a mis compañeros de clase por estos cinco años compartidos, llenos de buenos momentos, amistad y compañerismo. Parece mentira que esto se acabe, tengo suerte de haberos conocido.

Finalmente, me gustaría dar las gracias a mi familia y amigos, sobre todo a mis padres, mi hermana y mi abuela. Gracias a vosotros por todo el apoyo incondicional que me dais cada día, por toda la confianza que habéis puesto en mí y por cuidar de mí. Soy quien soy gracias a vosotros.

Implementación del criptosistema de Cocks

Resumen

palabras clave: cifrado basado en identidades, criptografía asimétrica, criptosistema de Cocks, residuosidad cuadrática

Los sistemas de encriptación basados en la identidad (IBE, Identity-Based Cryptosystem) son criptosistemas donde la clave pública de cada usuario se calcula a partir de un «string» que lo identifique.

En este trabajo, veremos la implementación del criptosistema de Cocks, un sistema de encriptación basado en la identidad cuya seguridad reside en la dificultad de resolver el problema de la residuidad cuadrática.

Además del estudio teórico del sistema de Cocks y de su implementación en python, se presentarán los fundamentos de teoría de complejidad, la dificultad del problema de la residuosidad cuadrática, las reducciones de seguridad, así como aplicaciones del criptosistema de Cocks a la computación con texto encriptado.

(Implementation of the Cocks cryptosystem)

Abstract

keywords: identity based encryption, asymmetric key cryptography, the Cocks cryptosystem, quadratic residuosity

The Identity-Based Cryptosystem (IBE) are cryptosystems where the public key of each user is calculated from a «string» that identifies it.

In this work, we will see the implementation of the Cocks cryptosystem, an identity-based encryption system whose security lies in the difficulty of solving the quadratic residuosity problem.

In addition to the theoretical study of the Cocks system and its implementation in python, will be presented the fundamentals of complexity theory, the difficulty of the quadratic residual problem, the security reductions, as well as Cocks cryptosystem applications to encrypted text computing.

Índice

1 Introducción

La necesidad de mantener a salvo nuestra información no es nueva, es bastante más antigua de lo que nos podemos llegar a imaginar. El origen de esta disciplina tan antigua (la criptografía) se remonta al nacimiento de nuestra civilización, desde que el hombre aprendió a comunicarse.
El ser humano tuvo que encontrar medios para asegurar la confidencialidad de una parte de sus comunicaciones. Un ejemplo son los jeroglíficos del Antiguo Egipto, los cuales suelen tomarse como uno de los primeros ejemplos de "escritura oculta" de la historia. Sin embargo, no fue hasta mediados del siglo VI antes de Cristo cuando se encuentra el primer testimonio del uso deliberado de métodos técnicos que permitieran cifrar mensajes.
A principios de los años 70, la criptografía inicio la época de los circuitos integrados y el desarrollo en los algoritmos, concretamente, el uso de las matemáticas modernas. Se han empleado muchas áreas de las matemáticas en la creación de criptosistemas, como, por ejemplo: los cuerpos finitos, la factorización de números enteros, los polinomios de permutación o la teoría de curvas elípticas.

El cifrado de la información o criptografía es la ciencia que estudia la escritura oculta, es decir, el diseño de métodos y algoritmos de cifrado para ocultar el contenido de un mensaje, siendo este públicamente disponible. Su objeto de estudio son los problemas teóricos relacionados con la seguridad en el intercambio de mensajes en clave entre un emisor y un receptor a través de un canal de comunicación. Puede llegar a tener distintos objetivos, desde proteger la confidencialidad de informaciones a garantizar la autenticación de datos.

Esta ciencia se divide en dos grandes ramas: la criptografía propiamente dicha, ocupada del cifrado de mensajes en clave y del diseño de criptosistemas; y el criptoanálisis, que trata de descifrar los mensajes en clave, rompiendo así su criptosistema.
Al mensaje que se pretende cifrar se le suele denominar texto en claro y al proceso encargado de ocultar el contenido mediante una serie de transformaciones regidas por un parámetro o valor secreto, la clave, se le denomina cifrado. El esfuerzo necesario para recobrar el mensaje sin la clave privada es demasiado grande. Por ello, cualquier persona ajena a la comunicación solamente será capaz de ver un escrito sin sentido y el contenido del mensaje le quedará totalmente oculto. Al mensaje cifrado se le suele denominar texto cifrado o criptograma, y al proceso de obtener el texto en claro a partir de un mensaje cifrado se le denomina descifrar el mensaje.

El concepto de sistema criptográfico se formaliza matemáticamente de la siguiente manera:

Definición 1.0.1 *Un criptosistema o esquema de cifrado se define como una tupla* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ *donde:*

1. $\mathcal{P}$ *es un conjunto denominado "espacio de texto plano". Sus elementos son textos planos y son los textos no cifrados.*
2. $\mathcal{C}$ *es un conjunto denominado "espacio de texto cifrado". Sus elementos son textos cifrados.*
3. $\mathcal{K}$ *es un conjunto denominado "espacio de clave". Sus elementos son las claves.*
4. $\mathcal{E} = \{E_k : k \in \mathcal{K}\}$ *es un conjunto de funciones* $E_k : \mathcal{P} \to \mathcal{C}$*. Sus elementos son "funciones de cifrado".*

5. $\mathcal{D} = \{D_k : k \in \mathcal{K}\}$ *es un conjunto de funciones* $D_k : \mathcal{C} \to \mathcal{P}$. *Sus elementos son "funciones de descifrado".*

Para cada $e \in \mathcal{K}$, *existe un* $d \in \mathcal{K}$ *tal que* $D_d(E_e(p)) = p$ *para todo* $p \in \mathcal{P}$.

1.1. Criptosistemas de clave pública

Un año importante para la criptografía fue el de 1976, cuando Whitfield Diffie y Martin Hellman crean el concepto de criptosistema de clave pública [31] [19].
Surge como solución a la cada vez más frecuente incapacidad de acordar una clave por un medio seguro y al número de claves necesarias para conectar varios usuarios entre sí. Se trata de un sistema donde la clave de cifrado se puede encontrar en un directorio público de usuarios, pero la clave de descifrado es diferente y no se obtiene fácilmente de la primera.
Esto motiva multitud de investigaciones y discusiones sobre la criptografía de clave pública y su impacto, hasta el punto de que la consideraban una amenaza peligrosa para la seguridad nacional.

Los criptosistemas asimétricos o de clave pública son los métodos criptográficos que usan una clave pública (a la que cualquier persona tiene acceso) para el envío de mensajes y una clave privada (que se mantiene en secreto) para descifrarlos.
Por tanto, en un proceso criptográfico, el receptor R usa la clave pública del emisor S para cifrar el mensaje destinado a S. Tras ser enviado, S lo recibe y procede a descifrar el mensaje usando para ello su clave privada. Es decir, la clave pública se usa para cifrar y la clave privada para descifrar.

Definición 1.1.1 *Un criptosistema de clave pública es una tupla* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ *tal que para cada usuario* U *del sistema se genera una clave pública* $a \in \mathcal{K}$ *y una clave privada* $r \in \mathcal{K}$ *tales que:*

1. *Para todo mensaje* $M \in \mathcal{P}$, *se verifica que* $D(\mathcal{E}(M, a), r) = M$.
2. *Las operaciones de cifrado* $C' = \mathcal{E}(M, a)$ *y descifrado* $M' = \mathcal{D}(C, a)$ *para cualesquiera* $M \in \mathcal{P}$ *y* $C \in \mathcal{C}$ *son fácilmente calculables (en el sentido de la complejidad, es decir, son algoritmos polinómicos, ver 2.3).*
3. *Para casi todo mensaje* $M \in \mathcal{P}$, *se confía en que hallar un equivalente a* r *o descifrar un criptograma* $C \in \mathcal{C}$ *a partir únicamente de* C *y de* a *sea computacionalmente difícil.*
4. *Es computacionalmente factible generar un par de claves* a *y* r.

Como hemos mencionado antes, las claves públicas de este criptosistema se almacenan en un directorio público en Internet, que debe de ser práctico y fiable. Este recibe el nombre de Infraestructura de clave pública (PKI, Public Key Infrastructure) y es necesario para crear, manejar, almacenar, distribuir y revocar certificados digitales basados en criptografía asimétrica.

El PKI tiene como principal objetivo garantizar seguridad y confianza en una comunicación electrónica entre las partes involucradas. Además, es el encargado de gestionar y de distribuir las claves públicas, asegurándose de que estas son auténticas, permitiendo a los usuarios intercambiar datos de forma segura.

Con el fin de asegurar la vinculación o correspondencia unívoca entre la identidad de un sujeto y su clave pública se emplea la Autoridad de Certificación (CA, Certificate Authority). Esta entidad es la encargada de emitir y revocar los certificados digitales utilizados por usuarios aceptados por las Autoridades de Registro dependientes de ella, firmarlos y almacenarlos en un repositorio de acceso público. Las Autoridades de Registro (RA, Registration Authority) se encargan de comprobar la veracidad y de la corrección de los datos que aportan los solicitantes en sus peticiones, y enviarlas a una

autoridad de certificación para que sean procesadas, es decir, constituyen el punto de comunicación entre los usuarios de la PKI y la autoridad certificadora.

Un certificado es el medio utilizado por una PKI para comunicar el valor de una clave pública o información sobre ella. Es decir, un certificado es un documento electrónico que contiene una clave pública e información de identificación sobre la entidad que la controla. Los certificados digitales deben estar firmados por la CA que los ha emitido para asegurar su validez.
El propietario de la clave privada recibe una copia del certificado y otra se almacena en un repositorio de certificados que es accesible por otros.

Una firma digital o electrónica es una aplicación de la criptografía asimétrica que permite identificar a una persona ante un sistema informático. Sirve para acreditar quién es el firmante o emisor del mensaje (autentificación) y también de asegurar que este no ha sido modificado por terceros.

Por otra parte, está la Autoridad de Validación (VA, Validation Authority), que es la entidad encargada de validación de las firmas digitales usando la llave pública. Comprueba la validez y no-cancelación de los certificados digitales de los suscriptores.
Dado que la información del certificado puede cambiar a lo largo del tiempo, el usuario necesita estar seguro de que los datos contenidos en él son fiables. Esto puede realizarse de dos modos: puede ser el propio usuario el que solicite directamente a la CA la confirmación de la validez de un certificado cada vez que lo va a usar o puede ser la propia CA quien incluya información relativa al periodo de validez en el certificado (mediante un rango de fechas).
Un tema relacionado es la revocación de certificados, por la que se le hace saber al usuario que la información contenida en un certificado ya no es válida. Esto puede ocurrir cuando la clave privada del sujeto queda comprometida o cuando la información incluida en el certificado ha variado. El método más común es utilizar una Lista de Revocación de Certificados (CRL, Certificate Revocation List), la cual es firmada y emitida periódicamente por una CA.

Todo el proceso explicado de la PKI se recoge esquemáticamente en la ilustración 1.

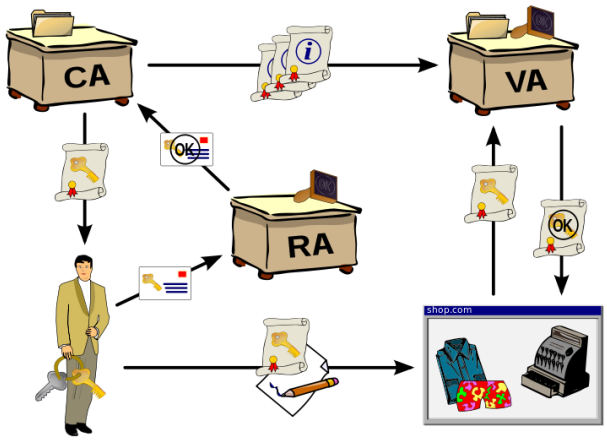

Ilustración 1: Esquema de la infraestructura de clave pública (PKI). [5]

Hoy en día, son muchas las propuestas existentes de infraestructuras de clave pública para Internet, pero no hay ninguna que haya adquirido un uso generalizado en la red; de hecho, cada vez se extiende más la idea de que en un futuro próximo habrá distintos tipos de PKI operando conjuntamente.

1.2. Sistemas criptográficos basados en identificador (IBE)

El cifrado basado en identidad (IBE, Identity Based Encryption) es un tipo especial de cifrado de clave pública que permite que cualquier cadena sea una clave pública.

El IBE es mencionado por primera vez por Adi Shamir en 1984 [25] cuando esboza las propiedades que debería tener dicho sistema y cómo podría utilizarse, aunque no pudo encontrar una tecnología segura y factible que funcionara como él describió. Pretendía aprovechar la facilidad de uso de IBE en relación con otras tecnologías.
De este modo:

Un esquema basado en identidad se parece a un sistema de correo ideal: si sabes el nombre y la dirección de alguien puedes enviarle mensajes que solo él puede leer, y puedes verificar las firmas que solo él pudo haber producido. Hace que los aspectos criptográficos de la comunicación sean casi transparentes para el usuario, y puede ser utilizado eficazmente incluso por personas que no saben nada sobre claves o protocolos. [25]

Para utilizar un sistema IBE, un usuario normalmente necesita obtener un conjunto de parámetros públicos de un tercero de confianza. Con estos parámetros, el usuario puede calcular la clave pública IBE de cualquier usuario y usarla para cifrar información destinada a ese usuario. Este proceso se muestra en la ilustración 2.

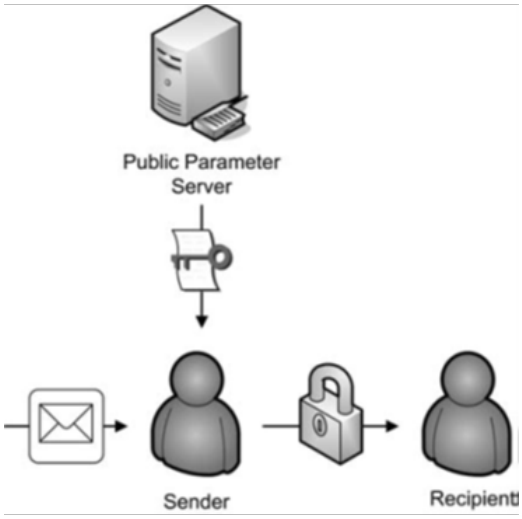

Ilustración 2: Cifrado con un sistema IBE. [25]

El destinatario de la información encriptada por el sistema IBE se autentica en un generador de claves privadas (PKG) donde un tercero de confianza calcula la clave privada IBE que corresponde a una clave pública IBE particular. El PKG normalmente utiliza información secreta llamada "parámetro de seguridad" y la identidad del usuario para calcular dicha clave privada. Una vez calculada esta clave privada, se distribuye al usuario autorizado. Esto se muestra en la ilustración 3.

Definición 1.2.1 *Un esquema IBE consta de cuatro algoritmos:*

1. *Configuración. La configuración es el algoritmo con el que se establecen los parámetros necesarios para los cálculos de IBE. Toma como entrada un parámetro de seguridad k y devuelve parámetros del sistema y la clave maestra. Los parámetros del sistema se conocerán públicamente, mientras que la clave maestra será conocida solo por el generador de claves privadas (PKG).*

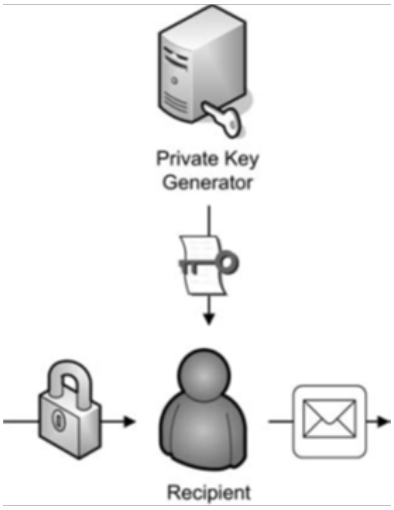

Ilustración 3: Descifrado con un sistema IBE. [25]

2. *Extracción. La extracción es el algoritmo para calcular una clave privada IBE a partir de los parámetros establecidos en el paso de configuración (la clave maestra y un* $ID \in \{0,1\}^*$ *arbitrario). Utiliza el parámetro de seguridad del PKG para hacerlo.*

3. *Cifrado. El cifrado se realiza con una clave pública IBE que se calcula a partir del parámetro del paso de configuración y la identidad de un usuario.*

4. *Descifrado. Es el proceso inverso al cifrado. Utiliza la clave privada ya calculada para hallar el texto en claro.*

Hay que tener en cuenta que estos algoritmos deben satisfacer la restricción estándar de consistencia, esto es que el descifrado del cifrado de un mensaje debe coincidir con el mensaje original.

Un buen sistema criptográfico debe cumplir ciertos requisitos básicos: que se pueda cifrar de forma eficiente un mensaje, así como descifrarlo con conocimiento de la clave privada. También debe ser resistente al criptoanálisis. Hay cinco objetivos principales que cabe esperar de un criptosistema:

1. *Confidencialidad.* La confidencialidad mantiene en secreto la información de aquellos no autorizados para verla.

2. *Integridad.* La integridad asegura que la información no haya sido alterada por personas no autorizadas o medios desconocidos.

3. *Disponibilidad.* La disponibilidad asegura que el usuario pueda acceder a la información en el lugar y en el momento requerido, así como en la forma en la que el usuario lo necesita.

4. *Autenticación.* La autenticación es la capacidad de verificar la identidad de un usuario.

5. *No repudio.* Evita la negación de compromisos o acciones previas.

El uso de la criptografía puede respaldar la mayoría de estos objetivos, mientras que el de IBE se centra mayormente en uno de estos objetivos aunque también puede abarcar otros (por ejemplo se puede firmar con IBE). IBE proporciona una solución sencilla que garantiza la confidencialidad de los datos. No proporciona integridad, disponibilidad, autenticación y no repudio. Estos objetivos se alcanzan más fácilmente mediante firmas digitales utilizando claves creadas y gestionadas por un sistema tradicional de clave pública. Sin embargo, las ventajas que ofrece IBE lo convierten en una muy buena solución para algunos problemas.

La capacidad de calcular claves según sea necesario les da a los sistemas IBE diferentes propiedades con respecto a los sistemas tradicionales de clave pública y estas propiedades proporcionan ventajas prácticas en algunas situaciones. Entonces, aunque estos sistemas no añaden nuevas funcionalidades a los sistemas tradicionales de clave pública, las soluciones que utilizan IBE pueden ser mucho más simples de implementar y menos costosas de mantener.

Las implementaciones de los criptosistemas tradicionales de clave pública se han ganado la reputación de ser difíciles y costosos, pero esto se puede remediar mediante la utilización de criptosistemas IBE. Las soluciones creadas por un IBE parecen violar el principio de Geer de que el uso de cifrado debe tener un coste elevado.

La validación o verificación de claves para asegurarse de que una clave en particular sea válida en un momento concreto puede ser un proceso costoso y difícil, particularmente si se trata de una clave que se utilizó con anterioridad. Para evitar las dificultades prácticas, los sistemas IBE normalmente utilizan claves de corta duración y con un vencimiento establecido. Entonces, si asumimos que una clave IBE es válida solo por un día, no hay ninguna disposición para revocarla o suspenderla durante ese período. Una vez pasado el periodo de validez (un día en el ejemplo), será necesario obtener una nueva clave privada del PKG para seguir usando el sistema. De esta manera, se pierde la capacidad de revocar o suspender inmediatamente una clave, pero hace la validación de tales claves trivial.

Por otro lado, en cuanto a la recuperación de claves, la capacidad de restaurar una clave perdida o no disponible de otro modo es una característica esencial para la tecnología de cifrado de éxito comercial. En los sistemas tradicionales de clave pública se requieren un almacenamiento seguro de archivos con copias de todas las claves privadas y controlar cuidadosamente el acceso al archivo de estas claves. Sin embargo, los sistemas IBE calculan las claves según sea necesario, por lo que no hay necesidad de archivar las claves.

Como hemos visto, la capacidad de calcular claves públicas y privadas según sea necesario es una sutil diferencia entre los sistemas IBE y los tradicionales de clave pública, pero que proporciona muchas propiedades útiles.

1.3. La seguridad de los criptosistemas IBE: tamaño de las claves

La efectividad de la protección que aporta la criptografía, y por tanto qué tan segura es, depende de una variedad de elecciones, como son el tamaño de la clave criptográfica, el diseño del protocolo que sigue el criptosistema y la selección de las claves. Cada uno de estos temas es igualmente importante, pues si una clave es demasiado pequeña, si un protocolo está mal diseñado o se utiliza incorrectamente o si una clave está mal seleccionada o protegida, entonces la protección falla y puede dar acceso a terceros indeseados.

De entre todos las elecciones mencionados haremos hincapié en la importancia de una buena elección del tamaño de las claves (es decir, su número de bits), dado que las otras dos cuestiones (diseño del protocolo y la selección de las claves) dependen del tipo del criptosistema IBE que se trate.

Con el fin de dificultar la rotura del criptosistema (entendiendo por rotura el hecho de conseguir el texto en claro a partir del cifrado sin la ayuda de la clave privada), presentamos una guía sobre cuánto deben aumentarse los tamaños de las claves con el fin de mantener un margen de seguridad aceptable para aplicaciones de criptografía.
Las pautas seguidas están pensadas como límites inferiores en el sentido de que las claves de tamaños iguales o mayores que los tamaños recomendados alcanzan al menos un cierto nivel de seguridad especificado. Desde el punto de vista de la seguridad, es aceptable errar recomendando claves que pueden ser un poco más grandes de lo que realmente se requiere.

Todas las estimaciones de tiempo de ejecución se miden en MIPS-years. Un MIPS-year es la medida estándar de esfuerzo computacional en criptografía: la cantidad de cálculo realizado, en un año, por una computadora que opera a razón de un millón de operaciones por segundo (1 MIPS). Un MIPS-year equivale aproximadamente a 2^{45} operaciones. Usaremos la convención de que un año de computación en una PC equivale a 450 MIPS-years y escribiremos MMY por 1 millón de MIPS-years.

Los cuatro puntos sobre los que se basa la elección del tamaño de la clave criptográfica son los siguientes:

1. La vida útil, es decir, el plazo esperado en el que la información necesita ser protegida. Es el usuario el responsable de decidir hasta qué año debe de ser eficaz la protección aplicada. Esta puede corresponder con las medidas de seguridad populares: "corto plazo", "medio plazo" o "largo plazo". La decisión puede depender del valor de los datos a cifrar.
   En el cuadro 1 se sugieren tamaños de clave para criptosistemas asimétricos dependiendo de la vida útil esperada.

2. El margen de seguridad, que nos da la inviabilidad de un ataque exitoso.
   Un criptosistema se considera seguro si es lo suficientemente inviable para un ataque exitoso. Pero es difícil cuantificar lo que significa exactamente "suficientemente inviable". Por ello es mejor tomar un margen flexible en vez de uno fijo, que se ajuste mejor a cada problema.
   Definimos el margen de seguridad (s) como el año hasta el cual un usuario está dispuesto a confiar en el criptosistema.
   Una elección particular para s no significa que el criptosistema vaya a ser vulnerable desde el año s en adelante, sino que el usuario que eligió s está dispuesto a confiar en el criptosistema hasta el año s.
   Téngase en cuenta que esta definición de margen de seguridad también tiene en cuenta la probabilidad de éxito de ataques incompletos. De hecho, confiar en el criptosistema implica que es suficientemente resistente a todo tipo de atacantes potenciales.
   Nuestra configuración predeterminada será $s = 1982$, pues en este año un esfuerzo computacional de 0.5 MMY proporcionó un nivel de seguridad adecuado para las aplicaciones comerciales del DES (Data Encryption Standard) contra ataques de software. El DES es el criptosistema de clave simétrica más conocido, introducido en 1977 con un tamaño de clave de 56 bits.

3. El entorno informático, que tiene en cuenta el cambio esperado en los recursos computacionales de los que disponen los atacantes.
   Para estimar cómo puede cambiar con el tiempo la potencia de cálculo de la que disponen los atacantes, usamos la ley de Moore. Esta ley establece que la densidad de componentes por circuito integrado se duplica cada 18 meses. Podemos interpretarlo como que la potencia de cálculo por chip se duplica cada 18 meses.
   Una ligera variación de la ley de Moore, que depende menos de la tecnología, es la siguiente: definamos la variable $m > 0$ como el número de meses en promedio que se necesitan para que se duplique la aceleración del procesador y para un aumento del tamaño de la memoria. Tomaremos $m = 18$ de forma predeterminada.
   Definamos también la variable t, cuyos posibles valores son 0 y 1, como sigue:
   Si $t = 1$, la cantidad de potencia de cálculo y la memoria de acceso aleatorio (RAM) que se recibe por un dólar se espera que se dupliquen cada m meses.
   Si $t = 0$, se espera que la cantidad de potencia de cálculo y RAM se duplique cada m meses, independientemente del precio.
   De forma predeterminada tomaremos $t = 1$.
   Es decir, si $t = 0$, se supone que los recursos computacionales disponibles para los atacantes se duplican cada m meses, por lo que sus presupuestos no son inmediatamente relevantes. Y si $t = 1$ hay que tener en cuenta los aumentos presupuestarios y la inflación.
   Representaremos mediante la variable $b > 0$ el número de años, en promedio, tras los que se

espera que el presupuesto se duplique. Nuestra configuración predeterminada será $b = 10$, dado que el Producto Nacional Bruto de EE.UU. muestra una tendencia a duplicarse cada 10 años.

4. Los desarrollos esperados en criptoanálisis. Aunque es prácticamente imposible predecir que desarrollos criptoanalíticos tendrán lugar, se supone que el ritmo de los futuros hallazgos publicados y su impacto no variarán drásticamente en comparación con lo obtenido en estos últimos años. Definamos el número $r > 0$ como el número de meses (en promedio) que se espera que los desarrollos criptoanalíticos que afectan a los sistemas asimétricos sean dos veces más efectivos. Es decir, dentro de r meses se espera que atacar un mismo sistema asimétrico cueste la mitad de esfuerzo computacional que lo que cuesta hoy.
   Predeterminadamente tomaremos $r = 18$, pues corresponde con el progreso criptoanalítico que afecta a los sistemas asimétricos durante los últimos 25 años.

Como observación, las consideraciones de eficiencia y almacenamiento relativas a las claves criptográficas también pueden influir en la elección del tamaño de la clave, pero como no están directamente relacionados con la seguridad no se han discutido.

Los valores predeterminados tomados en esta sección [23] se deben a que los valores de los tamaños de las claves recomendados son computacionalmente equivalentes (en el sentido de que el esfuerzo computacional o el número de MIPS-years para un ataque exitoso es más o menos el mismo) y que ofrecen una seguridad al menos equivalente a la seguridad de 1982 del DES.
En la actualidad, el DES no se considera suficientemente seguro dado que en 1997 se recuperó con éxito una clave DES después de una búsqueda en Internet de aproximadamente 4 meses. La potencia informática necesaria para una búsqueda de claves exhaustiva de este tipo de software se estima en 0,5 MMY.

A continuación, procedemos a presentar las fórmulas de las que sen derivan cotas inferiores para los tamaños de las claves criptográficas. Para ello nos centramos en recomendaciones de tamaño de clave que se esperan que ofrezcan un margen de seguridad aceptable hasta un año determinado (y) dado por el usuario.

Sabiendo que se tardó $5 \cdot 10^5$ MIPS-years en romper DES, esta cantidad de cálculo ofreció un nivel aceptable de seguridad en el año s. Se deduce entonces que en el año y, es decir, y–s años después, una cantidad de cálculo de

$$IMY(y) = 5 \cdot 10^5 \cdot 2^{12(y-s)/m} \cdot 2^{t(y-s)/b}$$

MIPS-years ofrece un nivel de seguridad aceptable.
La notación $IMY(y)$ significa "número inviable de MIPS-years para el año y".
El factor $2^{12(y-s)/m}$ se debe a la aceleración esperada del procesador en el período del año s al año y, mientras que el factor $2^{t(y-s)/b}$ refleja el aumento esperado en el presupuesto disponible para un atacante. El valor resultante $IMY(y)$ se usa para derivar tamaños de clave que ofrecen un nivel de seguridad aceptable hasta el año y. Este no tiene en cuenta las posibles mejoras de los algoritmos criptográficos puesto que esto no proporcionaría un gran cambio.

Para los sistemas asimétricos usamos el tiempo de ejecución asintótico $L[n]$, pues es proporcional al tiempo que se requiere para factorizar una clave n. Este es una versión abreviada de

$$L[n, u, v] = e^{(v+o(1))\cdot \ln(n)^u \cdot \ln(\ln(n))^{1-u}}$$

donde se ha tomado $u = 1/3$ y $v = 1,9229$. El término $o(1)$ va a cero cuando n tiende al infinito, por lo que le omitiremos.

Esto combinado con el hecho de que en 1999 se rompió una clave de 512 bits con un coste de menos de 10^4 MIPS-years y que esperamos un progreso criptoanalítico en un factor de $2^{12(y-1999)/r}$ en comparación con el estado de la técnica en 1999, se deduce que si se elige el tamaño de clave asimétrica k tal que

$$\frac{L[2^k]}{IMY(y) \cdot 2^{12(y-1999)/r}} \geq \frac{L[2^{512}]}{10^4}$$

entonces la seguridad ofrecida por los sistemas asimétricos hasta el año y es al menos computacionalmente equivalente a la seguridad ofrecida por el DES en el año s.
Debido a que el punto de datos utilizado sobrestima ligeramente el coste de factorizar una clave de 512 bits y debido a que omitimos la o(1), la dificultad de romper los sistemas asimétricos está sobrestimado, es decir, los tamaños de clave asimétricos clásicos deben ser ligeramente más grande de lo que se indica en el cuadro 1.

Por último, para los años que van desde 1982 a 2050 y para la configuración predeterminada vista en este apartado, se recoge en el cuadro 1 el cálculo del tamaño recomendado de las claves resultante de las fórmulas dadas anteriormente.

Año	Tamaño clave en sistemas asimétricos	Inviable número de MIPS-years	Límite inferior para el coste de hardware en EEUU$ por un ataque de 1 día	Años en una PC Pentium II 450 MHz
1982	417	$5,00 \cdot 10^5$	$3,98 \cdot 10^7$	$1,11 \cdot 10^3$
1984	463	$1,45 \cdot 10^6$	$4,57 \cdot 10^7$	$3,22 \cdot 10^3$
1986	513	$4,19 \cdot 10^6$	$5,25 \cdot 10^7$	$9,31 \cdot 10^3$
1988	566	$1,21 \cdot 10^7$	$6,04 \cdot 10^7$	$2,69 \cdot 10^4$
1990	622	$3,51 \cdot 10^7$	$6,93 \cdot 10^7$	$7,80 \cdot 10^4$
1991	652	$5,97 \cdot 10^7$	$7,43 \cdot 10^7$	$1,33 \cdot 10^5$
1992	682	$1,02 \cdot 10^8$	$7,96 \cdot 10^7$	$2,26 \cdot 10^5$
1993	713	$1,73 \cdot 10^8$	$8,54 \cdot 10^7$	$3,84 \cdot 10^5$
1994	744	$2,94 \cdot 10^8$	$9,15 \cdot 10^7$	$6,53 \cdot 10^5$
1995	777	$5,00 \cdot 10^8$	$9,81 \cdot 10^7$	$1,11 \cdot 10^6$
1996	810	$8,51 \cdot 10^8$	$1,05 \cdot 10^8$	$1,89 \cdot 10^6$
1997	844	$1,45 \cdot 10^9$	$1,13 \cdot 10^8$	$3,22 \cdot 10^6$
1998	879	$2,46 \cdot 10^9$	$1,21 \cdot 10^8$	$5,48 \cdot 10^6$
1999	915	$4,19 \cdot 10^9$	$1,29 \cdot 10^8$	$9,31 \cdot 10^6$
2000	952	$7,13 \cdot 10^9$	$1,39 \cdot 10^8$	$1,58 \cdot 10^7$
2001	990	$1,21 \cdot 10^{10}$	$1,49 \cdot 10^8$	$2,70 \cdot 10^7$
2002	1028	$2,06 \cdot 10^{10}$	$1,59 \cdot 10^8$	$4,59 \cdot 10^7$
2003	1068	$3,51 \cdot 10^{10}$	$1,71 \cdot 10^8$	$7,80 \cdot 10^7$
2004	1108	$5,98 \cdot 10^{10}$	$1,83 \cdot 10^8$	$1,33 \cdot 10^8$
2005	1149	$1,02 \cdot 10^{11}$	$1,96 \cdot 10^8$	$2,26 \cdot 10^8$
2006	1191	$1,73 \cdot 10^{11}$	$2,10 \cdot 10^8$	$3,84 \cdot 10^8$
2007	1235	$2,94 \cdot 10^{11}$	$2,25 \cdot 10^8$	$6,54 \cdot 10^8$
2008	1279	$5,01 \cdot 10^{11}$	$2,41 \cdot 10^8$	$1,11 \cdot 10^9$
2009	1323	$8,52 \cdot 10^{11}$	$2,59 \cdot 10^8$	$1,89 \cdot 10^9$
2010	1369	$1,45 \cdot 10^{12}$	$2,77 \cdot 10^8$	$3,22 \cdot 10^9$
2011	1416	$2,47 \cdot 10^{12}$	$2,97 \cdot 10^8$	$5,48 \cdot 10^9$
2012	1464	$4,19 \cdot 10^{12}$	$3,19 \cdot 10^8$	$9,32 \cdot 10^9$
2013	1513	$7,14 \cdot 10^{12}$	$3,41 \cdot 10^8$	$1,59 \cdot 10^{10}$
2014	1562	$1,21 \cdot 10^{13}$	$3,66 \cdot 10^8$	$2,70 \cdot 10^{10}$
2015	1613	$2,07 \cdot 10^{13}$	$3,92 \cdot 10^8$	$4,59 \cdot 10^{10}$

Año	Tamaño clave en sistemas asimétricos	Inviable número de MIPS-years	Límite inferior para el coste de hardware en EEUU$ por un ataque de 1 día	Años en una PC Pentium II 450 MHz
2016	1664	$3,51 \cdot 10^{13}$	$4,20 \cdot 10^{8}$	$7,81 \cdot 10^{10}$
2017	1717	$5,98 \cdot 10^{13}$	$4,51 \cdot 10^{8}$	$1,33 \cdot 10^{11}$
2018	1771	$1,02 \cdot 10^{14}$	$4,83 \cdot 10^{8}$	$2,26 \cdot 10^{11}$
2019	1825	$1,73 \cdot 10^{14}$	$5,18 \cdot 10^{8}$	$3,85 \cdot 10^{11}$
2020	1881	$2,94 \cdot 10^{14}$	$5,55 \cdot 10^{8}$	$6,54 \cdot 10^{11}$
2021	1937	$5,01 \cdot 10^{14}$	$5,94 \cdot 10^{8}$	$1,11 \cdot 10^{12}$
2022	1995	$8,52 \cdot 10^{14}$	$6,37 \cdot 10^{8}$	$1,89 \cdot 10^{12}$
2023	2054	$1,45 \cdot 10^{15}$	$6,83 \cdot 10^{8}$	$3,22 \cdot 10^{12}$
2024	2113	$2,47 \cdot 10^{15}$	$7,32 \cdot 10^{8}$	$5,48 \cdot 10^{12}$
2025	2174	$4,20 \cdot 10^{15}$	$7,84 \cdot 10^{8}$	$9,33 \cdot 10^{12}$
2026	2236	$7,14 \cdot 10^{15}$	$8,41 \cdot 10^{8}$	$1,59 \cdot 10^{13}$
2027	2299	$1,21 \cdot 10^{16}$	$9,01 \cdot 10^{8}$	$2,70 \cdot 10^{13}$
2028	2362	$2,07 \cdot 10^{16}$	$9,66 \cdot 10^{8}$	$4,59 \cdot 10^{13}$
2029	2427	$3,52 \cdot 10^{16}$	$1,04 \cdot 10^{9}$	$7,81 \cdot 10^{13}$
2030	2493	$5,98 \cdot 10^{16}$	$1,11 \cdot 10^{9}$	$1,33 \cdot 10^{14}$
2032	2629	$1,73 \cdot 10^{17}$	$1,27 \cdot 10^{9}$	$3,85 \cdot 10^{14}$
2034	2768	$5,01 \cdot 10^{17}$	$1,46 \cdot 10^{9}$	$1,11 \cdot 10^{15}$
2036	2912	$1,45 \cdot 10^{18}$	$1,68 \cdot 10^{9}$	$3,22 \cdot 10^{15}$
2038	3061	$4,20 \cdot 10^{18}$	$1,93 \cdot 10^{9}$	$9,33 \cdot 10^{15}$
2040	3214	$1,22 \cdot 10^{19}$	$2,22 \cdot 10^{9}$	$2,70 \cdot 10^{16}$
2042	3371	$3,52 \cdot 10^{19}$	$2,55 \cdot 10^{9}$	$7,82 \cdot 10^{16}$
2044	3533	$1,02 \cdot 10^{20}$	$2,93 \cdot 10^{9}$	$2,26 \cdot 10^{17}$
2046	3700	$2,95 \cdot 10^{20}$	$3,36 \cdot 10^{9}$	$6,55 \cdot 10^{17}$
2048	3871	$8,53 \cdot 10^{20}$	$3,86 \cdot 10^{9}$	$1,90 \cdot 10^{18}$
2050	4047	$2,47 \cdot 10^{21}$	$4,44 \cdot 10^{9}$	$5,49 \cdot 10^{18}$

Cuadro 1: Límites inferiores para tamaños de clave computacionalmente equivalentes, suponiendo que $s = 1982$, $m = 18$, $t = 1$, $b = 10$, $r = 18$ (obtenido de [23]).

La tabla se puede usar de la siguiente manera.
Supongamos que estamos desarrollando una aplicación comercial en el año 2000 y que debe garantizarse la confidencialidad o integridad de la información electrónica durante 20 años, es decir, hasta el año 2020. Mirando la fila para el año 2020 en el cuadro 1 vemos que una cantidad de computación de $2,94 \cdot 10^{14}$ MIPS-years en el año 2020 puede considerarse tan inviable como lo fue $5 \cdot 10^{5}$ MIPS-years en 1982.
Una seguridad computacionalmente equivalente a la ofrecida por el DES en 1982 se obtiene utilizando en el año 2020 al menos 1881 bits.
Del cuadro 1 se obtiene que hasta el año 2002 se puede esperar que las claves de 1024 bits ofrezcan una seguridad computacionalmente equivalente al DES en 1982.

1.4. Función hash criptográfica

Siguiendo en la línea de la seguridad criptográfica, también es importante que tengamos en cuenta la protección de las claves, que se debe evitar guardarlas en una base de datos mediante texto claro. Para ello, se suelen proteger mediante una función hash.
Si algún tercero no autorizado fuese capaz de vulnerar un servicio y robar su base de datos, si las claves no estuvieran hasheadas, sus credenciales se verían expuestas.

La idea que hay detrás de la función hash es la de evitar aquellos casos particulares menos costosos de ciertos problemas, mediante su transformación aparentemente aleatoria a otros casos mas generales, y por tanto mas difíciles, del mismo problema.

Una función criptográfica hash es un algoritmo matemático que transforma cualquier entrada de datos en una nueva serie de caracteres con una longitud fija (llamado valor hash o hash). La longitud de un hash es invariable para cualquier dato de entrada.

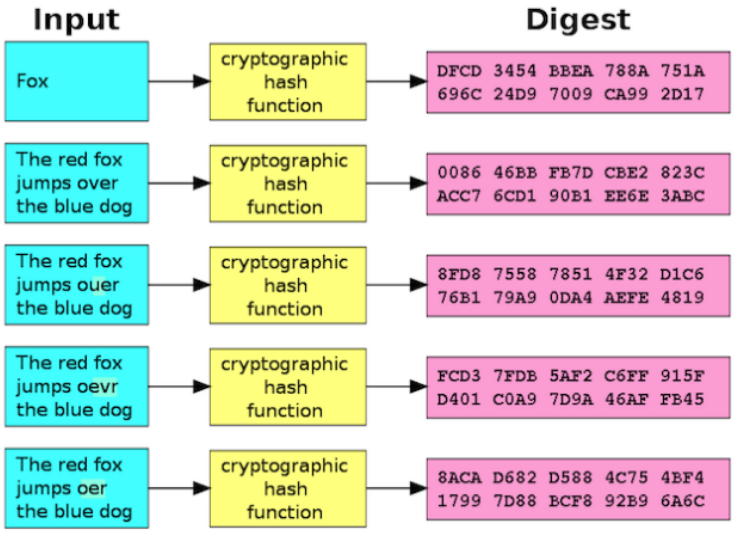

Ilustración 4: Ejemplo de función hash con entrada de texto y con salida en hexadecimal. [14]

La salida de la función hash se denomina resumen del mensaje de entrada. En general, la seguridad de una función hash está relacionada con la longitud del resumen y con la forma de definir la función hash.
Tampoco debería ser prácticamente factible encontrar un segundo dato cuyo hash sea el mismo que el del mensaje original (resistencia débil a la colisión).
Un ataque de colisión significa que se encuentran dos entradas diferentes que producen el mismo resultado de un hash particular. El hecho de que se encuentre una colisión no implica necesariamente que un algoritmo se haya roto.
Finalmente, no debería ser factible encontrar dos mensajes arbitrarios con el mismo hash (fuerte resistencia a colisiones).

Entonces, toda función hash debe ofrecer ciertos niveles de seguridad para garantizar la integridad del mensaje y que pueda ser empleada en criptografía, por lo que debe cumplir con las siguientes propiedades:

1. *Facilidad de cálculo:* debe ser computacionalmente fácil calcular el valor hash a partir de una entrada.
2. *Unidireccionalidad:* debe de ser computacionalmente difícil derivar los datos de entrada originales dada solo la salida de la función hash. Es decir, que no se pueda (o sea difícil) calcular la función inversa de la función hash o lo que es lo mismo sea una función de una vía.

   Definición 1.4.1 *Una función $f : X \to Y$ es de una vía cuando dados $x \in X, y \in Y$ es sencillo calcular $f(x)$ pero es computacionalmente difícil calcular $f^{-1}(y)$.*

3. *Compresión:* a partir de una entrada de datos de cualquier longitud, el resumen debe tener una longitud fija. Normalmente esta longitud fija es menor que la de la entrada.

4. *Difusión:* el resumen debe ser una función compleja de todos los bits de la entrada. Es decir, si tomamos dos entradas diferentes las cuales difieren en un solo bit, sus valores hash deberían cambiar en aproximadamente la mitad de sus bits.
5. *Colisión simple:* será computacionalmente difícil conocida una entrada x encontrar otra entrada y tal que $h(x) = h(y)$, siendo h la función hash. Esto se conoce como resistencia débil a las colisiones.
6. *Colisión fuerte:* será computacionalmente difícil encontrar un par de entradas x e y distintas de forma que $h(x) = h(y)$ siendo h la función hash. Esto se conoce como resistencia fuerte a las colisiones.

De las funciones hash nacieron algoritmos de seguridad con el objetivo de comparar y encontrar las colisiones. Existen varios modelos de seguridad basados en funciones hash, unos más conocidos que otros y algunos que se han dejado de usar pues su seguridad se ha visto afectada tras el hallazgo de colisiones en sus sistemas. Un ejemplo de esto último es el SHA-1, uno de los más utilizados hasta que Google encontró una colisión a la hora de usar dos PDF con la misma firma.
Veamos los algoritmos más nombrados:

- **MD5 (Message Digest Algorithm)** (valor hash de 128 bits). Fue desarrollado por Ron Rivest en 1992. Se utiliza ampliamente, se ha empleado en una amplia variedad de aplicaciones de seguridad y también se usa comúnmente para verificar la integridad de los archivos. Un hash MD5 generalmente se expresa como un número hexadecimal de 32 dígitos.
  Aunque está obsoleto desde 2005, su algoritmo constituye la base de otras funciones. MD5 es una mejora del MD4 y MD2 (1990), más lento pero con mayor nivel de seguridad. Además, MD5 se utilizó como modelo para SHA-1, ya que comparten muchas características comunes. MD5 y SHA-1 son los dos algoritmos hash más utilizados en la actualidad, aunque el uso de MD5 ha disminuido ya que ahora se considera roto.

- **SHA-1 (Secure Hash Algorithm)** (valor hash de 160 bits). Fue desarrollado por el NIST (National Institute of Standards and Technology) en 1995. SHA y sus derivadas se consideran las funciones hash más seguras hasta el momento. Aunque es considerado lo suficientemente seguro para aplicaciones prácticas, hay disponibles versiones más robustas que la reemplazarán como los de la familia del SHA-2.

- **SHA-256** (valor hash de 256 bits). Pertenece a la familia del SHA-2, y se suele usar en la autenticación de software y para la firma de mensajes. El SHA-2 tiene cuatro variantes según el número de bits de salida: SHA2-224, SHA2-256, SHA2-384 y SHA2-512.
  El algoritmo SHA2-256 es uno de los más usados gracias a su equilibrio entre seguridad y velocidad. Es un algoritmo muy eficiente y tiene una alta resistencia a colisiones. Por ejemplo, el método de verificar los Bitcoins está basado en SHA2-256.

- **RIPEMD** (valor hash de 128 bits). Fue creada por el proyecto europeo RIPE en el año de 1992. Su principal función era la de sustituir al estándar del momento, la función hash MD4. En la actualidad aún se considera muy seguro, especialmente en sus versiones RIPEMD-160 (con un valor hash de 160 bits). Se usa en el estándar de Bitcoin.

En el sistema mostrado en este trabajo, he utilizado la función hash SHA-256. Para ello, mediante una implementación en python, he utilizado la función “SHA256” del paquete “Hash”, el cual se encuentra a su vez dentro del paquete “Crypto”. Su funcionamiento esta descrito en [7].

1.5. Los sistemas IBE en la actualidad

Hasta ahora hemos visto los sistemas IBE como procedimientos que tienen como principal objetivo el intercambio de información entre dos usuarios de manera confidencial y segura. Pero, además de este uso, han encontrado otras aplicaciones tanto a nivel teórico como práctico.
Su uso original era el de proteger la confidencialidad de informaciones militares y políticas, pero en la actualidad es una ciencia interesante no solo en esos aspectos sino para cualquiera que esté interesado en la confidencialidad de unos determinados datos.

En general, estos sistemas tienen aplicabilidad en cualquier escenario en que una misma persona o entidad posee una cierta información que guarda cifrada (por ejemplo en la nube) y quiere permitir que diferentes usuarios tengan acceso a diferentes partes de esa información. Algunos ejemplos concretos (obtenidos de [18]) son:

- Con el fin de que diferentes dispositivos tengan acceso a diferentes partes de la información, un usuario puede cifrar toda su información en la nube usando diferentes identidades, cada una para diferentes partes de la información en función de los dispositivos que cree que van a tener que acceder a esos datos en el futuro.
- Una plataforma de venta y distribución de contenido digital (series, películas...) puede tener todo su contenido cifrado en la nube, bajo diferentes identidades. Por ejemplo, una misma película puede estar cifrada dos veces con las identidades “películas” y “total”, de manera que cuando un usuario se suscribe a la plataforma, en función de su tipo de suscripción se le asignan diferentes claves secretas de usuario. Si paga una cuota para ver solo películas y series, recibirá dos claves secretas, para “películas” y “series”. Si un usuario paga la cuota máxima, recibirá la clave secreta para la identidad “total”.
- Supongamos que una persona o entidad consigue de alguna manera una gran cantidad de información sensible y comprometedora para uno o varios países. Decide compartir esa información con todo el mundo, pero de una manera gradual: divide la información en paquetes, y decide que revelará un paquete cada mes. Como sospecha que los servicios secretos de los países implicados están intentando hacerse con esa información, lo que hace es cifrar toda la información y publicarla en Internet; cifra cada bloque con una identidad temporal, del tipo “octubre 2020”. Lo único que tiene que guardar secretamente esta persona es la clave secreta de máster. Así, cuando llegue cada mes (por ejemplo octubre de 2020) usará la clave secreta de máster para calcular la clave secreta correspondiente a id =“octubre 2020” y la compartirá con todos los medios que quiera.

Hoy en día, todos los intentos de diseñar sistemas IBE seguros y razonablemente eficientes (en particular, en los que los parámetros del sistema no dependan del número de posibles identidades) basados en herramientas criptográficas de clave pública clásica (como RSA o criptografía basada en la dificultad del logaritmo discreto en grupos cíclicos) han sido infructuosos.

Diseñar esquemas de cifrado basado en identidades seguros es una tarea difícil. Es más, existen algunos resultados negativos sobre la utilización de algunas técnicas a la hora de diseñar sistemas IBE. Para sortear estos resultados negativos, una posibilidad es la de relajar alguno de los requisitos del sistema que se quiere diseñar, por tanto ese sistema IBE no tendrá todas las propiedades posibles, pero si las suficientes para algunas aplicaciones reales interesantes.

Actualmente los únicos esquemas IBE eficientes y seguros utilizan emparejamientos bilineales (*bilinear pairings*), una herramienta algebraica definida en ciertas curvas elípticas.

2 Conceptos y Propiedades Utilizados

2.1. Residuosidad Cuadrática

Definición 2.1.1 (Número primo) *Un número entero $p \geq 2$ es primo si sus únicos divisores positivos son 1 y p.*

Definición 2.1.2 (Máximo común divisor) *El máximo común divisor de dos enteros a y b, no ambos nulos, es el mayor entero positivo que divide a ambos. Se escribe $mcd(a, b)$.*

Definición 2.1.3 (Identidad de Bézout) *Si a y b son dos enteros, no ambos nulos, existen dos números $r, s \in \mathbb{Z}$ tales que $ar + bs = mcd(a, b)$.*

Podemos calcular el máximo común divisor de dos enteros mediante el siguiente algoritmo.

Definición 2.1.4 (Algoritmo de Euclides) *Sean $a, b \in \mathbb{Z}$ tales que $a > b$, y sea $d = mcd(a, b)$. El algoritmo calcula sucesivamente un cociente q_i y un resto r_i, de manera que*

$$a = b \cdot q_1 + r_1 \text{ donde } q_1, r_1 \in \mathbb{Z} \text{ con } 0 \leq r_1 < b$$

$$b = r_1 \cdot q_2 + r_2 \text{ donde } q_2, r_2 \in \mathbb{Z} \text{ con } 0 \leq r_2 < r_1$$

$$r_1 = r_2 \cdot q_3 + r_3 \text{ donde } q_3, r_3 \in \mathbb{Z} \text{ con } 0 \leq r_3 < r_2$$

$$\dots$$

$$r_{t-1} = r_t \cdot q_{t+1} + r_{t+1} \text{ donde } q_{t+1}, r_{t+1} \in \mathbb{Z} \text{ con } 0 \leq r_{t+1} < r_t$$

$$r_t = r_{t+1} \cdot q_{t+2} \text{ con } q_{t+2} \in \mathbb{Z}$$

Cuando se obtiene una división exacta (resto cero), el máximo común divisor buscado es el resto anterior: r_{t+1}. Además, con este algoritmo se puede calcular la identidad de Bézout, simplemente aplicando las igualdades anteriores:

$$d = r_{t+1} = r_{t-1} - r_t \cdot q_{t+1} = \dots = a \cdot r + b \cdot s \text{ con } r, s \in \mathbb{Z}.$$

Definición 2.1.5 (Números coprimos) *Dos números enteros a y b son coprimos (primos entre sí) si se cumple $mcd(a, b) = 1$.*

Definición 2.1.6 (Congruencia) *Sean a, b y n números enteros. Se dice que a es congruente con b módulo n si n divide a $(b - a)$. Es decir, si a y b tienen el mismo resto al ser divididos por n. Se escribe: $a \equiv b \mod n$.*

Definición 2.1.7 (Teorema chino de los restos) *Sean $n_1, n_2, \dots, n_k$ enteros coprimos dos a dos y sean $a_1, a_2, \dots, a_k$ enteros. Entonces, el siguiente sistema de congruencias tiene solución única módulo $n = n_1 \cdot n_2 \cdot \dots \cdot n_k$.*

$$\begin{cases} x \equiv a_1 \mod n_1 \\ x \equiv a_2 \mod n_2 \\ \dots \\ x \equiv a_k \mod n_k \end{cases}$$

Definición 2.1.8 (Algoritmo de Gauss) *La solución del sistema de congruencias dado en el teorema chino de los restos se puede escribir del siguiente modo:*

$$x \equiv \sum_{i=1}^{k} a_i N_i M_i \mod n$$

donde $N_i = \frac{n}{n_i}$ *y* $M_i = N_i^{-1} \mod n_i$.

El algoritmo de Gauss se puede escribir también de una manera ligeramente diferente:

$$x \equiv \sum_{i=1}^{k} a_i \cdot e_i \mod n$$

donde cada e_i cumple

$$e_i = \begin{cases} 1 \mod n_i \\ 0 \mod n_j \ (j \neq i) \end{cases}$$

Así, aplicando esta última modificación del algoritmo de Gauss tendríamos el siguiente ejemplo:

Partimos del siguiente sistema de congruencias:

$$\begin{cases} x \equiv 2 \mod 3 \\ x \equiv 3 \mod 4 \end{cases}$$

Aplicando el algoritmo de Gauss, la solución al sistema es:

$$x \equiv (2 \cdot e_1 + 3 \cdot e_2) \mod (3 \cdot 4) \equiv (2 \cdot e_1 + 3 \cdot e_2) \mod 12$$

Encontremos entonces dos enteros e_1 y e_2 tale que:

$$e_1 = \begin{cases} 1 \mod 3 \\ 0 \mod 4 \end{cases} \quad y \quad e_2 = \begin{cases} 0 \mod 3 \\ 1 \mod 4 \end{cases}$$

Por ejemplo, podríamos tomar $e_1 = 4$ y $e_2 = 9$, obteniendo:

$$x = (2 \cdot 4 + 3 \cdot 9) = 35 \equiv 11 \mod 12$$

Definición 2.1.9 (Función de Euler) *Dado un entero positivo n, su función de Euler* $\phi(n)$ *denota el número de enteros positivos menores que n que son coprimos con n.*

Algunas de las propiedades de la función de Euler son las siguientes:

- Si p es primo entonces $\phi(p) = p - 1$.
- Si n y m son primos entre sí, entonces $\phi(n \cdot m) = \phi(n) \cdot \phi(m)$.
- Si p es primo y k es un entero positivo, entonces $\phi(p^k) = (p-1)p^{k-1}$.

Teorema 2.1.1 (Pequeño teorema de Fermat) *Sea p un número primo y a un entero cualquiera. Entonces,* $a^p \equiv a \mod p$.
Si, además, a es coprimo con p, se tiene que $a^{p-1} \equiv 1 \mod p$.

Teorema 2.1.2 (Teorema de Euler) *Sean n un entero positivo y a un entero coprimo con n. Entonces,* $a^{\phi(n)} \equiv 1 \mod n$.

Definición 2.1.10 (Anillo de restos) *Usaremos $\mathbb{Z}_n$ para denotar el conjunto de números $\{0, 1, 2, \ldots, n-2, n-1\}$.*
Podemos realizar aritmética en elementos de $\mathbb{Z}_n$ reduciendo una suma o un producto al resto obtenido de dividir por n. Esto se llama reducción módulo n. Así $a + b = c$ cuando $(a + b) \equiv c \mod n$.

Definición 2.1.11 (Residuo cuadrático) *Un elemento $a \in \mathbb{Z}_n$ no nulo se llama residuo cuadrático módulo n si existe algún $x \in \mathbb{Z}_n$ tal que $x^2 \equiv a \mod n$.*

La función siguiente determina cuándo un número es residuo cuadrático.

Definición 2.1.12 (Símbolo de Legendre) *Sea p un primo impar y sea $a \in \mathbb{N}$. El símbolo de Legendre $\left(\frac{a}{p}\right)$ se define como:*

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & \text{si } p \text{ divide a } a \\ 1, & \text{si } a \text{ es un residuo cuadratico módulo } p \\ -1, & \text{si } a \text{ no es residuo cuadratico módulo } p \end{cases}$$

A continuación, recogemos algunas propiedades de esta función.
Sean a y b números enteros y sean p y q primos impares.

- $\left(\frac{a}{p}\right) = a^{\phi(p)/2} \mod p = a^{(p-1)/2} \mod p$
- $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$
- Si $a = b \mod p$, se tiene $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$
- $\left(\frac{2}{p}\right) = (-1)^{(p^2-1)/8}$
- $\left(\frac{p}{q}\right) = \left(\frac{q}{p}\right)(-1)^{(p-1)(q-1)/4}$ **(Ley de reciprocidad cuadrática)**

Cabe observar que el símbolo de Legendre $\left(\frac{a}{p}\right)$ solo está definido si p es primo. La siguiente definición amplía la función al caso de módulo compuesto.

Definición 2.1.13 (Símbolo de Jacobi) *Sean $a, n \in \mathbb{N}$ con $a, n \geq 1$, y n impar. Si la descomposición en factores primos de n es $n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_r^{e_r}$ con $p_1, p_2, \ldots, p_r$ números primos y $e_1, e_2, \ldots, e_r \in \mathbb{N}$, el símbolo de Jacobi $\left(\frac{a}{n}\right)$ se define como:*

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{e_1}\left(\frac{a}{p_2}\right)^{e_2}\cdots\left(\frac{a}{p_r}\right)^{e_r},$$

donde cada factor $\left(\frac{a}{p_i}\right)^{e_i}$ es una potencia de un símbolo de Legendre.

Veamos algunas de sus propiedades:
Sean a y b números enteros y sean n y m enteros impares mayores que 2.

- $\left(\frac{a}{n}\right)$ puede valer 0, 1 o -1.
- Si $\text{mcd}(a, n) \neq 1$, se tiene $\left(\frac{a}{n}\right) = 0$
- $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$
- $\left(\frac{a}{mn}\right) = \left(\frac{a}{m}\right)\left(\frac{a}{n}\right)$
- Si $a = b \mod n$, se tiene $\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$

- $\left(\frac{1}{n}\right) = 1$
- $\left(\frac{-1}{n}\right) = (-1)^{(n-1)/2}$
- $\left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8}$
- $\left(\frac{m}{n}\right) = \left(\frac{n}{m}\right)(-1)^{(m-1)(n-1)/4}$ **(Ley de reciprocidad cuadrática)**

Así, por ejemplo, si queremos calcular el símbolo de Jacobi para un módulo compuesto del que no sabemos su factorización en primos, podemos recurrir a la ley de reciprocidad cuadrática.

En un caso concreto:

$$\left(\frac{21}{93}\right) = \left(\frac{93}{21}\right) \cdot (-1)^{92\cdot 20/4} = \left(\frac{93}{21}\right) \cdot (-1)^{460} = \left(\frac{93}{21}\right) = \left(\frac{9}{21}\right) = \left(\frac{21}{9}\right) \cdot (-1)^{20\cdot 8/4} =$$

$$= \left(\frac{21}{9}\right) \cdot (-1)^{40} = \left(\frac{21}{9}\right) = \left(\frac{3}{9}\right) = \left(\frac{9}{3}\right) \cdot (-1)^{8\cdot 2/4} = \left(\frac{9}{3}\right) \cdot (-1)^4 = \left(\frac{9}{3}\right) = 0$$

Cabe destacar que, al aplicar la ley de reciprocidad cuadrática como en el ejemplo anterior, en realidad no hace falta realizar el cálculo del exponente $(m-1)(n-1)/4$, lo que supondría un esfuerzo computacional innecesario. Bastaría con tener en cuenta si 4 es divisor de $m-1$ y de $n-1$, es decir, saber la paridad de $(m-1)/2$ y de $(n-1)/2$ (notemos que $m-1$ y $n-1$ siempre serán pares dado que m y n son impares).
Así, el exponente de -1 es impar si y solo si son impares sus dos factores $(m-1)/2$ y $(n-1)/2$, lo que, a su vez, equivale a que ni m ni n sean divisibles entre 4. En otro caso, el signo no cambia.
Siguiendo esta idea, hemos implementado una función en Python encargada de calcular el símbolo de Jacobi utilizando la ley de reciprocidad cuadrática. Esta se encuentra en el archivo "cocks.py" expuesto en el anexo de esta memoria.

Sea n el producto de dos primos impares distintos. Sea $\mathbb{Z}_n^* = \{x \in \mathbb{Z} : 1 \leq x \leq n \, y \, \text{mcd}(x,n) = 1\}$, el grupo (multiplicativo) de las unidades módulo n. Tenemos el siguiente teorema.

Teorema 2.1.3 *Sean p y q dos números primos distintos y sea $n = p \cdot q$. Entonces $a \in \mathbb{Z}_n^*$ es un residuo cuadrático módulo n si solo si a es un residuo cuadrático módulo p y y también módulo q.*

Exactamente la mitad de los elementos de $\mathbb{Z}_n^*$ tienen símbolo de Jacobi 1, y la otra mitad símbolo de Jacobi -1. Denotemos a los primeros por $\mathbb{Z}_n^*(+1)$ y a los segundos por $\mathbb{Z}_n^*(-1)$.
Se sabe que ninguno de los elementos de $\mathbb{Z}_n^*(-1)$ y exactamente la mitad de los elementos de $\mathbb{Z}_n^*(+1)$ son residuos cuadráticos.

Definición 2.1.14 (Problema de la residuosidad cuadrática) *Sea n el producto de dos primos impares distintos. El problema de la residuosidad cuadrática consiste en decidir si x es residuo cuadrático o no para $x \in \mathbb{Z}_n^*(+1)$.*

2.2. Codificaciones de caracteres

Dada las operaciones de división y multiplicación en binario desde el lado de la programación (cuya explicación se puede encontrar en el anexo B), son operaciones que consumen bastante tiempo de procesamiento, por lo tanto, cuando se realizan varios cálculos que conllevan multiplicación y división es más factible realizar cálculos con desplazamiento hacia la izquierda o derecha.

Definición 2.2.1 (Operador desplazamiento) *Los operadores de desplazamiento a izquierda (representado por ($\ll$) o a derecha ($\gg$) mueven el primer operando α según el número de posiciones que especifica el segundo operando β: $\alpha \ll \beta$ y $\alpha \gg \beta$. Ambos operandos, α y β, deben ser valores enteros. Para los desplazamientos hacia la izquierda, los bits de la derecha desocupados se establecen en 0. Para los desplazamientos hacia la derecha, los bits de la izquierda desocupados se rellenan según el tipo del primer operando después de la conversión.*
De este modo, la instrucción $\alpha \ll \beta$ es equivalente a multiplicar α por 2^β; y la instrucción $\alpha \gg \beta$ es equivalente a dividir α por 2^β.
Se debe de tener en cuenta que el resultado de una operación de desplazamiento es indefinido si el segundo operando es negativo o si el operando derecho es mayor o igual a la anchura en bits del operando izquierdo promovido.

Así el desplazamiento a la derecha ($\gg$) de números en el sistema binario equivaldría a una división, mientras que un desplazamiento a la izquierda ($\ll$) equivale a una multiplicación.

El sistema de base 2 es utilizado para almacenar y manejar toda la información dentro de una computadora, que maneja una gran cantidad de ceros y unos utilizados para representar un único símbolo, y más aun para una cadena de símbolos (como podría ser una palabra o frase). Tengamos en cuenta que, si queremos representar los números del sistema decimal en el sistema binario, harían falta n bits para representar los primeros 2^n números del sistema decimal. Es decir, para representar los números del 0 al $2^n - 1$ se usarán como máximo n bits. Así, por ejemplo, el numero 9 ($< 2^4$) se escribe como 1001 en el sistema binario, utilizando 4 bits.

Por la dificultad de manejar los ceros y unos en grandes cantidades, se formuló un conjunto de codificaciones de caracteres y regulaciones unificadas sobre la relación entre los caracteres ingleses y los dígitos binarios. A esto se le llamo código ASCII (American Standard Code for Information Interchange) y se ha utilizado hasta el día de hoy. Dado que los ordenadores solo pueden entender los números, el código ASCII proporciona una representación numérica de un carácter como “a” o “@” o una acción de algún tipo.
Originalmente, el código ASCII especifica un total de 128 caracteres (incluidos 32 símbolos de control no imprimibles), los cuales solo ocupan los últimos 7 bits de un byte (1 byte = 8 bits), y el primer bit se especifica uniformemente como 0. A continuación se observa una tabla con los caracteres que contiene (ilustración 5).

Por ejemplo, según la tabla tendríamos que el espacio “space” es 32 (binario 00100000) y la letra mayúscula “A” es 65 (binario 01000001).

Sin embargo, esta codificación presenta un problema a nivel mundial, pues no todos los símbolos usados en los diferentes países se encuentran en ella. Como solución se propuso añadir nuevos símbolos al código ASCII dado que este podía abarcar 128 caracteres más (pues con 8 bits se pueden representar $2^8 = 256$ caracteres diferentes). Pero los diferentes países tienen letras diferentes, por lo que incluso si todos usan 256 símbolos para codificar, las letras que representan son diferentes. Por ejemplo, 130 representa “é” en la codificación francesa, pero la letra Gimel (ג) en la codificación hebrea y otro símbolo en la codificación rusa. Aunque esto no pasaría con los primeros 128 caracteres.
Esto conllevó a crear un nuevo código, el Unicode.
Unicode es el estándar de codificación de caracteres universal utilizado para la representación de texto para procesamiento del equipo. Proporciona una manera consistente de codificación de texto multilingüe y facilita el intercambio de archivos de texto internacionales. Su tamaño actual puede acomodar más de 1 millón de símbolos.

Existen múltiples métodos de almacenamiento para Unicode, es decir, hay muchos formatos binarios diferentes que se pueden usar para representar Unicode. UTF-8 es la implementación de Unicode

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Ilustración 5: Tabla ASCII. [8]

más utilizada en Internet. La característica más importante de UTF-8 es que es un método de codificación de longitud variable. Puede usar 1 4 bytes para representar un símbolo y cambiar la longitud del byte de acuerdo con diferentes símbolos (recordemos 1 byte = 8 bits). Para esta codificación hay que seguir dos reglas:

1. Para los símbolos de un solo byte, el primer bit del byte se establece en 0 y los siguientes 7 bits son el código Unicode del símbolo. Por lo tanto, para letras en inglés, los códigos UTF-8 y ASCII son los mismos.

2. Para los símbolos de n bytes (con $n > 1$), se establece lo siguiente: en el primer byte, los primeros n bits se establecen en 1, y el bit $n+1$ se establece en 0; y en los bytes posteriores, los primeros dos bits se establecen en 10. Los bits binarios restantes no mencionados son todo el código Unicode de este símbolo.

En la ilustración 6 se muestra una tabla que resume las reglas de codificación. En ella, la letra x indica los bits de codificación disponibles.

Según la tabla anterior, es muy simple interpretar la codificación UTF-8. Si el primer bit de un byte es 0, entonces el byte es solo un carácter; si el primer bit es 1, el número de unos consecutivos indica cuántos bytes ocupa el carácter actual.
La ventaja de este formato es que es compatible con versiones anteriores de esquemas de codificación ASCII.

2.3. Complejidad Computacional

El análisis del coste de computación de un algoritmo puede realizarse mediante el estudio de la cantidad de recursos (tiempo de ejecución y ocupación de memoria) necesarios para que el algoritmo

Método de codificación UTF-8 (binario)	descripción
0xxxxxxx	7 bytes de un byte disponibles
110xxxxx 10xxxxxx	2 bytes 11 bits disponibles
1110xxxx 10xxxxxx 10xxxxxx	3 bytes 16 bits disponibles
11110xxx 10xxxxxx 10xxxxxx 10xxxxxx	4 bytes 21 están disponibles

Ilustración 6: Tabla sobre la codificación UTF-8. [35]

se ejecute y termine dando el resultado buscado.

Definición 2.3.1 (Algoritmo) *Un algoritmo es un conjunto reglas que dada una entrada generan una secuencia de instrucciones ordenadas y finitas que permite resolver un problema mediante pasos sucesivos que no generen dilema.*

Destacamos tres características de los algoritmos:

- Precisión: las instrucciones de un algoritmo deben darse sin ningún tipo de ambigüedad.
- Determinismo: un algoritmo debe responder del mismo modo antes las mismas condiciones, es decir, para mismas variables mismos resultados.
- Finitud: las instrucciones que definen al algoritmo deben ser finitas.

Definición 2.3.2 (Algoritmo determinista) *Un algoritmo determinista es un algoritmo completamente predecible si se conocen sus entradas. Es decir, partiendo de la misma entrada del algoritmo siempre producirá la misma salida, y la máquina pasará por la misma secuencia de estados internos.*

Definición 2.3.3 (Algoritmo probabilista) *Un algoritmo probabilista es un algoritmo que basa su resultado en la toma de algunas decisiones al azar, de tal forma que, en promedio, obtiene una buena solución al problema planteado para cualquier distribución de los datos de entrada. Es decir, a partir de unos mismos datos se pueden obtener distintas soluciones y, en algunos casos, soluciones erróneas. Esto se debe a la aleatoriedad que da el algoritmo a algunos de los datos de la entrada.*

La información que nos indica la eficiencia de un algoritmo es el tiempo requerido para que este se ejecute completamente. Un modo de medir este valor, de forma que no dependa del ordenador utilizado, sino solo de la propia definición del algoritmo, es medir el número de operaciones a nivel de bit. Calculando este número de operaciones necesarias para la ejecución de un algoritmo, obtenemos la complejidad bit de este algoritmo.
Dado un algoritmo, su complejidad nos da la idea de cómo se va a comportar a medida que incrementemos de forma indefinidamente el tamaño de la entrada (es decir, en su comportamiento asintótico).

Definición 2.3.4 (Complejidad computacional) *Es el estudio de la eficacia (y/o número de operaciones) de un algoritmo. Formalmente, la complejidad de un algoritmo (en tiempo) es una función* $f : \mathbb{N} \to \mathbb{N}$ *donde* $f(n) = max\{$*número de operaciones básicas para calcular el output a partir de un input de tamaño* $n\}$.

Dada una función f, a veces nos interesa estudiar aquellas funciones g tales que a lo sumo crecen tan deprisa como f. Para ello introduzcamos la siguiente notación.

Definición 2.3.5 (Cota superior asintótica. Notación O de Landau) *Sean $f, g : \mathbb{N} \to \mathbb{R}^+$, se dice que $f(n) \in O(g(n))$ si existen $n_0 \in \mathbb{N}$ y $c > 0$ tales que $f(n) \leq c \cdot g(n), \forall n \geq n_0$. Equivalentemente, si $\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c$.*

Algunas de las propiedades de la notación O de Landau son las siguientes:

- $O(f) + O(g) \in O(f + g) = O(\text{máx}\{f, g\})$
- $O(f) \cdot O(g) \in O(f \cdot g)$
- $O(k \cdot f) \in O(g)$, para cualquier entero $k \geq 0$

Al conjunto de las funciones que crecen a lo sumo tan deprisa como una función f se le llama la cota superior de f y lo denominamos $O(f)$. Conociendo la cota superior asintótica de un algoritmo podemos asegurar que, en ningún caso, el tiempo empleado será de un orden superior al de la cota.

El comportamiento asintótico de un algoritmo y su valor recogido mediante la notación O de Landau puede determinar si un algoritmo es factible o no factible.

Definición 2.3.6 (Algoritmo polinómico) *Un algoritmo se dice polinómico si su tiempo de ejecución está limitado por una expresión polinómica en el tamaño de la entrada del algoritmo. Es decir, si el número de operaciones del algoritmo está en $O(n^c)$ para algún entero $c \geq 1$, siendo n el tamaño de la entrada.*

Definición 2.3.7 (Algoritmo exponencial) *Un algoritmo se dice exponencial si la función que da el número de operaciones del algoritmo está en $O(k^n)$ para algún entero $k \geq 1$ y siendo n el tamaño de la entrada.*

Se dice que los problemas que se pueden resolver con un algoritmo polinómico en el tamaño de la entrada se consideran tratables, mientras que aquellos algoritmos más costosos, más lentos, se denominan intratables. Los siguiente conceptos sirven para comparar la relación entre dos problemas.

Definición 2.3.8 (Problema reducible en tiempo polinómico a otro) *Dados dos problemas A y B, se dice que A es reducible en tiempo polinómico (o reducible polinómicamente) a B si A se puede resolver en tiempo polinómico en caso de contar con un algoritmo que solucione B en tiempo polinómico.*

Definición 2.3.9 (Problemas computacionalmente equivalentes) *Dos problemas intratables A y B son computacionalmente equivalentes si son reducibles polinómicamente entre sí.*

3 Criptosistema de Cocks

El criptosistema IBE que vamos a estudiar es el de Cocks, que fue inventado por el matemático británico Clifford Cocks [25], que trabajaba para la agencia de inteligencia británica GCHQ (Communications-Electronics Security Group).

Aunque no se sabe a ciencia cierta, se cree que fue inventado en el año 1973. De ser así, seria el primer algoritmo de clave pública implementado.
Sin embargo, debido al elevado coste de las computadoras necesarias para implementarlo en la época, su idea no trascendió por lo que su descubrimiento no fue revelado hasta 1997 dada su confidencial. Para entonces, ya se sabía de un nuevo criptosistema de características similares llamado RSA (Rivest, Shamir y Adleman).

La idea principal del criptosistema asimétrico de Cocks se basa en la utilización del par de claves pública-privada para cifrar y descifrar un archivo (no tiene por qué ser un texto, podría ser otro tipo de documento como una imagen). Es decir, tras obtener la descomposición en bits del archivo, se les aplica el proceso de cifrado (o de descifrado según el caso) a cada uno de los bits por separado y en orden.

A continuación, explicamos los algoritmos de cifrado y de descifrado, así como la obtención de las dos claves.

3.1. Proceso de cifrado

Comenzamos con la obtención del par de claves.
Se utilizan dos elementos: un valor numérico n y una buena función hash criptográfica. Ambos son elementos públicos.

El valor n es el producto de dos números primos distintos que a partir de ahora llamaremos p y q. Además, estos factores deben de ser congruentes con 3 módulo 4.
Los valores dados a p y a q serán privados, es decir, solo debe de conocerlos el algoritmo que los generó.

La función hash criptográfica podrá ser cualquier función hash definida por $H_1 : \{0,1\}^* \to \mathbb{Z}_n$. Además, tiene que cumplir que el símbolo de Jacobi $\left(\frac{a}{n}\right) = 1$ siendo $H_1(ID) = a$ para cualquier valor $ID \in \{0,1\}^*$.

Esta propiedad nos asegura que uno de los dos valores a o $-a$ sea un cuadrado módulo n. Con el cumplimiento de la propiedad tendríamos $1 = \left(\frac{a}{n}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{q}\right)$ y por tanto o bien $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = 1$, o bien $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = -1$.
En el primer caso, cuando ambos son 1, tendríamos que a es un cuadrado módulo n por ser también un cuadrado módulo p y módulo q.

Y en el otro caso, cuando ambos son -1, $-a$ es un cuadrado módulo n.
Esto último se debe a que por definición teníamos $p = 3 \mod 4$ y $q = 3 \mod 4$, luego $\left(\frac{-1}{p}\right) = \left(\frac{-1}{q}\right) = -1$ y por tanto

$$\left(\frac{-a}{n}\right) = \left(\frac{a}{p}\right)\left(\frac{-1}{p}\right)\left(\frac{a}{q}\right)\left(\frac{-1}{q}\right) = \left(\frac{a}{p}\right)(-1)\left(\frac{a}{q}\right)(-1) = \left(\frac{a}{p}\right)\left(\frac{a}{q}\right) = (+1)(+1) = 1$$

Entonces, $-a$ es un cuadrado porque es el producto de dos números que son cuadrados.

Sin embargo, dado que el usuario desconoce los valores p y q también desconoce cuál de los dos valores (a o $-a$) es un cuadrado módulo n. Es necesario realizar dos cifrados, uno para cada caso.

Con todo ello tendríamos que el valor a es la clave pública correspondiente a una identificación *ID* concreta.

El proceso de cifrado cifrará un solo bit cada vez, obteniendo como resultado un par de números (s_1, s_2).
Para cifrar un bit m se siguen los siguientes pasos:
Primero, codificamos el bit como $x = (-1)^m$. Por tanto, tendríamos codificado el bit 0 como 1 y el 1 como -1.
A continuación, tomamos de manera aleatoria dos números enteros t_1 y t_2 distintos y que cumplan:

$$\left(\frac{t_1}{n}\right) = \left(\frac{t_2}{n}\right) = x$$

Por último, obtenemos los cifrados del bit mediante las fórmulas:

$$s_1 = \left(t_1 + \frac{a}{t_1}\right) \mod n, \quad s_2 = \left(t_2 - \frac{a}{t_2}\right) \mod n$$

Notemos la importancia de que los valores t_1 y t_2 sean distintos. Si fuesen iguales, obtendríamos:

$$\frac{s_1 + s_2}{2} = \frac{1}{2}\left(t_1 + \frac{a}{t_1}\right) + \frac{1}{2}\left(t_1 - \frac{a}{t_1}\right) \mod n = t_1 \mod n,$$

recuperando así el valor de t_1 y con ello el de $x = \left(\frac{t_1}{n}\right)$, lo cual nos diría fácilmente cual fue el bit a cifrar y el bit en claro.

A continuación, mostremos un resumen por pasos del cifrado:

Conocida la clave pública a, el bit m a cifrar y el entero n tenemos:

1. *Codificamos m como $x = (-1)^m$*
2. *Tomamos aleatoriamente t_1 y t_2 tales que $\left(\frac{t_1}{n}\right) = \left(\frac{t_2}{n}\right) = x$*
3. *Calculamos $s_1 = \left(t_1 + \frac{a}{t_1}\right) \mod n$, $s_2 = \left(t_2 - \frac{a}{t_2}\right) \mod n$*
4. *(s_1, s_2) es el bit cifrado.*

3.2. Proceso de descifrado

Tras recibir el documento cifrado, se procede a descifrarlo también bit a bit. Recordemos que de cada bit recibimos dos cifrados, s_1 y s_2, de los cuales solo aquel que fue cifrado mediante la clave que era un cuadrado módulo n es el que permite descifrar el mensaje. Para saber cuál es se utiliza la llave

privada.

La clave privada (a la que a partir de ahora llamaremos r) correspondiente a la clave pública a será la raíz cuadrada de a o de $-a$, es decir, $r^2 = a \mod n$ o $r^2 = -a \mod n$.

Teniendo esto en cuenta, y que $p = 3 \mod 4$ y $q = 3 \mod 4$, podemos decir que $p-1 = 2 \mod 4$ y $q-1 = 2 \mod 4$. Es decir, es posible escribir $p-1$ y $q-1$ como $p-1 = 4k_1+2$ y $q-1 = 4k_2+2$ para dos enteros k_1 y k_2.
Por otro lado, dado que $n = p \cdot q$ con p y q primos, la función de Euler correspondiente a n es $\phi(n) = \phi(p)\phi(q) = (p-1)(q-1) = (4k_1+2)(4k_2+2) = 16k_1k_2+8k_1+8k_2+4 = 8(2k_1k_2+k_1+k_2+1)-4$.
Así, $\phi(n)+4 = 8(2k_1k_2+k_1+k_2+1)$, es decir, 8 divide a $\phi(n)+4$, lo que permite usar para calcular la raíz cuadrada del siguiente modo:

$$r = a^{(\phi(n)+4)/8} \mod n.$$

Es fácil comprobar que tal r es una raíz de a o de $-a$, dado que:

$$r^2 = a^{2(\phi(n)+4)/8} = a^{(\phi(n)+4)/4} = a^{\frac{\phi(n)}{4}+1} = a^{\phi(n)^{1/4}}a \mod n$$

y tras aplicar el teorema de Euler $\left(a^{\phi(n)} = 1 \mod n\right)$ obtenemos

$$r^2 = 1^{1/4}a = \pm a \mod n$$

Concluimos así que si a es una raíz cuadrada módulo n, entonces r satisfará $r^2 = a \mod n$ y sera s_1 el valor correspondiente al bit cifrado. Y si $-a$ es una raíz cuadrada módulo n, entonces r satisfará $r^2 = -a \mod n$ y en este caso sera s_2 el valor a tomar como el cifrado del bit.

Tras, gracias a r, decidir el valor correcto s a descifrar ($s = s_1$ o $s = s_2$ según el caso), se descifra el bit mediante la fórmula

$$x = \left(\frac{s+2r}{n}\right)$$

Veamos porque esta fórmula descifra el mensaje.

- En el caso $r^2 = a \mod n$, tenemos $s = \left(t_1 + \frac{a}{t_1}\right) \mod n$, luego

$$s+2r = \left(t_1+\frac{a}{t_1}\right)+2r = t_1\left(1+\frac{a}{t_1^2}+\frac{2r}{t_1}\right) = t_1\left(1+\frac{r^2}{t_1^2}+\frac{2r}{t_1}\right) \mod n$$

y, como $\left(1+\frac{r^2}{t_1^2}+\frac{2r}{t_1}\right) = \left(1+\frac{r}{t_1}\right)^2$, entonces $s+2r = t_1\left(1+\frac{r}{t_1}\right)^2 \mod n$.
Así, $s+2r$ es un cuadrado módulo n cuando t_1 lo es, es decir, $\left(\frac{s+2r}{n}\right) = \left(\frac{t_1}{n}\right) = x$.

- En el caso $r^2 = -a \mod n$, tenemos $s = \left(t_2 - \frac{a}{t_2}\right) \mod n$, luego

$$s+2r = \left(t_2-\frac{a}{t_2}\right)+2r = t_2\left(1-\frac{a}{t_2^2}+\frac{2r}{t_2}\right) = t_2\left(1+\frac{r^2}{t_2^2}+\frac{2r}{t_2}\right) \mod n = t_2\left(1+\frac{r}{t_2}\right)^2 \mod n$$

Así, $s+2r$ es un cuadrado módulo n cuando t_2 lo es, es decir, $\left(\frac{s+2r}{n}\right) = \left(\frac{t_2}{n}\right) = x$.

Si resumimos este proceso brevemente tenemos

Conocida la clave pública a, y la factorización en primos p y q de n entonces

- *Se calcula la clave privada como* $r = a^{(\phi(n)+4)/8} \mod n$
- *Para el descifrado se siguen los siguientes pasos:*
  1. *Si* $r^2 = a \mod n$ *entonces* $s = s_1$, *en caso contrario* $s = s_2$
  2. *Se calcula* $x = \left(\frac{s+2r}{n}\right)$
  3. *Si* $x = 1$ *entonces* $m = 0$, *y si* $x = -1$ *sería* $m = 1$
  4. m *es el bit desencriptado.*

3.3. Ejemplo numérico

A continuación, mostramos un ejemplo práctico del criptosistema de Cocks en los dos apartados anteriores, es decir, un ejemplo de la obtención de las dos claves y de la realización de los dos procesos, cifrado y descifrado.

Supongamos que tenemos los siguientes datos: la clave pública $a = 10$, y los valores $p = 7$ y $q = 11$. Entonces tenemos $n = 77$.
El primer paso sería calcular la clave privada $r = a^{(\phi(n)+4)/8} \mod n$.
Entonces: $r = 10^{(\phi(77)+4)/8} \mod 77 = 10^{64/8} \mod 77 = 10^8 \mod 77 = 23$, por tanto en este caso tenemos $r^2 = -a \mod n$ dado que $r^2 = 23^2 = 67 \mod 77 = -10 \mod 77$.

• Utilicemos primero un bit "0":
Para cifrarlo, se codifica como $x = (-1)^0 = 1$ y se eligen dos enteros aleatorios t_1 y t_2 que cumplan $\left(\frac{t_1}{77}\right) = \left(\frac{t_2}{77}\right) = 1$. Por ejemplo podemos tomar $t_1 = 4$ y $t_2 = 6$.
Luego, calculamos los dos valores:

$$s_1 = \left(t_1 + \frac{a}{t_1}\right) \mod n = \left(4 + \frac{10}{4}\right) \mod 77 = (4 + 10 \cdot 58) \mod 77 = 584 \mod 77 = 45$$

$$s_2 = \left(t_2 - \frac{a}{t_2}\right) \mod n = \left(6 - \frac{10}{6}\right) \mod 77 = (6 - 10 \cdot 13) \mod 77 = -124 \mod 77 = 30$$

y se enviaría el par de texto cifrado $(s_1, s_2) = (45, 30)$ al destinatario.

El receptor sabe que su clave privada satisface $r^2 = -a \mod n$, por lo que elige s_2 para descifrar. Entonces calcula: $x = \left(\frac{s_2+2r}{n}\right) = \left(\frac{30+2\cdot 23}{77}\right) = \left(\frac{76}{77}\right) = 1$. Por tanto, descodifica el bit como "0".

• Hagamos lo mismo ahora para un bit "1":
Se codifica como $x = (-1)^1 = -1$ y se eligen dos enteros aleatorios t_1 y t_2 que cumplan $\left(\frac{t_1}{77}\right) = \left(\frac{t_2}{77}\right) = -1$. Por ejemplo podemos tomar $t_1 = 8$ y $t_2 = 2$.
Luego, calculamos los dos valores:

$$s_1 = \left(t_1 + \frac{a}{t_1}\right) \mod n = \left(8 + \frac{10}{8}\right) \mod 77 = (8 + 10 \cdot 29) \mod 77 = 298 \mod 77 = 67$$

$$s_2 = \left(t_2 - \frac{a}{t_2}\right) \mod n = \left(2 - \frac{10}{2}\right) \mod 77 = (2 - 10 \cdot 39) \mod 77 = -388 \mod 77 = 74$$

y se enviaría el par de texto cifrado $(s_1, s_2) = (67, 74)$ al destinatario.

El receptor sabe que su clave privada satisface $r^2 = -a \mod n$, por lo que elige s_2 para descifrar. Entonces calcula: $x = \left(\frac{s_2+2r}{n}\right) = \left(\frac{74+2\cdot 25}{77}\right) = \left(\frac{120}{77}\right) = -1$. Por tanto, descodifica el bit como "1".

3.4. Seguridad del sistema

Centrémonos ahora en la seguridad del sistema, en el grado de dificultad necesario para romperlo. Veamos que la seguridad del sistema de Cocks deriva del problema de la residuosidad cuadrática, aunque esta relación no es del todo obvia.

El hecho de que la seguridad de Cocks se base en la dificultad del problema de la residuosidad cuadrática se relaciona con el hecho de que la capacidad de descifrar un mensaje cifrado con Cocks requiera decidir si la clave pública (a) del usuario es un cuadrado módulo n o no. Esta relación la vimos en el apartado 3.2 (*Proceso de descifrado*) con el fin de elegir correctamente la parte del bit cifrado recibido.
Veamos esto con más detalle.

Primero observemos que el símbolo de Jacobi de un entero es el mismo que el de su inverso módulo n. Es decir, que $\left(\frac{t}{n}\right) = \left(\frac{1/t}{n}\right)$ para cualquier entero t con $n \in \mathbb{N}$ dado que $\left(\frac{t}{n}\right) \cdot \left(\frac{1/t}{n}\right) = \left(\frac{1}{n}\right) = +1$. Por tanto, para cualquier entero a:

$$\left(\frac{a/t}{n}\right) = \left(\frac{a}{n}\right) \cdot \left(\frac{1/t}{n}\right) = \left(\frac{a}{n}\right) \cdot \left(\frac{t}{n}\right).$$

Ahora, consideremos los sistemas de congruencias siguientes:

$$\begin{cases} t_1 = t & \text{mód } p \\ t_1 = t & \text{mód } q \end{cases}$$

$$\begin{cases} t_2 = t & \text{mód } p \\ t_2 = \frac{a}{t} & \text{mód } q \end{cases}$$

$$\begin{cases} t_3 = \frac{a}{t} & \text{mód } p \\ t_3 = t & \text{mód } q \end{cases}$$

$$\begin{cases} t_4 = \frac{a}{t} & \text{mód } p \\ t_4 = \frac{a}{t} & \text{mód } q \end{cases}$$

Si los resolvemos utilizando el teorema chino del resto obtenemos las siguientes soluciones:

$$t_1 = t \cdot e_1 + t \cdot e_2$$
$$t_2 = t \cdot e_1 + \frac{a}{t} \cdot e_2$$
$$t_3 = \frac{a}{t} \cdot e_1 + t \cdot e_2$$
$$t_4 = \frac{a}{t} \cdot e_1 + \frac{a}{t} \cdot e_2$$

donde e_1 y e_2 son tales que

$$e_1 = \begin{cases} 1 & \text{mód } p \\ 0 & \text{mód } q \end{cases} \quad y \quad e_2 = \begin{cases} 0 & \text{mód } p \\ 1 & \text{mód } q \end{cases}$$

Las expresiones anteriores también se pueden expresar como:

$$\left(\frac{t_1}{n}\right) = \left(\frac{t}{p}\right)\left(\frac{t}{q}\right)$$
$$\left(\frac{t_2}{n}\right) = \left(\frac{t}{p}\right)\left(\frac{a/t}{q}\right) = \left(\frac{t}{p}\right)\left(\frac{a}{q}\right)\left(\frac{t}{q}\right)$$
$$\left(\frac{t_3}{n}\right) = \left(\frac{a/t}{p}\right)\left(\frac{t}{q}\right) = \left(\frac{a}{p}\right)\left(\frac{t}{p}\right)\left(\frac{t}{q}\right)$$
$$\left(\frac{t_4}{n}\right) = \left(\frac{a/t}{p}\right)\left(\frac{a/t}{q}\right) = \left(\frac{a}{p}\right)\left(\frac{t}{p}\right)\left(\frac{a}{q}\right)\left(\frac{t}{q}\right)$$

Luego si a no es un cuadrado módulo n, tendríamos $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = -1$ y, por tanto:

$$\begin{aligned}
\left(\tfrac{t_1}{n}\right) &= \left(\tfrac{t}{p}\right)\left(\tfrac{t}{q}\right) \\
\left(\tfrac{t_2}{n}\right) &= \left(\tfrac{t}{p}\right)\left(\tfrac{a}{q}\right)\left(\tfrac{t}{q}\right) = -\left(\tfrac{t}{p}\right)\left(\tfrac{t}{q}\right) \\
\left(\tfrac{t_3}{n}\right) &= \left(\tfrac{a}{p}\right)\left(\tfrac{t}{p}\right)\left(\tfrac{t}{q}\right) = -\left(\tfrac{t}{p}\right)\left(\tfrac{t}{q}\right) \\
\left(\tfrac{t_4}{n}\right) &= \left(\tfrac{a}{p}\right)\left(\tfrac{t}{p}\right)\left(\tfrac{a}{q}\right)\left(\tfrac{t}{q}\right) = \left(\tfrac{t}{p}\right)\left(\tfrac{t}{q}\right)
\end{aligned}$$

Esto es:

$$\left(\tfrac{t_1}{n}\right) = \left(\tfrac{t_4}{n}\right) \quad y \quad \left(\tfrac{t_2}{n}\right) = \left(\tfrac{t_3}{n}\right)$$

Pero que:

$$\left(\frac{t_1}{n}\right) = -\left(\frac{t_2}{n}\right)$$

Tengamos en cuenta que si utilizamos los t_i anteriores como la pareja de elementos aleatorios utilizados en el cifrado de Cocks, obtendríamos los mismos cifrados en las distintas parejas: t_1, t_4 y t_2, t_3. Es decir:

Por un lado, para los valores t_1 y t_4 obtendríamos los cifrados:

$$s_1 = \left(t_1 + \frac{a}{t_1}\right) = \left(t \cdot e_1 + t \cdot e_2 + \frac{a}{t \cdot e_1 + t \cdot e_2}\right) = t + \frac{a}{t}$$

$$s_2 = \left(t_4 - \frac{a}{t_4}\right) = \left(\frac{a}{t} \cdot e_1 + \frac{a}{t} \cdot e_2 + \frac{a}{\frac{a}{t} \cdot e_1 + \frac{a}{t} \cdot e_2}\right) = \frac{a}{t} - t$$

Mientras que, por otro lado, si tomásemos otros valores, en este caso t_2 y t_3 obtendríamos:

$$s_1' = \left(t_2 + \frac{a}{t_2}\right) = \left(t \cdot e_1 + \frac{a}{t} \cdot e_2 + \frac{a}{t \cdot e_1 + \frac{a}{t} \cdot e_2}\right) = t + \frac{a}{t}$$

$$s_2' = \left(t_3 - \frac{a}{t_3}\right) = \left(t \cdot e_1 + \frac{a}{t} \cdot e_2 + \frac{a}{t \cdot e_1 + \frac{a}{t} \cdot e_2}\right) = \frac{a}{t} - t$$

Entonces cualquier t_i con $i \in \{1, 2, 3, 4\}$ puede ser utilizado como el elemento aleatorio en el cifrado del criptosistema de Cocks, y por tanto que con cualquiera de ellos se crea el mismo texto cifrado. Y la única forma de distinguir entre estos casos es poder determinar si a es un cuadrado o no módulo n, que es el problema de la residuosidad cuadrática.

En el caso en el que a no fuese un cuadrado, tendríamos entonces casos donde el mismo texto cifrado puede provenir de diferentes valores de texto plano. Veamos esto con más detalle.

Una vez que ya hemos identificado de dónde proviene la seguridad del sistema de Cocks, procedamos a ver que efectivamente este es seguro. Es decir, que derrotar la seguridad del sistema de Cocks no es más fácil que resolver el problema de residuos cuadráticos, de modo que si un adversario puede descifrar un mensaje encriptado con el esquema Cocks puede usar su algoritmo de descifrado para resolver el problema de los residuos cuadráticos.

Para ello, probaremos la equivalencia computacional de resolver el problema de la residuosidad cuadrática y del problema de factorizar n. Así, si el problema de la residuosidad cuadrática es suficientemente

intratable también tenemos que el esquema Cocks es suficientemente seguro.

Sabemos que obtener los factores primos de un número entero n es un problema relativamente difícil computacionalmente cuando se trata de números grandes.
Como curiosidad, existe un desafío, el RSA Factoring Challenge, para fomentar la investigación sobre la teoría de números computacionales y la dificultad práctica de factorizar enteros grandes y descifrar claves RSA. Hasta el momento, el mayor número factorizado en este desafío fue el número RSA-250, que tiene 250 dígitos decimales (829 bits) y fue factorizado en febrero de 2020 por Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé y Paul Zimmermann [6]. La factorización de RSA-250 utilizó aproximadamente 2700 años de núcleo de CPU.
Por el momento, no se conoce ningún algoritmo clásico para encontrar los factores de un número que realice la operación en un tiempo polinómico. Algunos de los algoritmos de factorización son incluso exponenciales, como por ejemplo el algoritmo de factorización por divisiones sucesivas.

A partir de ahora denotaremos con RQ_n al conjunto de los residuos cuadráticos módulo n, es decir, al subconjunto $RQ_n = \{x \in \mathbb{Z}_n^* \text{ tal que } x \equiv y^2 \mod n, \text{ con } y \in \mathbb{Z}_n^*\} \subseteq \mathbb{Z}_n$.
También usaremos J_n para referirnos al conjunto de elementos de $\mathbb{Z}_n$ que tienen símbolo de Jacobi igual a 1, es decir, $J_n = \{x \in \mathbb{Z}_n \text{ tal que } \left(\frac{x}{n}\right) = 1\}$.

Tengamos en cuenta que $RQ_n \subseteq J_n$ y, por lo tanto, dado $x \in \mathbb{Z}_n \setminus J_n$ sería fácil decidir que x no es un residuo cuadrático calculando el símbolo de Jacobi.
Si de alguna manera se consigue factorizar el numero n en dos primos distintos p y q, entonces se resuelve el problema de la residuosidad cuadrática módulo n porque se pueden calcular los símbolos de Jacobi módulo cada uno de los primos.

Veamos ahora el sentido inverso, es decir, si sabiendo resolver el problema de la residuosidad cuadrática se puede llegar a factorizar n.

Con la notación anterior, podemos considerar otra visión distinta del problema de la residuosidad cuadrática: el problema de la residuosidad cuadrática se puede ver como el problema de pertenencia a un subconjunto. Veámoslo.

Definición 3.4.1 ***(Problema de pertenencia a un subconjunto)*** *Sea $\mathcal{C} \subseteq \mathbb{Z}_n$ y $\mathcal{V} \subseteq \mathbb{Z}_n$ tales que $\mathcal{V} \subseteq \mathcal{C} \subseteq \mathbb{Z}_n$. El problema de pertenencia a un subconjunto consiste en dado $x \in \mathcal{C}$ decir si $x \in \mathcal{V}$.*

Así, tomando $\mathcal{C} = J_n$ y $\mathcal{V} = RQ_n$ en la definición anterior tendríamos el problema de la residuosidad cuadrática.

Introduzcamos algunos conceptos que nos ayudarán.

Sea $n = p \cdot q$ producto de primos. Por el teorema chino de los restos sabemos que el anillo $\mathbb{Z}_n$ es isomorfo al producto de anillos $\mathbb{Z}_p \times \mathbb{Z}_q$. Llamemos a este isomorfismo f, definido por $f : \mathbb{Z}_p \times \mathbb{Z}_q \longrightarrow \mathbb{Z}_n$.
Sea $\mathcal{C} \subseteq \mathbb{Z}_n$ y sean $\mathcal{C}_p = \{y \mod p : y \in \mathcal{C}\}$ y $\mathcal{C}_q = \{y \mod q : y \in \mathcal{C}\}$.

Definición 3.4.2 ***(Clausura uniforme de $\mathcal{C} \subseteq \mathbb{Z}_n$)*** *Definimos $\mathcal{U}[\mathcal{C}] \subseteq \mathbb{Z}_n$ como la clausura uniforme de $\mathcal{C} \subseteq \mathbb{Z}_n$ con $\mathcal{U}[\mathcal{C}] = \{y \in \mathbb{Z}_n : y = f(y_1, y_2) \text{ con } y_1 \in \mathcal{C}_p, y_2 \in \mathcal{C}_q\}$.*

Así si tenemos $\mathcal{C} = \{a, b, c\} = \{f(a,a), f(b,b), f(c,c)\}$ entonces $\mathcal{U}[\mathcal{C}] = \{f(d_p, d_q) : d_p \in \{a \mod p, b \mod p, c \mod p\},\ d_q \in \{a \mod q, b \mod q, c \mod q\}\}$.

Lema 3.4.1 *Tomar y uniformemente aleatorio de $\mathcal{U}[\mathcal{C}]$ es equivalente a primero tomar y_i uniforme e independientemente de $\mathcal{C}$ para $i \in \{1, 2\}$ y luego establecer $y = f(y_1, y_2)$.*

Otro concepto que tendremos en cuenta en esta demostracion es el de algoritmos de anillo genéricos. Son un tipo de algoritmos que realizan una secuencia de operaciones de anillo ($\{+, -, \cdot, /\}$ siendo $i/j = i \cdot j^{-1}$) en los valores de entrada ($1 \in \mathbb{Z}_n$ y x).
Estos algoritmos no realizan siempre la misma secuencia de operaciones de anillo para cualquier valor de entrada, sino qué pueden decidir adaptativamente que operación de anillo se realiza a continuación. Además, la salida de algoritmos de anillo genéricos no está restringida a elementos del anillo.
Un caso particular de los algoritmos de anillo genéricos son los straight line programs (programas de línea recta). La diferencia es que estos si realiza una secuencia fija de operaciones, y que si generan un elemento del anillo.

Definición 3.4.3 ***(Straight line program)*** *Un straight line program P formado por m secuencias de operaciones sobre $\mathbb{Z}_n$ es un secuencia de tuplas*

$$P = ((i_1, j_1, o_1), \ldots, (i_m, j_m, o_m))$$

donde $-1 \leq i_k, j_k < k$ y $o_i \in \{+, -, \cdot, /\}$ para $i \in \{1, \ldots, m\}$.
La salida $P(x)$ del algoritmo para la entrada $x \in \mathbb{Z}_n$ se calcula de la siguiente manera:

1. *Inicialmente $L_{-1} := 1 \in \mathbb{Z}_n$ y $L_0 := x$.*
2. *Para k desde 1 hasta m se hace:*
   - si $o_k = /$ y $L_{j_k} \notin \mathbb{Z}_n^*$ entonces: retorna $\perp$,
   - si no: $L_k := L_{i_k}\, o\, L_{j_k}$.
3. *Devuelve $P(x) = L_m$.*

A cada triple $(i, j, o) \in P$ se le denomina SLP-paso del algoritmo de anillo genérico.

Denotaremos con P_k al algoritmo de anillo genérico dado por la secuencia de los primeros k elementos de P. Además, supondremos $P_{-1}(x) = 1$ y $P_0(x) = x$ para todo $x \in \mathbb{Z}_n$.

Formalizamos esta noción de algoritmos de anillo genéricos en términos de un juego entre un algoritmo $\mathcal{A}$ y un oráculo de anillo genérico $\mathcal{O}$, donde ambos intentaran encontrar la solución del problema.
El oráculo de anillo genérico recibe como entrada un valor secreto $x \in \mathbb{Z}_n$. Este mantiene una secuencia P (siendo secuencia vacía al comienzo) e implementa dos subrutinas internas:

- $test()$, que toma como entrada una tupla $(j, o) \in \{-1, \ldots, |P|\} \times \{+, -, \ldots, /\}$ y devuelve: falso si $o = /$ y $P_j(x) \notin \mathbb{Z}_n^*$, y verdadero en otro caso.
- $equal()$, que toma como entrada una tupla $(i, j) \in \{-1, \ldots, |P|\} \times \{-1, \ldots, |P|\}$ y devuelve: verdadero si $P_i(x) \equiv P_j(x) \mod n$, y falso en caso contrario.

Para realizar los cálculos, el algoritmo envía los SLP-pasos (i, j, o) a $\mathcal{O}$. Siempre que el algoritmo envía (i, j, o) con $o \in \{+, -, \cdot, /\}$, el oráculo ejecuta $test(j, o)$ y si retorna falso devuelve el símbolo de error $\perp$. De lo contrario, (i, j, o) se agrega a P. Además, el algoritmo puede consultar el oráculo para comprobar la igualdad de los elementos del anillo calculados mediante el envío de una consulta (i, j, o) tal que $o \in \{=\}$. En este caso, el oráculo devuelve $equal(i, j)$. Medimos la complejidad de $\mathcal{A}$ por el número de consultas al oráculo.

Formalicemos ahora la noción del problema de pertenencia a un subconjunto en el modelo de anillo genérico como el que acabamos de ver. En este caso nuestro oráculo $\mathcal{O}_{smp}$ recibirá un elemento x uniformemente aleatorio de $\mathcal{C}$ como entrada. Diremos que el algoritmo $\mathcal{A}$ vence al oráculo si $x \in \mathcal{V}$ y $\mathcal{A}^{\mathcal{O}_{smp}}(n) = 1$, o si $x \in \mathcal{V}$ y $\mathcal{A}^{\mathcal{O}_{smp}}(n) = 0$.

Tengamos en cuenta que cualquier algoritmo para un problema de pertenencia a un subconjunto dado $(\mathcal{C}, \mathcal{V})$ tiene al menos la probabilidad de éxito trivial para acertar: $\Pi(\mathcal{C}, \mathcal{V}) = \text{máx}\{|\mathcal{V}|/|\mathcal{C}|, 1 - |\mathcal{V}|/|\mathcal{C}|\}$, esto se debe al hecho de que x se ha tomado de manera uniformemente aleatoria de $\mathcal{C}$. Por tanto, el algoritmo $\mathcal{A}$ que resuelven el problema de pertenencia a un subconjunto con una probabilidad

de éxito $Pr[\mathcal{S}]$ tiene como ventaja: $Adv_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{smp}}(n)) = |Pr[\mathcal{S}] - \Pi(\mathcal{C},\mathcal{V})|$.

Podemos enunciar así el siguiente teorema:

Teorema 3.4.1 *Para cualquier algoritmo de anillo genérico $\mathcal{A}$ que resuelva un problema de pertenencia a un subconjunto dado $(\mathcal{C},\mathcal{V})$ sobre $\mathbb{Z}_n$ con ventaja $Adv_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{smp}}(n))$ al realizar m consultas a $\mathcal{O}_{smp}$, existe un algoritmo $\mathcal{B}$ que calcula un factor de n con una probabilidad de éxito de al menos*

$$\frac{Adv_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{smp}}(n))}{2m(m^2+5m+3)} \cdot \left(\frac{|\mathcal{C}|}{|\mathcal{U}[\mathcal{C}]|}\right)^2$$

ejecutando $\mathcal{A}$ una vez y realizando $O(m^3)$ operaciones adicionales en $\mathbb{Z}_n$, m cálculos del mcd en $\lceil \log 2 \rceil$ números de n bits, y tomando muestreo de m elementos aleatorios de $\mathcal{U}[\mathcal{C}]$.

Veamos de donde viene la expresión del teorema anterior, es decir, la relación entre la probabilidad de éxito de $\mathcal{B}$ con la ventaja de $\mathcal{A}$.

Decimos que se produce un fallo de simulación, denominado $\mathcal{F}$, si no se logra simular $\mathcal{O}_{smp}$ perfectamente. Este fallo puede pasar en las funciones $test()$ y $equal()$ del oráculo, es decir, $\mathcal{F}$ implica que al menos uno de los eventos $\mathcal{F}_{test}$ y $\mathcal{F}_{equal}$ ha ocurrido. Esto implica que $Pr[\mathcal{F}] \leq Pr[\mathcal{F}_{test}] + Pr[\mathcal{F}_{equal}]$.

Acotemos estos fallos.
Para ello se tiene en cuenta el funcionamiento interno del algoritmo y la probabilidad obtenida para cada posible situación. A continuación doy las cotas, aunque estas no son probadas por su larga extension, pero estas pruebas se encuentran en los anexos del documento [20].

- $Pr[\mathcal{F}_{test}] \leq 2m(m+1)\ \text{máx}_{0\leq j\leq m}\left\{Pr\left[P_j(x) \notin \mathbb{Z}_n^* \text{ y } P_j(x') \in \mathbb{Z}_n^* : x, x' \xrightarrow{\mathcal{U}} \mathcal{C}\right]\right\}$
- $Pr[\mathcal{F}_{equal}] \leq 2m(m^2+3m+1)\ \text{máx}_{-1\leq i<j\leq m}\{Pr\left[P_i(x) \equiv P_j(x) \mod n \text{ y } P_i(x') \not\equiv P_j(x') \mod n : x, x' \xrightarrow{\mathcal{U}} \mathcal{C}\right]\} + 2m(m+1)\ \text{máx}_{0\leq k\leq m}\left\{Pr\left[P_k(x) \notin \mathbb{Z}_n^* \text{ y } P_k(x') \in \mathbb{Z}_n^* : x, x' \xrightarrow{\mathcal{U}} \mathcal{C}\right]\right\}$

donde $x, x' \xrightarrow{\mathcal{U}} \mathcal{C}$ quiere decir que se tomaron x, x' de manera independiente y mediante la distribución uniforme del conjunto $\mathcal{C}$.

Por tanto, juntando ambas cotas se obtiene:

$$Pr[\mathcal{F}] \leq Pr[\mathcal{F}_{test}] + Pr[\mathcal{F}_{equal}] \leq$$

$$2m(m^2+3m+1) \max_{-1\leq i<j\leq m}\left\{Pr\left[P_i(x) \equiv P_j(x) \mod n \text{ y } P_i(x') \not\equiv P_j(x') \mod n : x, x' \xrightarrow{\mathcal{U}} \mathcal{C}\right]\right\} +$$

$$+ 4m(m+1) \max_{0\leq k\leq m}\left\{Pr\left[P_k(x) \notin \mathbb{Z}_n^* \text{ y } P_k(x') \in \mathbb{Z}_n^* : x, x' \xrightarrow{\mathcal{U}} \mathcal{C}\right]\right\}$$

$$= 2m(m^2+3m+1)\cdot\gamma_2 + 4m(m+1)\cdot\gamma_1$$

Dado que todos los cálculos de un algoritmo de anillo genérico son independientes del valor de entrada x, cualquier algoritmo tiene solo la probabilidad de éxito trivial al interactuar con el simulador. Por lo tanto, la probabilidad de éxito de cualquier algoritmo al interactuar con el oráculo original está limitada por

$$\Pi(\mathcal{C},\mathcal{V}) + Adv_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{smp}}(n)) = Pr[\mathcal{S}] \leq Pr[\mathcal{S}_{sim}] + Pr[\mathcal{F}] \leq \Pi(\mathcal{C},\mathcal{V}) + Pr[\mathcal{F}]$$

lo cual implica que $Adv_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{smp}}(n)) \leq Pr[\mathcal{F}]$.

Con todo ello, usando las definiciones anteriores de γ_1 , γ_2 y $\gamma = \text{máx}\{\gamma_1, \gamma_2\}$, el hecho de que $Adv_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{smp}}(n)) \leq Pr[\mathcal{F}]$, y el límite obtenido para $Pr[\mathcal{F}]$, se puede obtener un límite inferior en γ del siguiente modo:

$$Adv_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{smp}}(n)) \leq Pr[\mathcal{F}] \leq 4m(m+1)\gamma_1 + 2m(m^2+3m+1)\gamma_2 \leq 2m(m^2+5m+3)\gamma.$$

Esto implica:

$$\gamma \geq \frac{Adv_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{smp}}(n))}{2m(m^2+5m+3)}.$$

Además, tengamos en cuenta los dos siguientes lemas sobre los straight line programs:

El primero proporciona un límite inferior de la probabilidad de factorizar n al evaluar un straight line program P:

Lema 3.4.2 *Para cualquier straight line program P y $\mathcal{C} \subseteq \mathbb{Z}_n$ se tiene*

$$Pr\left[P(x') \notin \mathbb{Z}_n^* \text{ y } P_k(x) \in \mathbb{Z}_n^* : x, x' \xrightarrow{\mathcal{U}} \mathcal{C}\right] \leq \left(\frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|}\right)^2 \cdot Pr\left[mcd(n, P(y)) \notin \{1, n\} : y \xrightarrow{\mathcal{U}} \mathcal{U}[\mathcal{C}]\right]$$

El segundo proporciona un límite inferior en la probabilidad de factorizar n calculando mediante dos straight line programs P y Q:

Lema 3.4.3 *Para cualquier par de straight line program P y Q, y $\mathcal{C} \subseteq \mathbb{Z}_n$ se tiene*

$$Pr\left[P(x) \equiv Q(x) \mod n \text{ y } P(x') \not\equiv Q(x') \mod n : x, x' \xrightarrow{\mathcal{U}} \mathcal{C}\right] \leq$$

$$\leq \left(\frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|}\right)^2 \cdot Pr\left[mcd(n, P(y) - Q(y)) \notin \{1, n\} : y \xrightarrow{\mathcal{U}} \mathcal{U}[\mathcal{C}]\right]$$

Gracias a ellos obtenemos que $\gamma_1 \cdot \left(\frac{|\mathcal{C}|}{|\mathcal{U}[\mathcal{C}]|}\right)^2$ y $\gamma_2 \cdot \left(\frac{|\mathcal{C}|}{|\mathcal{U}[\mathcal{C}]|}\right)^2$, y por tanto, como $\gamma = \text{máx}\{\gamma_1, \gamma_2\}$ tenemos que la probabilidad de éxito total del algoritmo $\mathcal{B}$ es al menos $\gamma \cdot \left(\frac{|\mathcal{C}|}{|\mathcal{U}[\mathcal{C}]|}\right)^2$.

Aplicándolo a la expresión anterior obtenemos finalmente que la probabilidad de éxito de $\mathcal{B}$ es al menos

$$\frac{Adv_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{smp}}(n))}{2m(m^2+5m+3)} \cdot \left(\frac{|\mathcal{C}|}{|\mathcal{U}[\mathcal{C}]|}\right)^2.$$

Este teorema le podemos aplicar al problema de la residuosidad cuadrática del la siguiente manera: Suponga que existe un algoritmo de anillo genérico $\mathcal{A}$ que resuelve el problema de la residuosidad cuadrática en $\mathbb{Z}_n$ (equivalente al problema de pertenencia a un subconjunto tomando $\mathcal{C} = J_n$ y $\mathcal{V} = RQ_n$). Si ademas lo hace con una ventaja $Adv_{(J_n,RQ_n)}(\mathcal{A}^{\mathcal{O}_{smp}}(n))$ al realizar m operaciones de anillo, entonces existe un algoritmo $\mathcal{B}$ que encuentra un factor de n con una probabilidad de al menos

$$\frac{Adv_{(J_n,RQ_n)}(\mathcal{A}^{\mathcal{O}_{smp}}(n))}{2m(m^2+5m+3)} \cdot \left(\frac{|J_n|}{|\mathcal{U}[J_n]|}\right)^2$$

Pero como el cardinal $|J_n|$ depende de si n es un cuadrado en $\mathbb{N}$, tenemos que

$$|J_n| = \begin{cases} \phi(n)/2, & \text{si n no es un cuadrado en } \mathbb{N} \\ \phi(n), & \text{si n es un cuadrado en } \mathbb{N} \end{cases}$$

siendo $\phi(\cdot)$ la función de Euler.
Ademas, $\mathcal{U}[J_n] = \mathbb{Z}_n^*$, por lo tanto $|J_n| \geq \phi(n)/2$ y $|\mathcal{U}[J_n]| = \phi(n)$. Por tanto, podemos aplicar que

$$\left(\frac{|\mathcal{C}|}{|\mathcal{U}[\mathcal{C}]|}\right)^2 = \left(\frac{|J_n|}{|\mathcal{U}[J_n]|}\right)^2 \geq \left(\frac{\phi(n)/2}{\phi(n)}\right)^2 = \frac{1}{4}$$

quedando así que el algoritmo $\mathcal{B}$ encuentra un factor de n con una probabilidad de al menos

$$\frac{Adv_{(J_n,RQ_n)}(\mathcal{A}^{\mathcal{O}_{smp}}(n))}{8m(m^2+5m+3)}$$

En resumen, acabamos de probar como, si conseguimos resolver el problema de la residuosidad cuadrática, podemos llegar a factorizar n. Pero lo hemos visto para un caso en concreto, cuando podemos resolver el problema de la residuosidad cuadrática mediante un algoritmos de anillo genéricos. Veámoslo ahora para un caso más general.

Sabemos que cada residuo cuadrático $x^2 \mod n$ tiene cuatro raíces distintas: $\pm x \mod n$ y $\pm y \mod n$. Pero dado que en el sistema de Cocks asumimos que $n = p \cdot q$ donde $p \equiv q \equiv 3 \mod 4$ entonces vamos a ver que cada residuo cuadrático módulo n tiene exactamente una raíz cuadrada, la cual también es un residuo cuadrático.

Lema 3.4.4 *Sea $n = p \cdot q$ donde p y q son primos distintos tales que $p = 3 \mod 4$ y $q = 3 \mod 4$. Entonces cada residuo cuadrático módulo n tiene exactamente una raíz cuadrada que es un residuo cuadrático.*

Demostración: Siempre que n es producto de dos primos impares distintos, cada residuo cuadrático módulo n tiene cuatro raíces cuadradas, denominémoslas $\pm x$ e $\pm y$.
Dado que $p = 3 \mod 4$, su símbolo de Jacobi satisface: $\left(\frac{-x}{p}\right) = \left(\frac{x}{p}\right) \cdot \left(\frac{-1}{p}\right)$ donde se tiene que $\left(\frac{-1}{p}\right) = (-1)^{\frac{\phi(p)}{2}} \mod p = (-1)^{\frac{p-1}{2}} \mod p = (-1)^{\frac{4k+3-1}{2}} \mod p = (-1)^{\frac{4k+2}{2}} \mod p = (-1)^{2k+1} \mod p = (-1) \mod p = -1$, para algún entero k. Por tanto: $\left(\frac{-x}{p}\right) = -\left(\frac{+x}{p}\right)$.
Del mismo modo, como también $q = 3 \mod 4$, obtenemos $\left(\frac{-x}{q}\right) = -\left(\frac{+x}{q}\right)$.
Por tanto: $\left(\frac{-x}{n}\right) = \left(\frac{-x}{p}\right)\cdot\left(\frac{-x}{q}\right) = \left[-\left(\frac{+x}{p}\right)\right]\cdot\left[-\left(\frac{+x}{q}\right)\right] = \left(\frac{+x}{p}\right)\cdot\left(\frac{+x}{q}\right) = \left(\frac{+x}{n}\right)$. Es decir, $\left(\frac{-x}{n}\right) = \left(\frac{+x}{n}\right)$.
Tras realizar el mismo proceso para las raíces $\pm y$, se obtiene también la igualdad $\left(\frac{-y}{n}\right) = \left(\frac{+y}{n}\right)$.

Pero como $p = 3 \mod 4$ se tiene $\left(\frac{+x}{n}\right) \neq \left(\frac{+y}{n}\right)$. Esto se debe a que $\text{mcd}(x+y, n) = p$ y $\text{mcd}(x-y, n) = q$. Por tanto $\left(\frac{+x}{n}\right) = \left(\frac{-x}{n}\right) \neq \left(\frac{+y}{n}\right) = \left(\frac{-y}{n}\right)$.
Si eliminamos entonces las dos raíces que tienen el símbolo de Jacobi respecto a n igual a -1 (supongamos que estas son $\pm y$), nos quedan las dos raíces que tienen el símbolo de Jacobi respecto a n igual a $+1$ (estas serían $\pm x$). Y solamente una de estas dos raíces, $\pm x$, tiene el símbolo de Jacobi igual a $+1$ respecto a p y a q, dado que como vimos antes $\left(\frac{-x}{p}\right) = -\left(\frac{+x}{p}\right)$ y $\left(\frac{-x}{q}\right) = -\left(\frac{+x}{q}\right)$. Luego solo tal raíz es un residuo cuadrático módulo n. □

En otras palabras, acabamos de ver que calcular los cuadrados módulo n es una función uno a uno en el conjunto RQ_n en sí mismo.

Podemos ver entonces el problema de la residuosidad cuadrática como un generador pseudoaleatorio de números $x^2 \mod n$ en sentido inverso. Es decir, si podemos saber si un número a es un residuo cuadrático, por el lema anterior tenemos que este tiene una única raíz x tal que $x^2 = a \mod n$ y que a su vez x también es un residuo cuadrático.
Podríamos calcular entonces la secuencia $x^2 \mod n$ en dirección inversa partiendo de cualquier residuo cuadrático dado.

Definición 3.4.4 (Generador de la secuencia $x^2 \mod n$, o generador Blum Blum Shub (BBS)) *Sea $n = p \cdot q$ producto de dos primos distintos tales que $p = 3 \mod 4$ y $q = 3 \mod 4$. Sea x_0 cualquier residuo cuadrático en $\mathbb{Z}_n$. La secuencia pseudoaleatoria generada por $x^2 \mod n$ para la entrada x_0 es la secuencia de bits $b_0, b_1, \ldots$ obtenida mediante el calculo $x_{i+1} = x_i^2 \mod n$ y extrayendo el bit de paridad (calculándolo módulo 2).*
La secuencia $x^2 \mod n$ en dirección inversa consiste en calcular $x_0, x_{-1}, x_{-2}, \ldots$ a partir del x_0.

Para llegar a nuestro objetivo (factorizar n) nos queda por ver el siguiente teorema, el cual nos garantiza lo buscado si conseguimos dos raíces distintas y no opuestas de un mismo residuo cuadrático.

Teorema 3.4.2 *Sea n el producto de dos números primos distintos. Si se conocen dos raíces distintas y no opuestas de un mismo residuo cuadrático en $\mathbb{Z}_n$, entonces se puede factorizar n.*

Demostración: Por hipótesis tenemos $x, y \in \mathbb{Z}_n$ tales que $x^2 \equiv y^2 \mod n$ pero que $x \neq \pm y \mod n$. También sabemos que $n = p \cdot q$ con p y q primos distintos.
Como $x^2 \equiv y^2 \mod n$ entonces $x^2 - y^2 \equiv (x-y)(x+y) \equiv 0 \mod n$, es decir, $(x-y)(x+y)$ es un múltiplo de n. Pero no se da ni $x - y \equiv 0 \mod n$ ni $x + y \equiv 0 \mod n$ dado que por hipótesis tenemos $x \neq \pm y \mod n$. Esto significa que o bien $x - y$ o bien $x + y$ es un múltiplo de p, y el otro factor es un múltiplo de q.
Con la ayuda del algoritmo de Euclides se puede hallar el máximo común divisor de $x - y$ y n obteniendo así uno de los dos factores de n. Si repetimos el mismo calculo ahora para $x + y$ y n se obtiene el otro factor primo de n, y con ello la factorización buscada. □

A modo de observación tenemos que, aunque el generador de Blum Blum Shub devuelve solamente el bit de paridad de la raíz cuadrada, esto es suficiente para recuperar todos los bits de la raíz cuadrada en la mayoría de los casos.

En conclusión, la seguridad criptográfica del sistema de Cocks reside en el problema de la residuosidad cuadrática como el generador de bits de Blum Blum Shub. Y como hemos visto que este es equivalente el poder factorizar el número n (el cual se sabe que es un problema intratable para números grandes), podemos afirmar que el criptosistema de Cocks es seguro, tanto como difícil es hallar la factorización de un número o resolver el problema de residuosidad cuadrática.

3.5. Estructura Homomórfica

El principal objetivo de los cifrados homomórficos es poder realizar operaciones directamente sobre los datos cifrados, sin la necesidad de descifrarlos previamente ni de contar con la clave con la que fueron cifrados. De esta manera se logra reducir la cantidad de veces que los datos deben ser descifrados, con la ventaja de disminuir la probabilidad de que estos puedan ser robados y, a su vez, al no necesitar

descifrar el dato para utilizarlo, se puede tener un control mucho más estricto sobre la disponibilidad de la información, garantizando su integridad y confidencialidad.

La idea básica del funcionamiento de estos sistemas se apoya en el concepto de homomorfismo matemático, el cual establece lo siguiente:

Definición 3.5.1 (Homomorfismo) *Dados dos grupos $(A, \cdot)$ y $(B, *)$ y una función $F : A \to B$ que toma elementos del conjunto A y devuelve elementos del conjunto B. F es un homomorfismo si y solo si $F(x \cdot y) = F(x) * F(y)$ para cualquier par de elementos x e y pertenecientes al conjunto A.*

Si analizamos este concepto en el ámbito de la criptografía, el conjunto A puede interpretarse como un texto que se quiere cifrar, el conjunto B como el texto cifrado, ' $\cdot$ ' como las operaciones que se pueden realizar sobre el texto a cifrar, ' $*$ ' como las operaciones que se pueden realizar sobre el dato cifrado y finalmente F como la función de cifrado.

Se ha probado que el sistema IBE de Cocks tiene una estructura que admite propiedades homomórficas sobre sus textos cifrados, si consideramos la representación de estos textos cifrados como polinomios. Introduciré brevemente esta propiedad sin entrar en profundidad. Para ello, nos centraremos en la adición homomórfica de textos cifrados de Cocks.

En 2013, Clear, Hughes y Tewari (CHT) fueron los responsables de la principal observación sobre cómo un texto cifrado de Cocks c puede verse como un polinomio cuadrático módulo a, es decir, como una función lineal, cuya forma es: $2x + c$ (como veremos más adelante). De aquí en adelante denotaremos por c al texto cifrado y por m al texto original (o descifrado).
También crearon un esquema IBE derivado del de Cocks (que llamaron CHT) basado en estas funciones lineales. Su ventaja era la sencillez de sus propiedades homomórficas: la suma homomórfica consistía en multiplicar los dos polinomios entre sí y tomarlos módulo el cuadrado, lo que daba como resulta otro texto cifrado lineal de la forma $ax + b$.
Pero la desventaja que tiene esta variante con respecto al sistema de Cocks es que requiere el doble de ancho de banda (cantidad de datos utilizados expresados en bits), pues los usuarios deben enviar 2 elementos de $\mathbb{Z}_n$ (recordemos $n = p \cdot q$) para describir una función lineal en lugar de solo un (dado que en Cocks $a = 2$).

Partimos de que el texto cifrado de Cocks, cuando se ve como una función lineal como en CHT, es $2x + c$. Entonces, si tenemos una función lineal de un texto cifrado $ax + b$ donde $a = 2$, podría tratarlo como un texto cifrado de Cocks. En este caso solo sería necesario enviar el término independiente de un texto cifrado (pues se supone que el término lineal es 2).
Pero en aquellos casos en los que $a \neq 2$ en una función lineal de la forma $ax + b$, necesitamos alguna forma de normalizarlo para que solo tengamos que enviar el término independiente en nuestro texto cifrado. Si $\left(\frac{a/2}{n}\right) = 1$ entonces simplemente se resolvería multiplicando la función lineal por $\frac{2}{a}$ obteniendo así $2x + \frac{2b}{a}$, lo cual si sería un texto cifrado de Cocks. Pero si $\left(\frac{a/2}{n}\right)$ no es 1 entonces debe de hallarse un modo de modificar la función lineal a una equivalente que tenga como termino lineal al valor 2.

Con ello, tenemos la capacidad de tomar un texto cifrado de función lineal de CHT y convertirlo en un texto cifrado de Cocks.

Como aportación, cabe decir que Marc Joye en 2016 consiguió demostrar que el esquema de Cocks ya tiene propiedades homomórficas sin hacer uso de las modificaciones CHT.

Veamos cómo se obtiene la representación lineal $2x + c$.
A la hora de manejar un texto cifrado de Cocks c, nos puede resultar algo engorroso considerarle junto a sus dos encriptaciones: $c = (c_1, c_2)$ (recordemos que el cifrado c_1 corresponde al caso en el que a es un cuadrado módulo n, y c_2 al caso en el que $-a$ es un cuadrado módulo n (siendo a la clave pública).

Por ello, asumiremos sin perder la generalidad de que a es un cuadrado módulo n. Así, mientras discutimos la estructura matemática detrás de Cocks supondremos que $c = t + \frac{a}{t}$ es un texto cifrado para la clave pública a.

Definimos cómo ver un texto cifrado de Cocks como una función lineal.
Sea $c = t + \frac{a}{t}$ un texto cifrado de Cocks, entonces la función lineal que le representa es $f(x) = 2x + c$.
Veamos de donde se obtiene esta función:
Tomamos un número entero positivo t que cumpla $\left(\frac{t}{n}\right) = m$. Sea $g(x) = (t + x)^2 = t^2 + 2tx + x^2$ y definamos

$$f(x) = \frac{g(x) \mod (x^2 - a)}{t} = \frac{t^2 + a + 2tx}{t} = t + \frac{a}{t} + 2x = c + 2x$$

Para el descifrado, primero se evalúa el polinomio en la clave privada r, y luego se descifra mediante el símbolo de Jacobi: $\left(\frac{f(r)}{n}\right) = \left(\frac{c+2r}{n}\right)$.

El algoritmo básico para la adicción textos cifrados c_1 y c_2 consiste en multiplicar sus formas de función lineal: $ax + b = (2x + c_1) \cdot (2x + c_2)$, y luego aleatorizar la función lineal resultante $ax + b$ hasta que podamos formularla como $2x + c$.

Hablemos ahora de homomorfismos aditivos entre las funciones lineales de textos cifrados.
Dados dos textos cifrados de Cocks como funciones lineales, $f_1(x) = 2x + c_1$ y $f_2(x) = 2x + c_2$, y sea $f_3(x) = f_1(x) \cdot f_2(x) \mod (x^2 - a)$, entonces por la definición de homomorfismo de grupos se tiene que el descifrado de $f_3(x)$ es el producto de los descifrados de $f_1(x)$ y $f_2(x)$, es decir:

$$\left(\frac{f_3(r)}{n}\right) = \left(\frac{f_1(r) \cdot f_2(r)}{n}\right) = \left(\frac{f_1(r)}{n}\right)\left(\frac{f_2(r)}{n}\right)$$

Todo ello asumiendo que $g(r) = g(r) \mod (x^2 - a)$ para todo polinomio $g(x)$.

A Anexo: Algoritmo Del Criptosistema De Cocks

A continuación, expongo el código implementado en Python del criptosistema de Cocks. Se encuentra en el archivo adjunto "*cocks.py*" y reproduce el funcionamiento del criptosistema de Cocks, cifrando y descifrando ficheros que no necesariamente deben ser de texto.

cocks.py

```
# coding: utf-8
# cocks.py

import subprocess
import sys
import pickle

def install(package):
    subprocess.check_call([sys.executable, "-m", "pip", "install", package])

install("pycryptodome")

import random
import os
from Crypto.Util import number
import Crypto.Hash.SHA256

import binascii



#Posibles errores que pueden aparecer:
class BusquedaPrimoError(Exception):
    '''Lanzada si no se encontro el numero primo buscado.'''
    pass

class ClavePublicaError(Exception):
    '''Lanzada si no se ha creado la clave publica.'''
    pass

class ClavePrivadaError(Exception):
    '''Lanzada si no se ha creado la clave privada.'''
    pass

class NoHayNombreFichero(Exception):
    '''Lanzada si no hay un nombre de fichero a leer.'''
    pass

class DatoIncorrectoError(Exception):
    '''Lanzada si el parametro 'dato' de la funcion
    'calculo_longitud_bits' no esta entre los esperados.'''
    pass

class JacobiError(Exception):
    '''Lanzada si el parametro 'n' del simbolo de
```

```
46      Jacobi (a/n) no es impar.'''
47      pass
48
49
50
51  #Funciones auxiliares que usaremos:
52  def Power(a, n, N):
53      '''
54      Función para calcular la potencia de un entero modulo N.
55      '''
56      if n == 0:
57          return 1
58      x = Power(a, n >> 1, N)
59      x = (x * x) %N
60      if n %2 ==1:
61          x = (x * a) %N
62      return x
63
64
65  def extended_gcd(a, b):
66      '''
67      Algoritmo extendido de Euclides, retorna:
68      g: maximo comun divisor
69      n,m: de forma que a*n + b*m = g
70      '''
71      if not b:
72          return (a, 1, 0)
73      else:
74          g, m, n = extended_gcd(b, a %b)
75          return g, n, m-(a//b)*n
76
77
78
79  def calcula_Jacobi(a, n):
80      '''
81      Función que calcula el simbolo de Jacobi (a/n) mediante
82      la aplicacion repetitiva de la ley de reciprocidad cuadratica.
83      '''
84      if n %2 == 0:
85          raise JacobiError(f'Tipo de dato incorrecto para el calculo del simbolo de Jacobi.')
86
87      if a == 0:
88          return 0
89      if a == 1:
90          return 1
91
92      if a %2 == 0:
93          if not ((n+1) %8 == 0) and not ((n-1) %8 == 0): aux = -1
94          else: aux = 1
95          return calcula_Jacobi(a/2, n)*aux
96
97      if not ((a-1) %4 == 0) and not ((n-1) %4 == 0): aux = -1
98      else: aux = 1
99      return calcula_Jacobi(n %a, a)*aux
100
101
102
103 class Cocks:
104     '''Programa que simula el criptosistema de Cocks.
105     Lee un archivo (no tiene porque ser de texto) para encriptar/desencriptar,
106     genera el par de claves publica-privada, y realiza las acciones de encriptar
107     y de desencriptar el archivo.'''
108
109     __slots__ = ('parametro_seguridad', 'nombre_fichero', 'id', 'p', 'q', 'n',
```

```
110                 'clave_publica', 'clave_privada', 'semilla', 'longitud', 'veces')
111     #Nota: el nombre_fichero puede ser cambiado
112
113
114     # CONSTRUCTOR:
115
116     def __init__(self, parametro_seguridad, nombre_fichero = None):
117         self.semilla = random.randint(1, 2**100)
118         self.parametro_seguridad = parametro_seguridad
119         self.nombre_fichero = nombre_fichero
120         self.configuracion_inicial()
121         self.clave_publica = None
122         self.clave_privada = None
123         self.longitud = 100
124         self.veces = 0
125
126
127     # METODOS:
128
129     def primo_random(self, longitud):
130         '''Calcua un numero primo al azar menor que limite y congruente con 3 mod 4'''
131         i = 0
132         encontrado = False
133         while not encontrado and i <= 100:
134             i += 1
135             primo = number.getPrime(longitud, os.urandom)
136             primo_mod = primo %4
137             if primo_mod == 3:
138                 encontrado = True
139         if not encontrado:
140             raise BusquedaPrimoError(f'No se ha encontrado un numero primo congruente'
141                                      + ' con 3 mod 4 con la longitud bits {longitud}')
142         return primo
143
144
145     def calculo_longitud_bits(self, dato):
146         '''Dado el parametro de seguridad, se calcula la longitud en bits que deben de
147         tener los parametros de las claves.'''
148         tabla_longuitudes = {20: 5, 80:166710, 112:458752, 128:768432, 256:7864320}
149         if not type(dato) == int:
150             raise DatoIncorrectoError(f'Tipo de dato incorrecto.')
151
152         if dato <= 20:
153             longitud = tabla_longuitudes[20]
154         elif 20< dato <= 80:
155             longitud = tabla_longuitudes[80]
156         elif 80 < dato and dato <= 112:
157             longitud = tabla_longuitudes[112]
158         elif 112 < dato and dato <= 128:
159             longitud = tabla_longuitudes[128]
160         elif 128 < dato:
161             longitud = tabla_longuitudes[256]
162         self.longitud = longitud
163         return longitud
164
165
166     def configuracion_inicial(self):
167         '''Se calculan los parametros: p, q, n=p*q. Estos son retornados en una tupla.'''
168         longitud = self.calculo_longitud_bits(self.parametro_seguridad)
169         # Tomo p, un primo al azar con p=3 mod 4
170         self.p = self.primo_random(longitud)
171         # Tomo q, un primo al azar con q=3 mod 4 y q!=p
172         valido = False
173         while not valido:
```

```
174             self.q = self.primo_random(longitud)
175             if self.q != self.p:
176                 valido = True
177         self.n = self.p*self.q
178         self.semilla = random.randint(1, self.n) % self.n
179 
180 
181     def calcula_clave_publica(self, id):
182         '''Calcula la clave publica mediante la funcion hash y el parametro id.'''
183         if isinstance(id, str):
184             id = id.encode("UTF8")
185         self.id = id
186         self.clave_publica = Crypto.Hash.SHA256.new(data=id).digest()
187         self.clave_publica = int.from_bytes(self.clave_publica, 'big')
188         self.clave_publica = self.clave_publica %(self.n)
189         # Se extiende la clave publica hasta que sea un residuo cuadratico:
190         while (calcula_Jacobi(self.clave_publica, self.n) != 1):
191             self.clave_publica += 1
192 
193 
194     def calcula_clave_privada(self):
195         '''Calcula la clave privada a partir de la clave publica.'''
196         if self.clave_publica is None:
197             raise ClavePublicaError(f'Aun no se ha creado la clave publica.')
198 
199         aux = ((self.p-1)*(self.q-1)+4) >> 3 #Esto es lo mismo que dividir entre 8
200         self.clave_privada = Power(self.clave_publica, aux, self.n)
201 
202 
203     def BBS(self):
204         '''Funcion que simula el generador pseudoaleatorio de números llamado
205         Blum Blum Shub (generador de la secuencia x^2 mód n).'''
206         resultado = ''
207         self.veces += 1
208         if self.veces > 100:
209             self.semilla = random.randint(1, self.n)
210         for i in range(self.longitud):
211             self.semilla = (self.semilla)**2 % self.n
212             resultado += str(self.semilla % 2)
213         resultado = int(resultado, 2)
214         return resultado % self.n
215 
216 
217     def valor_random(self, x):
218         '''Calcua un numero t al azar menor que el limite y que el simbolo
219         de Jacobi (t/n) sea x donde n=p*q.'''
220         limite = self.n
221         encontrado = False
222         while not encontrado:
223             t = random.randint(1, self.n)
224             simbolo_Jacobi_t = calcula_Jacobi(t, self.n)
225             if x == simbolo_Jacobi_t:
226                 encontrado = True
227         return t
228 
229 
230     def inverso(self, t):
231         '''Calcula el inverso de un numero mediante el algoritmo
232         extendido de Euclides.'''
233         return extended_gcd(self.n, t)[-1]
234 
235 
236     def encriptacion(self, entrada = []):
237         '''Encripta el fichero leido (dado como entrada), y lo hace bit a bit.
```

```
238         Ademas lo escribe en un nuevo fichero de igual nombre que el del original.'''
239         if self.clave_publica is None:
240             raise ClavePublicaError(f'Aun no se ha creado la clave publica.')
241         if not entrada:
242             lista = self.lectura_archivo()
243         else:
244             lista = entrada
245         lista_encriptada = []
246         limite = self.n
247         for m in lista:
248             x = (-1) ** int(m)
249             while True:
250                 t1 = self.valor_random(x)
251                 t2 = self.valor_random(x)
252                 while (t2 == t1):
253                     t2 = self.valor_random(x)
254                 s1 = t1 + (self.clave_publica * self.inverso(t1))
255                 s1 = s1 % self.n
256                 s2 = t2 + (self.clave_publica * self.inverso(t2))
257                 s2 = s2 % self.n
258                 s = extended_gcd(s1 + 2*self.clave_privada, self.n)[0]
259                 s = max(s, extended_gcd(s2 - 2*self.clave_privada, self.n)[0])
260                 if s == 1:
261                     break
262             lista_encriptada.append((s1, s2))
263         #La encriptacion se escribe en el fichero original:
264         f = open(f'{self.nombre_fichero}.enc', 'wb')
265         pickle.dump(lista_encriptada, f)
266         f.close()
267         return lista_encriptada
268
269
270     def desencriptacion(self, lista_encriptada):
271         '''Desencripta el fichero encriptado, y lo hace bit a bit (debe de leerle primero).
272         Ademas lo escribe en un nuevo fichero en formato hexadecimal'''
273         if self.clave_privada is None:
274             raise ClavePrivadaError(f'Aun no se ha creado la clave privada.')
275         lista_desencriptada = []
276
277         #Se estudia cual de las dos encriptaciones se debe desencriptar:
278         if Power(self.clave_privada, 2, self.n) == self.clave_publica: pos = 0
279         else: pos = 1
280         for tupla in lista_encriptada:
281             s = tupla[pos]
282             aux = (s+2*(-1)**(pos)*self.clave_privada) % self.n
283             x = calcula_Jacobi(aux, self.n)
284             if x == 1:
285                 m = 0
286             elif x == -1:
287                 m = 1
288             else:
289                 raise Exception(f"Es cero: {aux}, {s}")
290             lista_desencriptada.append(m)
291         #La desencriptacion se escribe en el fichero que tiene la encriptacion:
292         self.escritura_archivo(lista_desencriptada, self.nombre_fichero)
293         return lista_desencriptada
294
295
296     def lectura_archivo(self):
297         '''
298         Funcion que se encarga de leer un fichero, el cual puede ser
299         de cualquier tipo, y se almacena en bytes.
300         Ademas, transforma esta lectura de bytes a binario y
301         lo devuelve en una lista.'''
```

```
302         if self.nombre_fichero is None:
303             raise NoHayNombreFichero(f'No hay un nombre de fichero a leer.')
304         with open(self.nombre_fichero,'rb') as f:
305             lectura_fichero = f.read() #leido en bytes
306         lectura_binario = []
307         for i in lectura_fichero:
308             temp = bin(int(i))[2:]
309             lectura_binario.append('0'*(8-len(temp)) + temp)
310         lectura_binario = ''.join(lectura_binario)
311         lista_fichero = [int(i) for i in lectura_binario]
312         return lista_fichero
313
314
315     def escritura_archivo(self, lista, nombFichero):
316         '''
317         Funcion que se encarga de escribir un fichero.
318         Lo hace a partir una cadena binaria
319         y el nombre del fichero en el que se escribe.
320         '''
321         cadena = []
322         pos = 0
323         temp = []
324         for i in lista:
325             temp.append(str(i))
326             if pos == 0:
327                 temp = [str(i)]
328                 pos += 1
329             elif pos == 7:
330                 pos = 0
331                 cadena.append(int(''.join(temp), 2))
332             else:
333                 pos += 1
334         cadena = bytearray(cadena)
335         with open(nombFichero, 'wb') as f:
336             f.write(cadena)
```

Además del código anterior, implementé otro código en Python basado en la estructura "unittest". Este código se encarga de realizar encriptaciones y desencriptaciones con el algoritmo de Cocks y comprobar que en el proceso no ocurren errores y que al algoritmo funciona correctamente. Es decir, entre otras cosas, se encarga de comprobar que el mensaje que pasa por ambos procesos del criptosistema de Cocks corresponde con el original dado al algoritmo.

cocks_unittest.py

```
1  # Cocks_unittest.py
2
3
4  import unittest
5  import cocks
6
7
8  '''
9  Test que prueba el codigo Cocks.py
10 '''
11 class Cocks_unittest(unittest.TestCase):
12
13     def test_Jacobi(self):
14         '''Comprueba que la funcion que calcula el simbolo de Jacobi (a/n)
15         mediante la aplicacion repetitiva de la ley de reciprocidad cuadratica
16         funcione correctamente. Para ello se comparan sus resultados con otros
17         preiamente calculados.'''
18         soluciones = [(45, 7, -1), (2585, 245, 0), (852, 33, 0), (13795, 3569, 1),
```

```
19                      (3478, 11, -1), (333333, 45, 0), (82736575, 73651, 1),
20                      (86, 737, 1), (236, 68769, -1), (55, 69, 1)]
21                      #contiene 3-tuplas: (a, n, solucion)
22          for valor in soluciones:
23              a = valor[0]
24              n = valor[1]
25              resultado = cocks.calcula_Jacobi(a, n)
26              self.assertEqual(resultado, valor[2])
27
28
29      def test_create(self):
30          '''Comprueba que el constructor de la clase Cocks funcione.'''
31          s = cocks.Cocks(11, 'prueba.txt')
32          self.assertEqual(s.parametro_seguridad, 11)
33          self.assertEqual(s.nombre_fichero, 'prueba.txt')
34          self.assertIsNotNone(s.p)
35          self.assertIsNotNone(s.q)
36          self.assertIsNotNone(s.n)
37
38
39      def test_errores(self):
40          '''Comprueba que se lanzan los errores cuando es necesario.'''
41          s = cocks.Cocks(11, 'prueba.txt')
42          with self.assertRaises(cocks.ClavePublicaError):
43              s.calcula_clave_privada()
44          with self.assertRaises(cocks.ClavePublicaError):
45              s.encriptacion()
46          with self.assertRaises(cocks.ClavePrivadaError):
47              s.desencriptacion([1,0,1,0])
48
49          ss = cocks.Cocks(11)
50          with self.assertRaises(cocks.NoHayNombreFichero):
51              ss.lectura_archivo()
52
53          #Comprobar: mal introducido el parametro de seguridad en el constructor
54          with self.assertRaises(cocks.DatoIncorrectoError):
55              cocks.Cocks('casa')
56
57
58      def test_claves(self):
59          '''Comprueba que la clave privada al cuadrado es igual a la clave
60          publica o a menos la clave publica.'''
61          s = cocks.Cocks(11, 'prueba.txt')
62          s.calcula_clave_publica(b'Nombre')
63          s.calcula_clave_privada()
64          r2 = abs(s.clave_privada*s.clave_privada) %s.n
65          a = abs(s.clave_publica) %s.n
66          self.assertTrue(r2 == a)
67
68
69      def test_funciona(self):
70          '''Asegurar que el archivo tras ser encriptado y luego desencriptado,
71          es igual al original.'''
72          s = cocks.Cocks(11, 'prueba.txt')
73          original = s.lectura_archivo()
74          s.calcula_clave_publica(b'Nombre')
75          s.calcula_clave_privada()
76          encriptado = s.encriptacion()
77          desencriptado = s.desencriptacion(encriptado)
78          self.assertEqual(original, desencriptado)
79
80
81  if __name__ == '__main__':
82      unittest.main()
```

Ambos códigos son proporcionados adjuntos a la memoria del trabajo, bajo los nombres "*cocks.py*" y "*cocks_unittest.py*".

B Anexo: Operadores a nivel de bit

Una operación bit a bit opera sobre números binarios a nivel de sus bits individuales. Es una acción primitiva rápida, soportada directamente por los procesadores. Veremos las operaciones básicas, así como la transformación entre las representaciones decimal y binaria de un número.

El sistema base 2 o binario está compuesto por dos elementos que son 0 y 1. En él es posible realizar las operaciones matemáticas básicas, así como pasar al sistema decimal y viceversa. Veamos su funcionamiento:

1. **De decimal a binario:** Dado un número en el sistema decimal a, si se quiere escribir este en el sistema binario simplemente se debe de dividir a entre 2 y obtener su cociente y su resto. Tras esta primera operación, se vuelve a realizar ahora dividiendo el cociente anterior entre 2 y obteniendo de nuevo su cociente y su resto. Se repite este procesa hasta obtener un cociente nulo. Por tanto, ordenando los restos, del último obtenido al primero, obtenemos el número binario que buscamos.
   Un ejemplo de ello se muestra en la ilustración 8.

```
131 dividido entre 2 da 65 y el resto es igual a 1
 65 dividido entre 2 da 32 y el resto es igual a 1
 32 dividido entre 2 da 16 y el resto es igual a 0
 16 dividido entre 2 da 8  y el resto es igual a 0
  8 dividido entre 2 da 4  y el resto es igual a 0
  4 dividido entre 2 da 2  y el resto es igual a 0
  2 dividido entre 2 da 1  y el resto es igual a 0
  1 dividido entre 2 da 0  y el resto es igual a 1
   -> Ordenamos los restos, del último al primero que estan en color Azul:
10000011₂
```

Ilustración 7: Ejemplo paso de sistema decimal a binario. [2]

2. **De binario a decimal:** Si por el contrario se quiere pasar un número b dado en el sistema binario a su forma en el sistema decimal, se debe de: empezando a leer el número b por el lado derecho, se toma su primer dígito (llamémosle c) y elevamos el dos a este, es decir, 2^c y el resultado multiplicarlo por el dígito, c. Tras este cálculo realizamos el mismo proceso con la cifra situada a la izquierda de c hasta realizarlas todas. El número en el sistema decimal es el resultado de sumar todas las operaciones anteriores.
   Es decir, sea $b = c_3c_2c_1c_0$ un número en binario, en el sistema decimal equivaldría al resultado de la operación $c_0 \cdot 2^{c_0} + c_1 \cdot 2^{c_1} + c_2 \cdot 2^{c_2} + c_3 \cdot 2^{c_3}$.

3. **Suma en binario:** Las reglas que tener en cuente en la operación suma entre dos bits son: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$, y $1 + 1 = 0$ donde "nos llevaríamos una". Con ello, para sumar dos números en sistema binario bastaría con realizar la operación como en el sistema decimal donde vamos aplicando estas reglas.
   A continuación muestro un ejemplo:

Dada la suma:

$$\begin{array}{r} 100110101 \\ +\ 11010101 \\ \hline \end{array}$$

Comenzamos a sumar desde la derecha: 1 + 1 = 0, entonces escribimos 0 en la fila del resultado y llevamos 1. A continuación se suma la llevada a la siguiente columna: 1 + 0 + 0 = 1. Seguimos hasta terminar todas las columnas obteniendo:

$$\begin{array}{r} 100110101 \\ +\ 11010101 \\ \hline 1000001010 \end{array}$$

4. **Resta en binario:** En cuanto a la reta de bits, las reglas que hay que tener en cuenta son: $0-0=0$, $1-1=0$, $1-0=1$, y $0-1=1$ donde "nos llevaríamos una" al igual que en la resta decimal. Como en la suma, para restar dos números en sistema binario se realizar la operación como en el sistema decimal, pero aplicando las reglas anteriores.
   Un ejemplo de esto sería:

Dada la resta:

$$\begin{array}{r} 11011001 \\ -\ 10101011 \\ \hline \end{array}$$

Comenzamos a restar desde la derecha: 1 - 1 = 0, entonces escribimos 0 en la fila del resultado. A continuación, se resta la siguiente columna: 0 - 1 = 1 y nos llevamos 1 a la siguiente columna. Así, el siguiente paso tenemos una llevada y seria: 0 - (0 + 1) = 0 - 1 = 1 y volvemos a llevarnos 1. Seguimos hasta terminar todas las columnas obteniendo:

$$\begin{array}{r} 11011001 \\ -\ 10101011 \\ \hline 00101110 \end{array}$$

5. **Multiplicación en binario:** La multiplicación de números en binario es igual que en números decimales, aunque se lleva cabo con más sencillez ya que el 0 multiplicado por cualquier número da 0 y el 1 es el elemento neutro del producto.
   Por ejemplo:

Dada la multiplicación:

$$\begin{array}{r} 10110 \\ \times\ 1001 \\ \hline \end{array}$$

Consiste en multiplicar como en el sistema decimal: se procede a multiplicar cada dígito del segundo elemento (empezando de derecha a izquierda) con los del primero, donde los resultados parciales de cada multiplicación se ponen en fila, pero corriendo la posición de estos en cada operación. Posteriormente se realiza la suma de las filas resultantes. Obtenemos así el siguiente proceso y resultado:

$$\begin{array}{r} 10110 \\ \times\ 1001 \\ \hline 10110 \\ 00000 \\ 00000 \\ 10110 \\ \hline 11000110 \end{array}$$

6. **División en binario:** Por último, la división de números en el sistema binario también es muy parecida a la del sistema decimal, solo hay que tener en cuenta que, a la hora de hacer las operaciones dentro de la división, estas deben ser realizadas en binario.
Muestro un ejemplo del proceso:

Dada la división:

$$1110 \mid 10$$

Los pasos de esta división siguiendo el procedimiento del sistema decimal sería el siguiente:

$$\begin{array}{rl} 1\,1\,1\,0 & \mid 10 \\ \underline{1\,0} & \quad 111 \\ 0\,1\,1 & \\ \underline{1\,0} & \\ 0\,1\,0 & \\ \underline{1\,0} & \\ 0 & \end{array}$$

Referencias

[1] Descripción general de Unicode. En *Guía para entornos de idiomas internacionales*. Oracle Solaris 11 Information Library (Español), Marzo 2012.
https://docs.oracle.com/cd/E26921_01/html/E27143/glmgn.html

[2] Sistema de numeración en base 2 (binarios). Grupo Alquerque, 2012.
http://www.grupoalquerque.es/ferias/2012/archivos/s-n_nuevos/s-n_base_2.pdf

[3] Operaciones con números binarios. IES Chamoso Lamas, 2013.
http://centros.edu.xunta.es/iesmanuelchamosolamas/electricidade/fotos/numeracion.htm

[4] Operadores de desplazamiento bit a bit. Microsoft, 2018.
https://docs.microsoft.com/es-es/cpp/c-language/bitwise-shift-operators

[5] Archivo: Public-key-infrastructure.svg - Wikimedia Commons. Wikimedia Commons, 2020.
https://commons.wikimedia.org/wiki/File:Public-Key-Infrastructure.svg#/media/Archivo:Public-Key-Infrastructure.svg

[6] RSA numbers - Wikipedia. Wikipedia contributors, 2021.
https://en.wikipedia.org/wiki/RSA_numbers#RSA-250

[7] Crypto.hash. Generado por Epydoc 3.0.1, Mayo 2012.
https://www.dlitz.net/software/pycrypto/api/current/Crypto.Hash-module.html

[8] ASCII Table and description - ASCII character codes and html, octal, hex and decimal chart conversion.
http://www.asciitable.com/

[9] I. G. F. F. ARTEAGA. Criptoanálisis a la función hash de un sistema operativo nivel C1 empleando un clúster HPC con software de uso libre. Tesis, Instituto Tecnológico de Acapulco (México), Noviembre 2019.
http://www.itacapulco.net/depi/wp-content/uploads/2020/01/Tesis-Gaddiel-Flores-Arteaga.pdf

[10] L. Blum, M. Blum y M. Shub. Comparison of two pseudo-random number generators. En *Advances in Cryptology: Proceedings of CRYPTO '82*, pp. 61–78. Plenum, 1982.

[11] D. Boneh y M. Franklin. Identity-Based Encryption from the Weil Pairing. En *Advances in Cryptology — CRYPTO 2001*, pp. 213–229. Springer Berlin Heidelberg, 2001.
https://link.springer.com/content/pdf/10.1007%2F3-540-44647-8_13.pdf

[12] J. Booher. Square roots in finite fields and quadratic nonresidues. Palo Alto, CA: Stanford University, 2012.
https://www.math.canterbury.ac.nz/~j.booher/expos/sqr_qnr.pdf

[13] J. C. G. Diaz. *Criptografía: historia de la escritura cifrada*. Editorial Complutense, 1995.
https://books.google.es/books?id=9jUjYLjJZCAC

[14] B. Donohue. ¿Qué Es Un Hash Y Cómo Funciona? Kaspersky daily, Abril 2014.
https://latam.kaspersky.com/blog/que-es-un-hash-y-como-funciona/2806/

[15] L. S. B. Escalona y I. L. V. Inclán. Funciones resúmenes o hash. *Telemática*, 10(1), 2012.
https://revistatelematica.cujae.edu.cu/index.php/tele/article/view/52

[16] P. M. A. Garau. *Diseño, implementación y optimización de algoritmos criptográficos de generación de aleatorios y factorización de enteros.* Tesis, Universidad de Murcia, Septiembre de 2003.
https://doi.org/10.31428/10317/787

[17] R. Guerequeta y A. Vallecillo. *LA COMPLEJIDAD DE LOS ALGORITMOS*, Capítulo 1, pp. 1–57. Universidad de Málaga, 2019.
http://190.57.147.202:90/jspui/bitstream/123456789/451/1/tecnicas_de_diseno_de_algoritmos.pdf

[18] J. Herranz. Cifrado basado en identidades simétrico. En Josep M. Miret - Francesc Sebé, editor, *Actas de la XVI Reunión Española sobre Criptología y Seguridad de la Información (RECSI)*, pp. 45–49. Universidad Politecnica de Cataluña, Abril 2021.
http://www.recsi2020.udl.cat/proceedings

[19] A. V. Huerta. 55. Criptografía: historia. Teoría de números. Sistemas de clave pública. Sistemas de clave privada. *Colegio Oficial de Ingeniería Informática de la Comunidad Valenciana (COIICV)*, 2003.
http://shutdown.es/alfa55.pdf

[20] T. Jager y J. Schwenk. The generic hardness of subset membership problems under the factoring assumption. *IACR Cryptol. ePrint Arch.*, 2008:482, 2008.
https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.215.7655&rep=rep1&type=pdf

[21] D. Kundro. Criptografía homomórfica: un paradigma de cifrado cada vez más cercano. WeLiveSecurity, Sep 2019.
https://www.welivesecurity.com/la-es/2019/09/04/criptografia-homomorfica-paradigma-cifrado-c

[22] R. LaVigne. Simple homomorphisms of cocks ibe and applications. IACR Cryptology ePrint Archive, 2016.
https://eprint.iacr.org/2016/1150.pdf

[23] A. K. Lenstra y E. R. Verheul. Selecting Cryptographic Key Sizes. *Journal of Cryptology*, 14(4): 255–293, Agosto 2001.
https://doi.org/10.1007/s00145-001-0009-4

[24] J. Lopez, A. Mana, J. A. Montenegro, J. J. Ortega, y J. M. Troya. Aspectos de Implementación de una Infraestructura de Clave Pública Distribuida. En *Simposio Español de Informática Distribuida (SEID'00)*, pp. 313–320, Orense, Espa, Septiembre 2000.
https://www.nics.uma.es/pub/papers/JavierLopez2000.pdf

[25] L. Martin. *Introduction to Identity-Based Encryption.* Arthech House, 2008.

[26] S. Noriega. Operaciones matemáticas con números binarios. Facultad de Ingeniería, Universidad Nacional de La Plata, 2003.
https://catedra.ing.unlp.edu.ar/electrotecnia/islyd/apuntes/opmatematicas2003.pdf

[27] J. C. S. Peña, Y. N. Musa, R. S. Lima, y A. R. Suárez. Propuesta de aplicación de un sistema de Infraestructura de Clave Pública (Public Key Infrastructure "PKI") y los Certificados Digitales en la trazabilidad de productos agrícolas. *Revista Ciencias Técnicas Agropecuarias*, 18:75–78, 2009.
https://www.redalyc.org/pdf/932/93212367015.pdf

[28] J. Ramió Aguirre. *Libro electrónico de seguridad Informática y Criptografía*, volumen Versión 4.1. Dpto. de Publicaciones E.U.I. Universidad Politécnica de Madrid, Marzo 2006.
http://www.criptored.upm.es/guiateoria/gt_m001a.htm

[29] S. Sánchez, P. Domínguez y L. Velásquez. Hashing: Técnicas y hash para la protección de datos. *Universidad Tecnológica de Panamá, Grupo de Investigación*, 2018.
http://www.laccei.org/LACCEI2018-Lima/student_Papers/SP96.pdf

[30] E. Schaeffer. Complejidad computacional de problemas y el análisis y diseño de algoritmos. *Publicación en línea*, Actualizado el 18 de marzo de 2020.
https://elisa.dyndns-web.com/teaching/aa/pdf/aa.pdf

[31] P. X. Solana. Antecedentes y perspectivas de estudio en historia de la criptografía. B.S. tesis, Universidad Carlos III de Madrid, 2009.
https://e-archivo.uc3m.es/bitstream/handle/10016/6173/PFC_Patricia_Xifre_Solana.pdf?sequence=1

[32] Y. Urquijo Morales y A. Orellana García. Esquema de confianza basado en infraestructura de clave pública (PKI) para el intercambio de información clínica electrónica en el sistema XAVIA HIS. *Revista Cubana de Informática Médica*, 12, 12 2020.
http://scielo.sld.cu/pdf/rcim/v12n2/1684-1859-rcim-12-02-e390.pdf

[33] A. D. O. Vintimilla, J. A. C. Zenteno y F. J. B. Burgos. ESTANDARES CRIPTOGRAFICOS APLICADOS A LA INFRAESTRUCTURA DE CLAVE PUBLICA DE AMERICA DEL SUR. *3C Tecnología_Glosas de innovación aplicadas a la pyme*, 6(3):14–32, Sept. 2017.
https://www.3ciencias.com/wp-content/uploads/2017/09/ART-2-2.pdf

[34] G. Yesid Díaz y J. M. C. Lovelle. Análisis de la función hash criptográfica en cadenas de bloques y su impacto en la seguridad de transacciones de datos. *Redes de Ingeniería*, 9(2):82–87, Dic. 2018.
https://revistas.udistrital.edu.co/index.php/REDES/article/view/14383/15147

[35] R. Yifeng. Codificación de caracteres: ASCII, Unicode y UTF-8. Programador clic, 2018.
https://programmerclick.com/article/4424428167/