TESIS PUCP

PONTIFICIA **UNIVERSIDAD CATÓLICA** DEL PERÚ

Esta obra ha sido publicada bajo la licencia Creative Commons
Reconocimiento-No comercial-Compartir bajo la misma licencia 2.5 Perú.
Para ver una copia de dicha licencia, visite
http://creativecommons.org/licenses/by-nc-sa/2.5/pe/

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

PONTIFICIA UNIVERSIDAD CATOLICA DEL PERU

ESCUELA DE GRADUADOS
MAESTRIA DE INFORMATICA

TESIS PARA OPTAR EL GRADO DE MAGISTER EN INFORMATICA

KATIA REGINA LEON LOMPARTE

ENCRIPTACION RSA DE ARCHIVOS DE TEXTO

LIMA – PERU
2005

SOME RIGHTS RESERVED

Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

A mis padres

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

INDICE

INTRODUCCION 5

CAPITULO I CRIPTOGRAFIA 8

Historia 9

Conceptos básicos 15

Criptografía de Clave Pública 21

Criptografía de Clave Secreta 29

CAPITULO II ENCRIPTACION RSA 45

Generación de Claves 46

Encriptación de Mensajes 48

Decriptación de Mensajes 49

Matemática del RSA 51

CAPITULO III DISEÑO E IMPLEMENTACION 68

Algoritmo del RSA 69

Claves RSA 70

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Encriptación RSA 74

Desencriptación RSA 74

Implementación 75

Requerimientos 76

Herramienta utilizada 77

Organización interna..................... 77

Visión General del Sistema 78

Descripción del Algoritmo RSA 81

Estructura de datos..................... 83

Estándar de codificación..................... 86

Descripción del Clases del Sistema..................... 88

ANEXOS 123

Manual de Usuario..................... 124

Estándar de codificación JAVA..................... 135

Listado de Programas 156

BIBLIOGRAFÍA 194

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

INTRODUCCION

Este trabajo de tesis fue iniciado en agosto del 2004, en base a un programa en lenguaje C elaborado en el curso Seminario de Tesis. Desde ese programa en C hasta la versión actual en lenguaje java, han habido muchos cambios, mejoras, revisiones y correcciones que hacen que el sistema que ahora se presenta sea un programa completamente nuevo.

Los algoritmos de encriptación tienen una base fuertemente matemática y de esto no escapa el RSA, poder manejar estos conceptos dentro de este programa no ha sido precisamente de las tareas más sencillas, sino por el contrario demandó mucho tiempo no solo en entender el funcionamiento del algoritmo sino también en la implementación y pruebas del mismo.

Siendo el algoritmo implementado una técnica de encriptación muy elaborada, empezaremos esta introducción comentando muy brevemente sobre el nacimiento de la criptografía. Los códigos secretos han sido utilizados desde la antigüedad para enviar mensajes seguros, por ejemplo en tiempo de guerra o de tensiones diplomáticas. Hoy día se guarda, con frecuencia, información delicada de naturaleza médica o financiera en los ordenadores, y es importante mantenerla en secreto.

Muchos códigos están basados en teoría de números. Uno muy simple es sustituir cada letra del alfabeto por la siguiente o por tres letras después. Estos

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

códigos son fáciles de romper pues se pueden probar todos los posibles valores hasta obtener un mensaje comprensible, o podemos comparar las letras más frecuentes en el mensaje con las que se saben que son más frecuentes en la lengua original.

Históricamente primero han aparecido los códigos basados en una única clave conocidos como sistemas de clave privada. Estos tienen el inconveniente de que el remitente y el receptor deben estar de acuerdo, con anterioridad, en los valores de la clave. Teniendo en cuenta que por seguridad deberán cambiar la clave de vez en cuando, ¿de qué modo podrían hacerlo de una manera segura?

Esta dificultad se supera con la aparición de los sistemas criptográficos de clave pública que permite a todos los remitentes utilizar la clave pública del emisor para la encriptación de un mensaje que sólo el destinatario podrá recuperar pues será el único poseedor de la clave privada.

Desde la Segunda Guerra Mundial ha habido una crecimiento espectacular de la tecnología criptográfica, aunque la mayor parte de estos secretos se mantenían y aún en la actualidad, se mantienen en secreto. Financiados en su mayoría por la NSA (Agencia Nacional de Seguridad de los EE.UU.) han sido tratadas como secretos militares. Actualmente las universidades también llevan a cabo estudios criptográficos y han tenido diversos resultados exitosos entre los que se cuenta la creación de los sistemas de clave pública.

En este trabajo se presentará el método desarrollado en 1978 por R. L. Rivest, A. Shamir y L. Adleman y que es conocido como sistema criptográfico RSA por las iniciales de sus autores. Basa su seguridad en la dificultad de factorizar números primos muy grandes aunque como todo sistema de encriptación de clave

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

pública, el RSA puede estar sujeto a ataques con el fin de obtener el mensaje original o descubrir la clave privada.

También conoceremos como las claves pueden protegerse para que no sean vulnerables a estos ataques y sean casi inviolables. Uno de los conceptos más importantes sobre el cual también reside la seguridad del sistema, es la longitud de los números primos que dan lugar a las claves. Actualmente se estima que números de 1024 o 2048 bits son inviolables (128 o 256 dígitos decimales)

El primer capítulo de este trabajo está dedicado a la evolución histórica, conceptos de criptografía, tipos de criptografía y métodos. El segundo capítulo nos describe en detalle el método de encriptación RSA con la aplicación del estándar de criptografía RSA llamado PKCS # 1.

En el tercer capítulo de este trabajo de Tesis, se describe la implementación de los algoritmos RSA en Java en un sistema llamado Sistema de Encriptación RSA. Cuenta con una interfase que permite la interacción del usuario con la generación de claves pública y privada, así como la encriptación y decriptación de archivos. Las consideraciones de seguridad recomendadas por la diversa bibliografía consultada, también ha sido implementada.

Como última parte se incluye la sección de Anexos en la que se encontrarán el Manual de Estándares utilizados, el Manual de usuario así como el listado completo del programa.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

CAPITULO I

CRIPTOGRAFIA

El origen de la criptografía se remonta sin duda a los orígenes del hombre, desde que aprendió a comunicarse. Entonces, tuvo que encontrar medios de asegurar la confidencialidad de una parte de sus comunicaciones. Incluso en el antiguo Egipto, la escritura jugó a veces ese papel. El principio básico de la criptografía es mantener una comunicación entre dos personas de forma que sea incomprensible para el resto.

Fue considerada un arte, hasta que Shannon publicó en 1949 la "Teoría de las comunicaciones secretas", la cual fue aplicada por el NBS (National Bureau of Standars) de EEUU para desarrollar el sistema criptográfico DES (Data Encryption Standard). Entonces la criptografía empezó a ser considerada una ciencia aplicada, debido a su relación con otras ciencias, como la estadística, la teoría de números, la teoría de la información y la teoría de la complejidad computacional.

Ahora bien, la criptografía corresponde sólo a una parte de la comunicación secreta. Si se requiere secreto para la comunicación, es porque existe desconfianza o peligro de que el mensaje transmitido sea interceptado por un enemigo. Este enemigo, si existe, utilizará todos los medios a su alcance para

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

descifrar esos mensajes secretos mediante un conjunto de técnicas y métodos que constituyen una ciencia conocida como criptoanálisis. Al conjunto de ambas ciencias, criptografía y criptoanálisis se le denomina criptología.

Historia

Del Antiguo Egipto a la era digital, los mensajes cifrados han jugado un papel destacado en la Historia. Arma de militares, diplomáticos y espías, son la mejor defensa de las comunicaciones y datos que viajan por Internet.

Esclavos con textos grabados en su cuero cabelludo, alfabetos de extraños símbolos, escritos de tinta simpática, secuencias interminables de números... Desde la Antigüedad, el hombre ha hecho gala de su ingenio para garantizar la confidencialidad de sus comunicaciones. La criptografía (del griego kryptos, "escondido", y graphein, "escribir"), el arte de enmascarar los mensajes con signos convencionales, que sólo cobran sentido a la luz de una clave secreta, nació con la escritura. Su rastro se encuentra ya en las tablas cuneiformes, y los papiros demuestran que los primeros egipcios, hebreos, babilonios y asirios conocieron y aplicaron sus inescrutables técnicas, que alcanzan hoy su máxima expresión gracias al desarrollo de los sistemas informáticos y de las redes mundiales de comunicación.

Entre el Antiguo Egipto *e* Internet, los criptogramas han protagonizado buena parte de los grandes episodios históricos y un sinfín de anécdotas. Existen mensajes cifrados entre los 64 artículos del Kamasutra, el manual erótico hindú del Vatsyayana, abundan en los textos diplomáticos, pueblan las órdenes militares

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

en tiempos de guerra y, por supuesto, son la esencia de la actividad de los espías.

Los métodos clásicos

Los espartanos utilizaron, en el 400 a.c., la Scitala (o "escitalo"[1]), que puede considerarse el primer sistema de criptografía por transposición, es decir, que se caracteriza por enmascarar el significado real de un texto alterando el orden de los signos que lo conforman. Los militares de la ciudad griega escribían sus mensajes sobre una tela o cuero que envolvían sobre una vara. El mensaje sólo podía leerse cuando se enrollaba sobre un bastón del mismo grosor, que poseía el destinatario lícito.

El método de la scitala era extremadamente sencillo, como también lo era el que instituyó Julio César, basado en la sustitución de cada letra por la que ocupa tres puestos más allá en el alfabeto.

En los escritos medievales sorprenden términos como Xilef o Thfpfklbctxx. Para esconder sus nombres, los copistas empleaban el alfabeto zodiacal, formaban anagramas alterando el orden de las letras (es el caso de Xilef, anagrama de Félix) o recurrían a un método denominado fuga de vocales, en el que éstas se sustituían por puntos o por consonantes arbitrarias (Thfpfklbctxx por Theoflactus).

La criptografía resurgió en la Europa de la Edad Media, impulsada por las intrigas del papado y las ciudades-estado italianas. Fue un servidor del Papa Clemente VII, Grabiele de Lavinde, quien escribió el primer manual sobre la materia en el viejo continente.

[1] Loidreau, Piere, tomado de http://ldp.rtin.bz/linuxfocus/Castellano/May2002/article243.shtml el 11 de abril del 2005.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

En 1466, León Battista Alberti, músico, pintor, escritor y arquitecto, concibió el sistema polialfabético que emplea varios abecedarios, saltando de uno a otro cada tres o cuatro palabras. El emisor y el destinatario han de ponerse de acuerdo para fijar la posición relativa de dos círculos concéntricos, que determinará la correspondencia de los signos.

Un siglo después, Giovan Battista Belaso de Brescia instituyó una nueva técnica. La clave, formada por una palabra o una frase, debe transcribirse letra a letra sobre el texto original. Cada letra del texto se cambia por la correspondiente en el alfabeto que comienza en la letra clave.

Pero los métodos clásicos distan mucho de ser infalibles. En algunos casos, basta hacer un simple cálculo para desentrañar los mensajes ocultos. Si se confronta la frecuencia habitual de las letras en el lenguaje común con la de los signos del criptograma, puede resultar relativamente sencillo descifrarlo. Factores como la longitud del texto, el uso de más de una clave o la extensión de esta juegan un papel muy importante, así como la intuición, un arma esencial para todo criptoanalista.

En el siglo XIX, Kerchoffs estableció los principios de la criptografía moderna. Uno de los principios establece que la seguridad de un sistema de cifrado no reside más que en la clave y no en el procedimiento de cifrado.

En lo sucesivo, concebir sistemas criptográficos debía responder a esos criterios. Sin embargo, todavía les faltaba a esos sistemas unos cimientos matemáticos suficientes que proveyeran utilidades para medir y cuantificar su resistencia a eventuales ataques y, por qué no, encontrar el “Santo Grial” de la criptografía: el sistema incondicionalmente seguro.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

En 1948 y 1949 dos artículos de Claude Shannon, "Teoría Matemática de la Comunicación" y sobre todo "La Teoría de la Comunicación de los Sistemas Secretos" dieron los cimientos científicos a la criptografía borrando tanto vanas promesas como falsos prejuicios. Shannon probó que el cifrado de Vernam introducido algunas decenas de años antes –todavía llamado one-time pad—era el único sistema incondicionalmente seguro. Sin embargo el sistema era impracticable. Es por lo que hoy en día la evaluación de la seguridad de un sistema se interesa principalmente en la seguridad computacional. Se dice que un sistema de cifrado de clave secreta es seguro si ningún ataque conocido de menor complejidad que la búsqueda exhaustiva sobre el espacio de claves ofrece mejores resultados que ésta.

Enigma, la victoria aliada

El siglo XX ha revolucionado la criptografía. Retomando el concepto de las ruedas concéntricas de Alberti, a principios de la centuria se diseñaron teletipos equipados con una secuencia de rotores móviles. éstos giraban con cada tecla que se pulsaba. De esta forma, en lugar de la letra elegida, aparecía un signo escogido por la máquina según diferentes reglas en un código polialfabético complejo. Estos aparatos, se llamaron traductores mecánicos. Una de sus predecesoras fue la Rueda de Jefferson, el aparato mecánico criptográfico más antiguo que se conserva.

La primera patente data de 1919, y es obra del holandés Alexander Koch, que comparte honores con el alemán Arthur Scherbius, el inventor de Enigma una máquina criptográfica que los nazis creyeron inviolable, sin saber que a partir de 1942, propiciaría su derrota. En efecto, en el desenlace de la contienda, hubo un

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

ET LUX IN TENEBRIS LUCET MCMXVII PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

factor decisivo y apenas conocido: los aliados eran capaces de descifrar todos los mensajes secretos alemanes.

Una organización secreta, en la que participó Alan Turing, uno de los padres de la informática y de la inteligencia artificial, había logrado desenmascarar las claves de Enigma, desarrollando más de una docena de artilugios -las bombas- que desvelaban los mensajes cifrados. La máquina alemana se convertía así en el talón de Aquiles del régimen, un topo en el que confiaban y que en definitiva, trabajaba para el enemigo.

Los códigos de la versión japonesa de Enigma (llamados Purple, violeta) se descifraron en el atolón de Midway. Un grupo de analistas, dirigidos por el comandante Joseph J. Rochefort, descubrió que los nipones señalaban con las siglas AF su objetivo. Para comprobarlo, Rochefort les hizo llegar este mensaje: "En Midway se han quedado sin instalaciones de desalinización". Inmediatamente, los japoneses la retransmitieron en código: "No hay agua potable en AF". De esta forma, el almirante Nimitz consiguió una clamorosa victoria, hundiendo en Midway cuatro portaviones japoneses.

Mientras los nazis diseñaron Enigma para actuar en el campo de batalla, los estadounidenses utilizaron un modelo llamado Sigaba y apodado por los alemanes como "la gran máquina". Este modelo, funcionó en estaciones fijas y fue el único artefacto criptográfico que conservó intactos todos sus secretos durante la guerra.

La existencia de Enigma y el hecho de que los aliados conociesen sus secretos fueron, durante mucho tiempo, dos de los secretos mejor guardados de la II Guerra Mundial. ¿La razón? Querían seguir sacándole partido tras la guerra potenciando su uso en diversos países, que, al instalarla, hacían transparentes

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

sus secretos.

Finalizada la contienda, las nuevas tecnologías electrónicas y digitales se adaptaron a las máquinas criptográficas. Se dieron así los primeros pasos hacia los sistemas criptográficos más modernos, mucho más fiables que la sustitución y transposición clásicas. Hoy por hoy, se utilizan métodos que combinan los dígitos del mensaje con otros, o bien algoritmos de gran complejidad. Un ordenador tardaría 200 millones de años en interpretar las claves más largas, de 128 bits.

Criptogramas en la red

Alertado por las posibilidades que las innovaciones tecnológicas abrían, el Gobierno estadounidense intentó, en los años cincuenta, introducir el DES (Data Encryption Standard), un sistema desarrollado por la National Security Agency (NSA). El objetivo era que todos los mensajes cifrados utilizaran el DES; un intento de control que pocos aceptaron.

No ha sido el único. Philip Zimmermann, un criptógrafo aficionado, levantó hace unos años las iras del Gobierno estadounidense. Su delito fue idear un sistema de codificación aparentemente inviolable, el PGP (Pretty Good Privacy), y distribuirlo por las redes de comunicación para que cualquiera pudiera utilizarlo. Algo que no podía agradar a quienes ven en la criptografía un arma de doble filo, útil para los gobiernos y funesta en manos de terroristas y delincuentes.

Con la expansión de la red se ha acelerado el desarrollo de las técnicas de ocultación, ya que, al mismo ritmo que crece la libertad de comunicarse, se multiplican los riesgos para la privacidad. La Agencia de Protección de Datos, máximo órgano español para velar por la intimidad personal frente al abuso de las nuevas tecnologías, ha advertido de que, a no ser que se utilice un mecanismo de

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

cifrado, debe asumirse que el correo electrónico no es seguro. Métodos como el asimétrico de clave pública defienden la confidencialidad del correo electrónico, fácilmente violable sin ellos, o la necesaria seguridad de las compras por Internet. Sin embargo, la duda persiste. ¿Son capaces las complejas claves actuales de garantizar el secreto? Muchas de las técnicas que se han considerado infalibles a lo largo de la Historia han mostrado sus puntos débiles ante la habilidad de los criptoanalistas, desde los misterios de Enigma, que cayeron en poder del enemigo, hasta el DES, desechado por el propio Gobierno estadounidense por poco fiable.

Pero a pesar de los muchos rumores que hablan de la poca seguridad que garantizan las transmisiones vía Internet, es muy improbable que un estafador pueda interceptar los datos reservados de una transacción, por ejemplo, el número de una tarjeta de crédito, porque los formularios que hay que rellenar han sido diseñados con programas que cifran los datos. Los hay tan simples como el Ro13, que sustituye cada letra por la situada 13 puestos más adelante, o extremadamente complicados.

En palabras de un apasionado de la criptografía, Edgar Allan Poe, "es dudoso que el género humano logre crear un enigma que el mismo ingenio humano no resuelva".

Conceptos básicos

En los procesos de almacenamiento y transmisión de la información normalmente aparece el problema de la seguridad. En el almacenamiento, el peligro lo representa el robo del soporte del mensaje o simplemente el acceso no

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

autorizado a esa información, mientras que en las transmisiones lo es la intervención del canal.

La protección de la información se lleva a cabo variando su forma. Se llama cifrado (o transformación criptográfica) a una transformación del texto original (llamado también texto inicial o texto claro) que lo convierte en el llamado texto cifrado o criptograma. Análogamente, se llama descifrado a la transformación que permite recuperar el texto original a partir del texto cifrado.

La teoría de la información estudia el proceso de la transmisión ruidosa de un mensaje:

canal ruidoso

mensaje transmitido ---> mensaje recibido

Por otro lado, la criptografía estudia el proceso de cifrado de un texto.

transformación criptográfica

texto original ---> texto cifrado

En el primer caso, la distorsión del mensaje transmitido no es intencionada, sino que se debe al canal. Además, el receptor trata de recuperar el mensaje transmitido, usando para ello el mensaje recibido y una descripción probabilística del canal ruidoso. En el otro caso, la distorsión si es intencionada y el interceptor intenta obtener el texto original, usando para ello el criptograma y una descripción parcial del proceso de cifrado.

En cuanto a objetivos, en la teoría de la información se intenta transmitir el mensaje lo más claro posible, mientras que en criptografía se trata de lo contrario; es decir, hacer el mensaje incompresible para el enemigo. Sin embargo, aunque opuestos, estos propósitos se pueden combinar para proteger los mensajes contra

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

el enemigo y el ruido a la vez. Para ello, primero hay que cifrar el mensaje y luego hay que aplicar al criptograma resultante un código corrector de errores.

Shannon, padre de la teoría de la información, ya mencionó en 1949 la relación existente entre ésta y la criptografía. Desde entonces se usan su teoría y nomenclatura para el estudio teórico de la criptografía.

Como ya se ha dicho, la criptología representa una lucha entre el criptógrafo, que trata de mantener en secreto un mensaje usando para ello una familia de transformaciones, y el enemigo, que intenta recuperar el texto inicial. Como en toda lucha, es necesario establecer unas reglas.

<u>Reglas de Kerckhoffs</u>

Kerckhoffs (s. XIX), en su trabajo titulado "La criptografía militar", recomendó que los sistemas criptográficos cumpliesen las siguientes reglas, que efectivamente han sido adoptadas por gran parte de la comunidad criptográfica:

No debe existir ninguna forma de recuperar mediante el criptograma el texto inicial o la clave. Esta regla se considera cumplida siempre que la complejidad del proceso de recuperación del texto original sea suficiente para mantener la seguridad del sistema.

- *Todo sistema criptográfico debe estar compuesto por dos tipos distintos de información.*
    - *Pública, como es la familia de algoritmos que lo definen.*
    - *Privada, como es la clave que se usa en cada cifrado particular*. En los sistemas de clave pública, parte de la clave es también información pública.
- La forma de escoger la clave debe ser fácil de recordar y modificar.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

- Debe ser factible la comunicación del criptograma por los medios de transmisión habituales.
- La complejidad del proceso de recuperación del texto original debe corresponderse con el beneficio obtenido.

Tipos de ataque

Para el estudio de los sistemas criptográficos es conveniente conocer la la situación del enemigo. Se tienen los siguientes ataques posibles:

- *Ataque sólo con texto cifrado*. Esta es la peor situación posible para el criptoanalista, ya que se presenta cuando sólo conoce el criptograma.
- *Ataque con texto original conocido*. Consiste en que el criptoanalista tiene acceso a una correspondencia del texto inicial y cifrado. Se de este caso, por ejemplo, cuando conoce el tema del que trata el mensaje, pues eso proporciona una correspondencia entre las palabras más probables y las palabras cifradas mas repetidas.
- *Ataque con texto original escogido*. Este caso se da cuando el enemigo puede obtener, además del criptograma que trata de descifrar, el cifrado de cualquier texto que él elija, entendiéndose que no es que él sepa cifrarlo, sino que lo obtiene ya cifrado.
- *Ataque con texto cifrado escogido*. Se presenta cuando el enemigo puede obtener el texto original correspondiente a determinados textos cifrados de su elección.

Retomando la primera regla de Kerckhoffs, se pueden distinguir dos tipos de secreto: el secreto teórico o incondicional y el secreto práctico o computacional.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

El primero se basa en que la información disponible para el enemigo no es suficiente para romper el sistema. Por ejemplo, se da este caso cuando el enemigo sólo conoce una cantidad de criptograma insuficiente para el criptoanálisis. Por el contrario, el secreto práctico se mide de acuerdo con la complejidad computacional del criptoanálisis. Según las necesidades actuales, y debido principalmente a la gran cantidad de información que se transmite habitualmente, los diseñadores de sistemas criptográficos deben suponer que el enemigo puede hacer al menos un ataque del segundo tipo, luego deben intentar conseguir al menos secreto práctico.

Como ya se ha mencionado, existen muchos puntos en común entre la teoría de la información y la criptografía. En concreto, entre codificación y criptografía se tiene que la codificación representa una forma alternativa de esconder un mensaje. La diferencia esencial entre un cifrado y un código estriba en que este último es un diccionario fijo; es decir, a cada palabra le corresponde siempre la misma palabra código.

Las principales desventajas del código cuando se utiliza como cifrado son:

- Sólo se pueden transmitir aquellas palabras que tengan traducción asignada en el diccionario del código.
- El receptor debe tener el diccionario para poder decodificar; es decir, el código completo constituye la clave.
- Su implementación, sobre todo a la hora de cambiar el código, es muy costosa.
- El criptoanálisis se puede basar en un análisis de frecuencias.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

La ventaja de este sistema es la compresión de la información siempre que las palabras de códigos usadas sean más cortas que las palabras originales.

Fuente del texto

En primer lugar hay que distinguir, según la fuente que proporciona el texto, el cifrado digital del analógico. Si dicha fuente es digital, los mensajes están constituidos por grupos de elementos pertenecientes a una colección finita denominada alfabeto. Por otro lado, se dice que la fuente es analógica si genera un continuo de valores. El cifrado de fuentes analógicas no resulta en absoluto seguro, por lo que es necesario previamente digitalizar los mensajes analógicos.

En los cifrados digitales, tanto el texto original como el texto cifrado están formados por letras de un alfabeto. Se considera, además del alfabeto A={a1,a2,...,am}, los alfabetos formados a partir de la concatenación de sus letras A^*n*={(a1,...(*n*,a1),...,(am,...,(*n*,am)}. Sus m^*n* elementos son las llamadas *n*-palabras. Por motivos prácticos se sustituyen las letras por números, usándose entonces como alfabeto el conjunto de enteros Zm={0,1,...,m-1}. Al conjunto de *n*-palabras correspondiente a este alfabeto se le denota Zm,*n*. Principalmente se usan dos alfabetos numéricos, el alfabeto decima Z10={0,1,...9,} y el alfabeto binario {0,1}.

El diseñador de un criptosistema, además de escoger un modelo para la fuente del texto, tiene que elegir los símbolos matemáticos necesarios para representar las unidades del texto.

Secreto perfecto

Un criptosistema tiene secreto perfecto cuando es incondicionalmente seguro frente a un ataque sólo con texto cifrado si la clave se usa sólo una vez.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Equivocación

Es más fácil deducir acertadamente el texto original correspondiente a un criptograma que acertar con la clave utilizada.

Redundancia y distancia unicidad

Se supone equiprobabilidad para las claves y para las palabras con significado, y probabilidad nula para las palabras sin significado. Si un enemigo hace un ataque sólo con texto cifrado, una condición necesaria y generalmente suficiente para romper el cifrado con un análisis de frecuencias es que la fuente del texto sea redundante.

La distancia de unicidad representa en muchos casos la cantidad de texto que hay que interceptar para que no haya incertidumbre sobre la clave usada. Si la fuente es redundante, entonces la distancia de unicidad es finita. La distancia de unicidad no es la cantidad mínima de criptograma necesaria para asegurar un criptoanálisis fácil, sino la que proporciona suficiente confianza en una solución del criptoanálisis.

Criptografía de Clave Pública

En los cifrados asimétricos o de clave pública la clave de descifrado no se puede calcular a partir de la de cifrado.

En 1975, dos ingenieros electrónicos de la Universidad de Stanford, Whitfield Diffie y Martin Hellman, publican un artículo llamado “New Directions in Cryptography” (Nuevas direcciones en Criptografía) que tuvo el efecto de una bomba en la comunidad de criptógrafos, introduciendo el concepto de criptografía

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

de clave pública. Los algoritmos de cifrado con clave secreta (o clave privada), los únicos conocidos hasta entonces, ya no satisfacían las nuevas necesidades que aparecían paralelamente a la explosión de nuevos medios de comunicación muy impersonales – como por ejemplo, el desarrollo de las redes de comunicaciones.

La solución completamente revolucionaria que propusieron fue introducir la noción de función de una sola vía con trampa, o trapdoor one-way function (también conocida como función en un sentido irreversible). Es una función que se calcula fácilmente en un sentido, pero que es computacionalmente imposible de invertir si no se conoce un secreto llamado trampa, aunque la función sea conocida por todos. Entonces la clave pública es la función, mientras que la trampa, conocida por un número restringido de usuarios se denomina clave privada. Asó nació el mundo de Alice, Bob y compañía. Alice y Bob son dos personas que buscan comunicarse de forma íntegra mientras algunas personas pueden interponerse, escuchando o incluso alterando el canal de comunicación.

Cualquier usuario puede cifrar usando la clave pública, pero sólo aquellos que conozcan la clave secreta pueden descifrar correctamente.

En consonancia con el espíritu de la criptografía moderna, y tal cómo sucedía en los sistemas simétricos, los algoritmos de cifrado y de descifrado son públicos, por lo que la seguridad del sistema se basa únicamente en la clave de descifrado.

Según Diffie y Hellman, todo algoritmo de clave pública debe cumplir las siguientes propiedades de complejidad computacional:

1. Cualquier usuario puede calcular sus propias claves pública y privada en tiempo polinomial.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

2. El emisor puede cifrar su mensaje con la clave pública del receptor en tiempo polinomial.
3. El receptor puede descifrar el criptograma con la clave privada en en tiempo polinomial.
4. El criptoanalista que intente averiguar la clave privada mediante la pública se encontrará con un problema intratable.
5. El criptoanalista que intente descifrar un criptograma teniendo la clave pública se encontrará con un problema intratable.

En la práctica, el diseñador de algoritmos asimétricos se encuentra con cinco problemas numéricos distintos. Los tres primeros, correspondientes a las condiciones 1,2 y 3, deben pertenecer a la clase polinomial. Los otros dos, correspondientes a las condiciones 4 y 5, son problemas complejos, preferiblemente NP-completos. Hay que señalar que las condiciones 4 y 5 no exigen sólo la simple pertenencia a la clase de los problemas NP-completos, ya que aunque un problema pertenezca a esta clase siempre puede darse algún ejemplo concreto que se resuelva en tiempo polinomial.

En líneas generales, un esquema a seguir para la construcción de un criptosistema de clave pública es el siguiente:

- Escoger un problema difícil P, a ser posible intratable.
- Escoger un subproblema de P fácil, Pfacil, que se resuelva en tiempo polinomial preferiblemente en tiempo lineal.
- Transformar el problema Pfacil de tal manera que el problema resultante Pdificil, no se parezca al inicial, pero sí al problema original P.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

- Publicar el problema Pdificil y la forma en que debe ser usado, constituyendo este proceso la clave (pública) de cifrado. La información sobre cómo se puede recuperar el problema Pfacil a partir del problema Pdificil se mantiene en secreto y constituye la clave (secreta) de descifrado.

Los usuarios legítimos utilizan la clave secreta para llevar a cabo el descifrado convirtiendo el problema Pdificil en el problema Pfacil, mientras que, por el contrario, el criptoanalista debe enfrentarse forzosamente a la resolución del problema Pdificil.

Es más difícil diseñar un sistema de clave pública seguro contra un ataque con texto original escogido que un sistema de clave secreta seguro frente al mismo tipo de ataque.

En la construcción de criptosistemas se pueden observar diferencias entre los algoritmos para sistemas simétricos y los usados en clave pública. En primer lugar, existen mayores restricciones de diseño para un algoritmo asimétrico que para uno simétrico, debido a que la clave pública representa información adicional que potencialmente un enemigo puede usar para llevar a cabo el criptoanálisis. Normalmente, el algoritmo de clave pública basa su seguridad en la dificultad de resolver algún problema matemático conocido, mientras que algunos algoritmos simétricos, cómo el DES, se diseñan de tal manera que las ecuaciónes matemáticas que los describen son tan complejas que no son resolubles analíticamente.

En segundo lugar, existen grandes diferencias en la generación de claves. En los algoritmos simétricos, en los que el conocimiento de la clave de cifrado es

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

equivalente al de la de descifrado, y viceversa, la clave se puede seleccionar de forma aleatoria. Sin embargo, en los algoritmos asimétricos, cómo la relación entre clave de cifrado y de descifrado no es pública, se necesita un procedimiento para calcular la pública a partir de la clave privada que sea computacionalmente eficiente y tal que el cálculo inverso sea imposible de realizar.

Hace algunos años, este tipo de sistemas no parecía tener ninguna ventaja en el mundo criptográfico, porque tradicionalmente la criptografía se usaba sólo con propósitos militares y diplomáticos, y en estos casos el grupo de usuarios es lo suficientemente pequeño cómo para compartir un sistema de claves. Sin embargo, en la actualidad, las aplicaciones de la criptografía han aumentado progresivamente, hasta alcanzar muchas otras áreas donde los sistemas de comunicación tienen un papel vital. Cada vez con mayor frecuencia se pueden encontrar grandes redes de usuarios en las que es necesario que dos cualesquiera sean capaces de mantener secretas sus comunicaciones entre sí. En estos casos, el intercambio continuo de claves no es una solución muy eficiente.

Por otro lado, hay que resaltar la ventaja que representa en los sistemas asimétricos la posibilidad de iniciar comunicaciones secretas sin haber tenido ningún contacto previo.

A continuación, nombramos algunos de los sistemas de clave pública que han tenido más trascendencia.

- **Sistema RSA.**

  A principios del los ’90, los investigadores Rivest, Shamir y Adleman propusieron un estándar de codificación asimétrica, denominado RSA. Los algoritmos de clave pública como RSA, se sustentan en una base

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

matemática donde cada una de las partes intervinientes dispone de un par de claves: una se denomina clave pública, y está destinada a ser distribuida libremente (de hecho, cuanto más ampliamente se haya distribuido esta clave, menos posibilidades habrá de “usurpación de identidad electrónica”

La segunda clave es privada y será conocida solamente por su legítimo propietario. Esta debe ser custodiada con el mismo celo con que se haría para una clave DES. La base matemática se basa en el hecho de que no existe una forma eficiente de factorizar números que sean productos de dos grandes primos.

- **Sistema de Rabin.**

  Se basa también en la factorizacion.

- **Sistema de ElGamal.**

  Se basa en el problema del logaritmo discreto.

- **Sistema de Merkle-Hellman.**

  Esta basado en el problema de la mochila.

- **Sistema de McEliece.**

  Se basa en la teoría de la codificación algebraica, utilizando el hecho de que la decodificación de un código lineal general es un problema NP-completo.

- **Sistemas basados en curvas elípticas.**

  En 1985, la teoría de las curvas elípticas encontró de la mano de Miller aplicación en la criptografía. La razón fundamental que lo motivó fue que las curvas elípticas definidas sobre cuerpos finitos proporcionan grupos finitos abelianos, donde los cálculos se efectúan con la eficiencia que requiere un

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

criptosistema, y donde el cálculo de logaritmos es aún más difícil que en los cuerpos finitos. Además, existe mayor facilidad para escoger una curva elíptica que para encontrar un cuerpo finito, lo que da una ventaja más frente a su predecesor, el sistema de ElGamal.

- **Sistema probabilístico.**

Aunque la criptografía de clave pública resuelve el importante problema de la distribución de claves que se presenta en la criptografía de clave secreta; en clave pública se presenta otro problema, el texto cifrado C=Ek (M) siempre deja escapar alguna información sobre el texto original porque el criptoanalista puede calcular por sí mismo la función de cifrado con la clave pública sobre cualquier texto que quiera. Dado cualquier M' de su elección, puede fácilmente descubrir si el mensaje original M=M', pues esto se cumple si, y sólo si Ek(M')=C. Incluso aunque recuperar M a partir de C fuera efectivamente infactible, no sabemos cómo medir la información que deja escapar sobre M.

El propósito de la criptografía probabilística (noción ideada por Golwaser y Micali) es cifrar mensajes de manera que no exista cálculo factible que pueda producir información en lo que respecta al texto original correspondiente (salvo con una probabilidad ínfima). Hay que decir que estos sistemas no ofrecen verdadero secreto perfecto, son totalmente inseguros contra criptoanalistas con poder de cálculo ilimitado.

La principal diferencia técnica entre el cifrado probabilístico y los criptosistemas de clave pública es que los algoritmos de cifrado son probabilísticos en lugar de determinísticos: el mismo mensaje original puede

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

dar lugar a un gran número de criptogramas distintos. En consecuencia, un criptoanalista que tenga un candidato para el texto original no podría verificar su suposición cifrándolo y comparando el resultado con el criptograma interceptado.

- **Sistema SET**

Recientemente, la asociación entre Visa y Mastercard, con el apoyo de empresas de la industria de la computación como GTE, IBM, Microsoft, Netscape, SAIC, Terisa y Verisign, definió el estándar denominado SET (Secure Electronic Transactions), un conjunto de especificaciones que permitirá el desarrollo del comercio electrónico en Internet y otras redes públicas, de forma segura para todos los participantes: usuario final, comerciante, entidades financieras, administradoras de tarjetas y propietarios de marcas de tarjetas.

Estas especificaciones permiten

- Proporcionar la autentificación necesaria entre compradores, comerciantes e instituciones financieras.
- Garantizar la confidencialidad de la información sensible (número de tarjeta o cuenta, fecha de caducidad, monto de la transacción, etc).
- Preservar la integridad de la información que contiene tanto la orden de pedido como las instrucciones de pago.
- Definir los algoritmos criptográficos y protocolos necesarios para los servicios anteriores.

La encriptación de mensajes en bloques de 1024 bits, como permiten RSA

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

y SET, asegura su inviolabilidad, al menos para las capacidades de procesamiento de la computación de hoy. Sin embargo, y nuevamente, el problema no es técnico sino legal. En Estados Unidos, por ejemplo, está prohibida la encriptación de mensajes con algoritmos superiores a los 40 bits.

· **Sistema PGP**

PGP (Pretty Good Privacy) es un protocolo de dominio público para encriptar mensajes de correo electrónico, creado por Philip Zimmerman y que ha alcanzado una gran popularidad en la red.

Básicamente utiliza el sistema RSA de llave pública y privada. Solo podrá ser descifrado por quien posea ambas claves, la pública y la privada. Este programa está a libre disposición en varios servidores de la red, aunque su uso es ilegal en países como Estados Unidos, Francia, China, Vietnam e Indonesia, entre otros.

Criptografía de Clave Secreta

En los cifrados de clave secreta, la seguridad depende de un secreto compartido exclusivamente por emisor y receptor.

La principal amenaza criptoanalítica proviene de la alta redundancia de la fuente. Shannon sugirió por ello dos métodos básicos para frustrar un criptoanálisis estadístico: la difusión y la confusión.

El propósito de la difusión consiste en anular la influencia de la redundancia de la fuente sobre el texto cifrado. Hay dos formas de conseguirlo. La primera,

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

conocida como transposición, evita los criptoanálisis basados en las frecuencias de las *n*-palabras. La otra manera consiste en hacer que cada letra del texto cifrado dependa de un gran número de letras del texto original.

El objetivo de la confusión consiste en hacer que la relación entre la clave y el texto cifrado sea lo más compleja posible, haciendo así que las estadísticas del texto cifrado no estén muy influidas por las del texto original. Eso se consigue normalmente con la técnica de la sustitución. En solitario, ni confusión ni difusión constituyen buenas técnicas de cifrado.

Transposición, sustitución y producto

Se puede hacer otra gran división de los cifrados según el tipo de operación que se realiza en el cifrado. Dadas la característica finita del alfabeto y la hipótesis de no variación de la longitud del texto, existen dos opciones para el cifrado. La primera, llamada transposición, consiste en crear el texto cifrado simplemente desordenando las unidades que forman el texto original. La segunda, llamada sustitución, consiste en sustituir las unidades del texto original por otras.

El cifrado por transposición consiste en la alteración del orden de las unidades del texto original según una clave. El cifrado por sustitución consiste en el reemplazamiento de las unidades del texto original según una clave.

Se llama cifrado producto a la aplicación iterativa de cifrados sobre textos ya cifrados, es decir, a la composición de varios cifrados. En general, los cifrados simétricos son cifrados producto de las dos operaciones mencionadas, sustitución y transposición.

Sustitución y transposición no resultan muy efectivos usados individualmente, sin embargo constituyen la base de sistemas mucho más difíciles

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

de criptoanalizar. Algunos de estos esquemas fueron usados en los años veinte para el diseño de las máquinas de rotor.

Cifrado en bloque, DES

Independientemente de la clasificación realizada en el apartado anterior según la fuente que genera el texto, los cifrados simétricos se pueden clasificar en dos grandes grupos: los correspondientes a fuentes que generan *n*-palabras y los correspondientes a fuentes que generan letras. En el primer caso se habla de cifrados en bloque y en el segundo de cifrados en flujo.

El cifrado en bloque opera sobre textos formados por *n*-palabras, convirtiendo cada una de ellas en una nueva *n*-palabra.

Sin duda el cifrado en bloque más conocido es el llamado DES. Este sistema se puede catalogar como un cifrado en bloque que es a la vez un cifrado producto de transposiciones y sustituciones.

A finales de los años cuarenta, Shannnon sugirió nuevas ideas para futuros sistemas de cifrado. Sus sugerencias se referían al uso de operaciones múltiples que mezclaran transposiciones y sustituciones. Estas ideas fueron aprovechadas por IBM en los años setenta, cuando desarrolló un nuevo sistema llamado LUCIFER. Poco después en 1976, el gobierno de EEUU adoptó como estándar un sistema de cifrado basado en el LUCIFER y denominado DES (Data Encryption Standard). En consecuencia casi todos los gobiernos del mundo aceptaron el mismo cifrado o parte de él como estándar en las comunicaciones de las redes bancarias y comerciales.

En el DES, el bloque de entrada M en primer lugar sufre una transposición bajo una permutación denominado IP, originando To=IP(M). Después de pasar To

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

dieciséis veces por una función f, se transpone bajo la permutación inversa IP', obteniéndose así el resultado final.

Cifrado en Flujo

El cifrado en flujo se aplica sobre textos formados por letras combinándolas con un flujo de bits secretos (secuencia cifrante) mediante un operador *. El descifrado se realiza de una forma análoga, combinando mediante el operador *^-1 las letras del texto cifrado con la secuencia cifrante, produciendo así las letras del texto original. La secuencia cifrante se produce mediante un generador de bits.

El primero en proponer un cifrado en el que el texto cifrado resulta de combinar mediante una operación eficiente el texto original con el flujo de bits secretos fue Vernam (1917). El cifrado de Vernam se basa en el operador o-exclusivo (suma módulo 2). Esta operación resulta especialmente indicada para este tipo de cifrado, ya que *^-1=*.

En el cifrado de Vernam, primero se codifica cada letra del alfabeto según el código de Baudot, es decir A=11000,...,Z=10001, y luego se le suma en módulo 2 la clave expresada en binario.

Por otra parte, se puede utilizar como representación del cifrado en flujo un autómata finito determinista en el que los elementos de entrada corresponden a los mensajes M, los elementos de salida a los mensajes cifrados C y la clave determina las funciones de transición de estados y salida.

Los diferentes tipos de criptosistemas quedan reflejados en las restricciones que se imponen a estas funciones. El descifrado puede verse como un autómata análogo en el que se intercambian M y C.

Los cifrados en flujo pueden dividirse en cifrados síncronos y cifrados

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

asíncronos, según la dependencia entre la función de transición de estados y el espacio de los elementos de entrada.

Los cifrados síncronos son aquellos en los que cada estado sólo depende anterior y no de los caracteres de la secuencia del mensaje original. En otras palabras, la función de transición de estados es independiente del mensaje. Esto implica que si durante la transmisión se pierde un símbolo del criptograma, entonces se pierde la sincronización entre emisor y receptor, por lo que ambos tienen que sincronizar sus generadores de clave antes de continuar. Sin embargo, como ventaja se tiene que una distorsión en la transmisión de un símbolo se traduce sólo en un símbolo mal descifrado.

Los cifrados autosíncronos, por el contrario, se caracterizan porque cada símbolo del texto cifrado depende de un número fijo de símbolos del texto cifrado anterior. Eso elimina la necesidad de sincronización, pero a cambio un error de transmisión o una pérdida de un carácter origina la pérdida o mala obtención de un número de símbolos del mensaje original; es decir propaganda de errores.

La secuencia cifrante ideal es una secuencia infinita, determinada por una clave de manera que parezca aleatoria y que ningún enemigo pueda generarla.

Clave Pública o Clave Secreta

En realidad, el interés de la criptografía de clave pública es el de ocuparse de muchos problemas de seguridad, y ofrecer una gran flexibilidad. Lo que permite entre otras cosas encontrar soluciones a los problemas de autentificación tales como la identificación de personas y la autentificación de documentos.

Por otro lado, como la criptografía de clave secreta, la criptografía de clave

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

pública permite elaborar sistemas de cifrado, asegurando la confidencialidad de las comunicaciones.

Supongamos que Alice desea comunicarse con Bob de forma privada. Alice encuentra en un directorio la clave pública de Bob, y cifra el mensaje con esta clave. Cuando Bob recibe el mensaje cifrado, utiliza su clave privada para descifrar el mensaje y recuperar el texto llano. Las dos claves juegan papeles completamente distintos, ésta es la razón por la que se habla de criptosistemas asimétricos, en oposición a los criptosistemas de clave secreta que utilizan la misma clave para cifrar que para descifrar y se llaman criptosistemas simétricos.

La criptografía de clave pública presenta otra ventaja sobre la criptografía de clave secreta. Si n personas desean comunicarse mediante un criptosistema de clave secreta, cada una de ellas debe disponer de una clave diferente para cada persona del grupo. Por tanto hace falta poder generar en total n(n-1) claves. Teniendo en cuenta que n puede ser del orden de varios millares, será necesario generar ficheros de varios millos de claves. Además, añadir un miembro al grupo no es sencillo, ya que habrá que generar otras n claves para que el nuevo miembro pueda comunicarse con los demás integrantes del grupo, y después distribuir las nuevas claves a todo el grupo. Por el contrario, en el caso de un criptosistema asimétrico, se guardan las n claves públicas de los miembros del grupo en un directorio. Para añadir un miembro, basta con que ponga su clave pública en el directorio.

Aplicaciones Criptográficas

Entre las muchas aplicaciones de la criptografía, se encuentran la

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

autenticación, la firma digital, la identificación de usuario, seguridad en redes y protocolos criptográficos. Hablaremos solamente de los dos primeros.

Autenticación

- *Criptoanalistas pasivos*: Su único propósito es escuchar mensajes transmitidos por el canal.
- *Criptoanalista activo:* No sólo escucha, si no que intenta inyectar mensajes propios para que el receptor crea que fueron enviados por un emisor legítimo.

El propósito de un esquema de autenticación es la detección de la presencia de criptoanalistas activos. Siempre que un receptor B reciba un mensaje que parezca provenir del emisor A, el esquema debe permitirle averiguar no sólo si el mensaje viene de A, si no si fue modificado por el camino. Se supone que este tipo de intrusos tiene acceso a la escucha de cuantos mensajes quiera y que su meta es conseguir que un mensaje falsificado no sea detectado por el receptor.

Un sistema proporciona:

- *secreto*, si permite determinar quién puede recibir un mensaje.
- *autenticidad*, si permite determinar quién puede enviar un mensaje.

Hay dos posibles situaciones en las que se pueden encontrar los interlocutores. Puede que sea suficiente la comprobación de que un mensaje no ha sido modificado por un tercero, o bien puede ser necesario que el receptor sea capaz de demostrar que realmente recibió el mensaje del emisor legítimo tal y como éste lo envió. La solución a la primera situación protege a ambos comunicantes de la acción de posibles enemigos, pero no protege contra los fraudes cometidos por uno de ellos. En la segunda situación, si se produce un

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

fraude cometido por uno de los dos comunicantes, se presenta el llamado problema de la disputa, que se resuelve proporcionando una demostración de la identidad del emisor al receptor. El primer problema se afronta con la llamada autenticación, mientras que el segundo problema se resuelve mediante las llamadas firmas digitales.

Se distinguen aquellos esquemas de autenticación en los que el enemigo conoce la información que se transmite (llamados con secreto) de los esquemas en los que el enemigo ignora la información que se transmite (llamados sin secreto). En el primero el mensaje se codifica según una regla de codificación acordada por ambos comunicantes, mientras que en el segundo el mensaje se cifra utilizando una clave también acordada por ambos usuarios.

Métodos.

Las actividades del adversario pueden consistir en:

- Bloquear el flujo de información.
- Grabar información y repetirla luego en una transmisión falsa.
- Cambiar la información borrando, insertando y/o reordenándola.

Tal y como se definieron los esquemas de autenticación con y sin secreto se tiene que en el primero, un autenticador conocido por emisor y receptor se añade al texto que se desea autenticar y luego se cifra. De esta manera al texto resultante se le proporciona al mismo tiempo secreto y autenticación. En el segundo caso, al texto simplemente se le añade un código de autenticación. Si en alguno de estos dos casos se usa un sistema simétrico, entonces emisor y receptor han de confiar plenamente el uno en el otro, porque ambos comparten la

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

misma clave.

Por tanto se pueden clasificar claramente los métodos de autenticación, según si se utiliza criptografía simétrica o asimétrica, en:

- métodos basados en criptosistemas simétricos y
- métodos basados en criptosistemas asimétricos.

Veremos solamente el caso de los esquemas de autenticación mediante criptosistemas asimétricos. Consideremos en particular el algoritmo RSA. Cada usuario i genera una clave secreta di a partir de dos números primos (pi,qi) y pública ni=pi*qi y ei-di-1(mod (ni)). Para usar este sistema como esquema de autenticación, el usuario i obtiene Di(m) y lo envía al receptor. Este para verificar su autenticidad, le aplica Ei con la clave pública de i, obteniendo obteniendo Ei(Di(m)) - m (mod ni), demostrando así que es autentico.

Uno de los principales problemas que se presentan en la autenticación es la posibilidad de que el enemigo utilice mensajes anteriores. Para intentar resolverlo, es conveniente añadir al texto alguna señal que impida el engaño, como, por ejemplo, la fecha.

Aunque un esquema de autenticación permite al usuario A confiar en que el mensaje que ha recibido viene de quién dice venir, eso no le permite convencer a un tercero de ello. Por eso, los esquemas de autenticación resultan débiles ante el engaño de uno de los dos interlocutores legítimos. El uso de las funciones trampas hace posible la solución a este problema con la noción de firma digital.

Firma digital

El desarrollo de las telecomunicaciones en estos últimos años ha creado toda una variedad de nuevas necesidades. Por ejemplo, dado que en la mayoría

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

de las operaciones bancarias es necesario firmar los documentos, con el uso de los ordenadores se requiere un nuevo planteamiento, donde una firma digital sustituye a la firma manual y cumple las mismas propiedades que ésta. Se puede distinguir la firma:

- implícita (contenida dentro del texto) de la explícita (añadida al texto como una marca inseparable);
- privada (legible sólo para quien comparte cierto secreto con el emisor) de la pública (legible para todo el mundo).

La firma digital debe ser:

- única, pudiéndola generar solamente el usuario legítimo;
- no falsificable, el intento de falsificación debe llevar asociada la resolución de un problema numérico intratable;
- fácil de autenticar, pudiendo cualquier receptor establecer su autentidad aún después de mucho tiempo;
- irrevocable, el autor de una firma no puede negar su autoría;
- barata y fácil de generar.

Otra característica que han de tener las firmas digitales es que deben depender tanto del mensaje como del autor. Esto debe ser así porque en otro caso el receptor podría modificar el mensaje y mantener la firma, produciéndose así un fraude.

Si el emisor A envía un mensaje firmado digitalmente al receptor B, este último no sólo debe convencerse de que el mensaje fue firmado por el primero, sino que, además, debe ser capaz de demostrar a un juez que A realmente firmó

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

ese mensaje. Esta noción fue ideada por Diffie y Hellman en 1976. La firma digital y el correo electrónico ofrecen conjuntamente sustanciosas ventajas, una de ellas es hacer posible el correo certificado y la firma electrónica de contratos.

La idea principal de la firma digital es que solamente el emisor la pueda producir y además se pueda demostrar que, efectivamente, es él quien la produce. Representa por tanto, un control más fuerte que la autenticación.

Considérese un sistema de clave pública donde Mk y Ck denotan respectivamente, a los espacios de mensajes originales y cifrados asociados a una clave k.

Un sistema de clave pública ofrece la posibilidad de ser usado para firmas digitales siempre que para k perteneciente a K; Mk=Ck y para m perteneciente a M; Ek(Dk(m))=m.

Denotando la clave escogida por el usuario A como a, se tiene que si el criptosistema es seguro, entonces sólo A puede calcular Da, pero cualquiera puede calcular Ea de forma eficiente.

Considérese un mensaje m perteneciente a Ma y s=Da(m). Cualquier usuario puede calcular Ea(s) y comprobar que coincide con m, pero, sin embargo, sólo A puede deducir el valor de s para el que Ea(s)=m. En este sentido, s puede ser considerado como la firma de A para el mensaje m. Si B muestra el mensaje s y su cifrado Ea(s) a un juez, éste debe de convencerse de que ningún otro m s que A pudo haber firmado ese documento con Ea(s). En otras palabras, los algoritmos de descifrado y de cifrado pueden verse, respectivamente, como un algoritmo de firma digital y su correspondiente algoritmo de verificación.

La actual velocidad de los sistemas de clave pública recomienda su

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

utilización para generar firmas digitales cortas y separadas del texto (firmas explícitas).

Si además de capacidad de firma digital se desea secreto, entonces la firma digital puede ser utilizada conjuntamente con un cifrado de clave pública. Si el usuario A quiere enviar un mensaje secreto firmado a un usuario B, puede usar el algoritmo secreto de firma digital Da y el algoritmo público de verificación Eb, produciendo c=Eb(Da(m)). Si envía este mensaje c al usuario B a través de un canal inseguro, entonces B puede calcular la firma de A mediante s=Db(c) y de ahí recuperar el mensaje claro m=Ea(s). Esto supone que en el caso de que hubiera más de un posible emisor en el encabezado del mensaje debe decir claramente que el mensaje viene de A para que B pueda saber qué algoritmo de verificación debe aplicar. Es más, si B guarda el mensaje s, llegado el momento podrá probar ante un juez que A le envió el mensaje m, tal y como se explicó previamente.

Hay que tener cuidado cuando se afirma que el conocimiento por parte de B del valor de una firma s tal que Ea(s)=m tiene que ser considerada como una demostración de que A firmó m, ya que B podría elegir un valor s aleatorio, calcular m=Ea(s) y luego argumentar que A envió m, aunque no tenga sentido. Las cosas empeoran si la proporción de mensajes con significado es alta, porque en este caso B podría escoger varios valores de s aleatoriamente hasta toparse con uno para el que Ea(s) tenga significado. Por esta razón, es recomendable alguna forma normalizada para las firmas digitales, y que se acepte s como firma del mensaje m sólo si se tiene que m=Ea(s) y que m está en forma normalizada. Desafortunadamente, todavía no está claro cual sería la forma normalizada recomendable para utilizar con el sistema RSA.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Si se utiliza el RSA para conseguir secreto y como firma digital, entonces es preferible que cada ususario use claves distintas para cada uno de los dos propósitos. De esta forma, cada usuario tendría asignada una clave en el directorio público de claves de cifrado y otra distinta en el directorio público de firma digitales. Esta separación es útil para dos propósitos. En primer lugar, ayuda a evitar el problema que surge cuando el módulo del emisor es mayor que el del receptor. En segundo lugar dado que el RSA es débil frente a algunos ataques con texto escogido, tales ataques pueden verse facilitados si se utiliza la misma clave para ambos fines, y en consecuencia es preferible evitarlo.

El criptosistema RSA presenta algunos inconvenientes para las firmas digitales parecidos a los que presentaba como sistema de cifrado. En particular, no se sabe a ciencia cierta si es tan difícil de romper como la factorización de grandes enteros. Incluso aunque así fuera, dados un mensaje original escogido m y la clave de cifrado de otro usuario (*e*,*n*), calcular la firma digital s tal que m - s^*e*(mod *n*) puede ser mucho más fácil si se tiene, además, (s',m'), donde s' es la firma digital del usuario legítimo para un mensaje m' muy parecido al mensaje m. En otras palabras, podría resultar fácil falsificar firmas digitales para algún mensaje dado después de haber visto las firmas digitales auténticas de varios mensaje parecidos.

Lo visto anteriormente sugiere que podría resultar más favorable para diseñar esquemas de firmas digitales el empleo de sistemas probabilísticos, en vez de los sistemas de clave pública. Sin embargo, ésta es una tarea difícil, ya que, por ejemplo, se ha demostrado que el sistema probabilístico de Blum-Goldwaser es completamente inútil par firmas digitales. Afortunadamente, ya se

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

han desarrollado algunos esquemas de firmas digitales suficientemente fuertes y seguros basados en el problema de la factorización. Estos esquemas resisten incluso ataques con texto escogido.

Seguridad y Privacidad de la Información

El comercio electrónico en Internet llegará a convertirse, si es que no lo es ya, en un hecho tan habitual como el comercio convencional. Los medios de pago utilizados serán las tarjetas de débito y crédito, probablemente en una modalidad "inteligente"[2].

Pero para que ello ocurra, es necesario que todos los actores que intervienen en el proceso adquieran la convicción de la seguridad que los diversos intercambios de información que tienen lugar; más aún si dicha información asume la representación de valores en la esfera de la circulación de las mercancías y el dinero.

Sin embargo, existe una segunda perspectiva para aquilatar debidamente el tema de la seguridad de los datos: la privacidad de las comunicaciones electrónicas es una extensión natural de los derechos individuales de la libertad de expresión, intimidad, integridad, seguridad, propiedad intelectual y protección de la hora de las personas, reconocidos tanto en la Declaración Universal de los Derechos del Hombre como en las constituciones dela inmensa mayoría de los países del orbe.

Desde el punto vista técnico, la investigación y desarrollo, en especial en las áreas de encriptación y cifrado de mensajes, ha proveído soluciones

---

[2] http://www.reuna.cl/central_apunte/apuntes/soc_info5.html, tomado el 11 de abril del 2005

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

consistentes, sin perjuicio de que en materia de seguridad la prudencia aconseja tomar todo avance como un paso esencialmente transitorio.

En términos genéricos, el objetivo de proteger la información apunta a obtener:

- Confidencialidad, que los mensajes transferidos o recibidos sean secretos y nadie más que su legítimo destinatario tenga acceso a ellos.
- Disponibilidad, que los recursos estén disponibles cuando se necesiten, y que no se usen indebidamente o sin autorización de su autor.
- Integridad y confiabilidad, que consiste en la certeza que el mensaje transferido no ha sido modificado en ninguna parte del circuito o proceso de transferencia.

Por definición, el tema de la seguridad de los datos distingue tres niveles:

1. Seguridad a nivel de los sistemas
2. Seguridad en recurso y servicios
3. Seguridad en la información

En nivel de los sistemas, el administrador tiene la opción de controlar las contraseñas, el acceso de usuarios y la generación de informes que den cuenta de anomalías o problemas

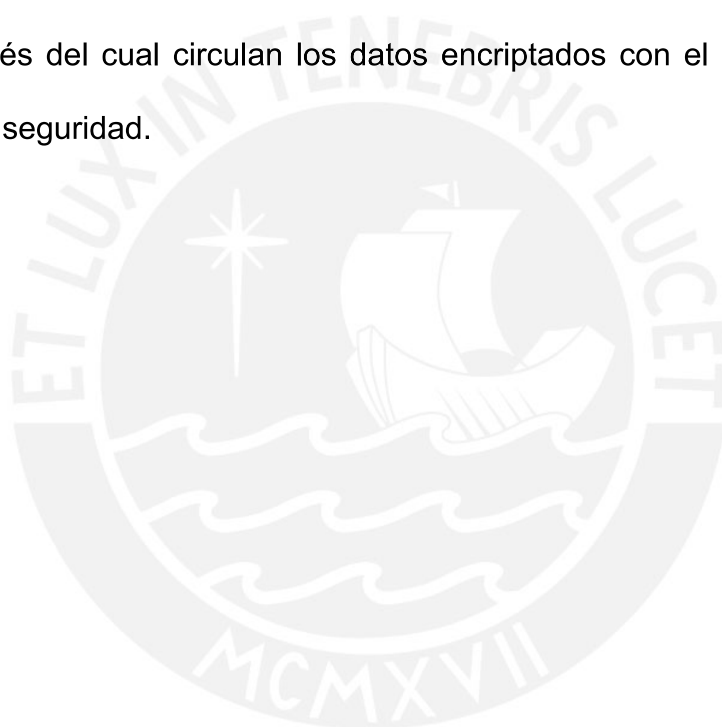
Bajo el nivel de recursos y servicios, las medidas de seguridad están orientadas a proteger la red misma y los recursos y servicios involucrados, como ancho de banda, tiempo de respuesta, acceso a servidores de la red, etc. Estas consisten en la instalación de mecanismos o dispositivos denominados cortafuegos (firewalls), cuya función principal es mantener un control del acceso a

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

la red y los recursos. Este puede ser del tipo filtro de paquetes, compuerta (gateway) a nivel de circuito o compuerta (gateway) a nivel de aplicación.

El nivel de seguridad de la información es probablemente el más importante, pues es aquél donde más participa el usuario y distingue varias clases y niveles. A nivel de protocolos, el IP provee dos cabeceras de datagramas destinados a la seguridad y a nivel de transporte, el Netscape propuso el estándar SSL (Secure Socket Layer), que genera una túnel virtual entre el cliente y el servidor, a través del cual circulan los datos encriptados con el sistema DES en condiciones de seguridad.



SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

CAPITULO II

ENCRIPTACION RSA

En criptografía, RSA es un algoritmo para encriptación con clave pública. Es utilizado ampliamente en protocolos de comercio electrónico. Sujeto a múltiples controversias, desde su nacimiento nadie ha conseguido probar o rebatir su seguridad, y se le tiene como uno de los algoritmos asimétricos más seguros[3].

Debe su nombre a sus tres inventores: Ronald Rivest, Adi Shamir y Leonard Adleman, y desde 1983 estuvo bajo patente de los RSA Laboratories hasta el 20 de setiembre del 2000, por lo que su uso comercial estuvo restringido hasta esa fecha. De hecho, las primeras versiones de PGP lo incorporaban como método de cifrado y firma digital, pero se desaconsejó su uso a partir de la versión 5 en famor de otros algoritmos, que por entonces sí eran libres.

La seguridad del RSA se basa en el hecho de que no existe una forma eficiente de factorizar números que sean productos de dos grandes primos. Las claves pública y privada se calculan a partir de un número que se obtiene como producto de dos primos grandes. El atacante, si quiere recuperar un texto claro a partir del criptograma y la llave pública, a un problema de factorización. SI alguien conoce la clave pública y la factorización del modulo, podrá calcular d.

[3] Manuel Lucena, “Criptografía y Seguridad en Computadores”, 3ra Edición actualiza y ampliada, v2.01 (3 de marzo de 2003) tomado el 11 de abril del 2005 de http://www.criptored.upm.es/guiateoria/gt_m027a.htm.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Dicho de otro modo, el RSA se basa en el uso de una función matemática que es fácil de calcular pero es difícil de invertir. La única manera de invertirla es utilizando la clave privada. Este tipo de funciones es conocida como funciones trampa de un solo sentido pues requieren que se conozca un secreto (que simboliza la trampa).

Generación de Claves

A continuación enunciaremos los pasos a seguir para la generación de claves RSA[4]

1. Escoger aleatoriamente dos primos grandes ***p*** y ***q*** tales que sean diferentes
2. Calcular ***n*** = ***p q***
3. Calcular ***z*** = (***p***-1) (***q***-1), función de Euler
4. Escoger ***e*** = 3, coprimo con ***z***, si es necesario recalcular ***p*** y ***q***
5. Calcular ***d*** tal que ***d e*** = 1 mod ***z***

La clave pública esta compuesta por el par (***e***, ***n***) donde ***n*** se conoce como ***módulo*** y ***e*** el exponente de cifrado. Es utilizada por los emisores para que encripten el mensaje original.[5]

La clave privada esta compuesta por el par (***d***, ***n***) donde ***d*** es el exponente de descifrado. Es utilizada por el receptor para recuperar el mensaje original a través de la decriptación. Debe notarse que sólo d es un secreto ya que n también es conocida por el público.

[4] "RSA", tomado de http://en.wikipedia.org/wiki/RSA el 16 de febrero del 2005
[5] "Public Key Cryptography", Robert Rolland, Diciembre 2001, tomado el 12 de abril del 2005 de http://tlapixqui.izt.uam.mx/cimpa/files/rolland/pub_key.pdf

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Es de notar que esta no es la única forma de calcular las claves. El PKCS # 1 v2.0 incluye otra forma de cálculo utilizando en Teorema Chino del Resto, e incluso propone un clave multiprimo, sin embargo ambos puntos están fuera del alcance de este trabajo.

Algunas consideraciones sobre las claves

- RSA no es seguro para valores del exponente d menores que un determinado umbral (Boneh-Durfee $d < n^{0.292}$)
- n debe tener en la práctica por lo menos 700 bits, aunque mientras más tenga es mejor.
- La seguridad del RSA depende en gran parte de la dificultad de factorización. Por esta razón, algunos valores de n deben evitarse por sus características. Por ejemplo si p-1 es producto de factores primos pequeños, es posible factorizar n según el método de Pollard. Pero esto casos ocurren con poca probabilidad sin embargo es posible incluir alguna prueba en la selección de p y q para descartar estos eventos.
- El método más rápido para encontrar primos grandes es el uso de una prueba de primalidad probabilística, como la prueba de Miller-Rabin. Una prueba de este tipo no asegura completamente que el valor en análisis sea primo, pero la probabilidad de que no lo sea es muy baja.
- Para el cálculo de e se prefiere un valor pequeño conocido con el fin de agilizar la encriptación. Puede ser 3 ó 65537. La seguridad del sistema RSA no se ve afectada.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

- El cálculo de d se puede realizar fácilmente aplicando el Algoritmo Extendido de Euclides ya que el módulo n y el exponente de encriptación e son conocidos
- Seguridad Semántica significa que ninguna información del mensaje original puede ser recuperado del mensaje encriptado. RSA no cumple con esta característica. Por ejemplo, si el texto claro es representado en valores ascii y se encripta carácter por carácter, el sistema puede ser fácilmente quebrado. Es esquema es malo. El estándar PKCS # 1 v2.1 recomienda el uso de RSAES-OAEP (RSA Encryption Scheme – Optimal Asymmetric Encryption Padding). Este estándar escapa del alcance de este trabajo.
- Es probable que pueda quebrarse la clave RSA sin factorizar el módulo n, sin embargo, esto aún no ha sido demostrado..

Encriptación de Mensajes

Para la encriptación de un archivo de texto se requiere que cada carácter de dicho archivo sea convertido a número. La manera más común de hacerlo es convirtiendo cada carácter a su correspondiente código ASCII y sumándole 100 para así trabajar con valores de tres dígitos. Por lo general esta larga secuencia de números es almacenada en otro archivo.

A continuación se conformarán mensajes m para cada bloque de j caracteres que puede conformarse. j debe determinarse de tal manera que $10^{(j-1)}$

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

< $n < 10^j$ pues un mensaje m mayor al módulo n no podrá encriptarse. [6]. En caso no contar con los caracteres suficientes para último bloque, este será rellenado con ceros hasta alcanzar la longitud j requerida.

Se encriptará cada bloque M así formado con la siguiente expresión

$$c = m^e \bmod n$$

que representa la potencia modular e módulo n de c. Se obtendrá un mensaje c por cada bloque m que deberá ser almacenado en otro archivo

Decriptación de Mensajes

Para la decriptación de un archivo encriptado se leerá cada bloque guardado anteriormente. Cada bloque representa el mensaje encriptado c que será decriptado con la siguiente expresión:

$$m = c^d \bmod n$$

que representa la potencia modular d módulo n de m. Se obtendrá un mensaje m por cada bloque c que deberá ser almacenado en otro archivo. Este archivo contiene el archivo convertido a números correspondiente al inicial.

Adicionalmente deberá transformarse cada secuencia de tres dígitos a su correspondiente equivalente en carácter ascii.

Ejemplo

Se seleccionan dos números primos:

p=5

q=11

[6] "Introducción a la Criptografía", http://rinconquevedo.iespana.es/rinconquevedo/criptografia/rsa.htm, tomado el 11 de abril del 2005

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Para ilustrar el método se una manera sencilla se han elegido primos muy pequeños.

Se calcula n = p * q = 5 * 11 = 55, por tanto **n=55**

Se calcula la función de Euler z = (p-1) * (q-1) = 4 * 10 = 40, por tanto **z=40**

Se escoge ***e=3***, se verifica que es coprimo con z

Se calcula e d = 1 mod n

3 * 27 = 1 mod 40 (= 81 = 2 * 40 + 1, por tanteo)

por tanto ***d=27***

Para encriptar el mensaje "4", calculamos el tamaño del bloque, aplicando la fórmula $10^{(j-1)} < n < 10^{j}$ y recordando que n es 55.

j=1, $10^{(0)} < n < 10^{1}$, 1 < n < 10

j=2, $10^{(1)} < n < 10^{2}$, 10 < n < 100

Por tanto, el bloque a encriptar debe ser de 2 caracteres de longitud. "4" se convierte en 52. No sumaremos 100 como indicamos en el algoritmo pues obtendríamos un mensaje mayor a 55 el cual ya no podría ser encriptado con estas claves. Se conforma un bloque de 2 caracteres y aplicamos la siguiente fórmula $c = m^{e} \bmod n$:

m=52, $c = 52^{3} \bmod 55$ = 140608 mod 55

= 2556 * 55 + 28

= 28 mod 55

por tanto ***c=28***

Para la decriptación del mensaje c, aplicaremos la fórmula $m = c^{d} \bmod n$:

c=28, $m=28^{27} \bmod 55$ = 1183768682616191959377597437620164493312 mod 55

= 21523066956658035625047226138548445333 * 55 + 52

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

= 52 mod 55

por tanto ***m=52***, el cual convertido a caracter ascii representa el valor original "4"

Matemática del RSA

Esta sección no pretende ser un compendio riguroso de teoremas y demostraciones matemáticas. Es más bien una pequeña recolección de algunos conceptos aplicados en el Algoritmo RSA que deben ser revisados para un mejor entendimiento de las fórmulas requeridas.

- **Aritmética Modular**

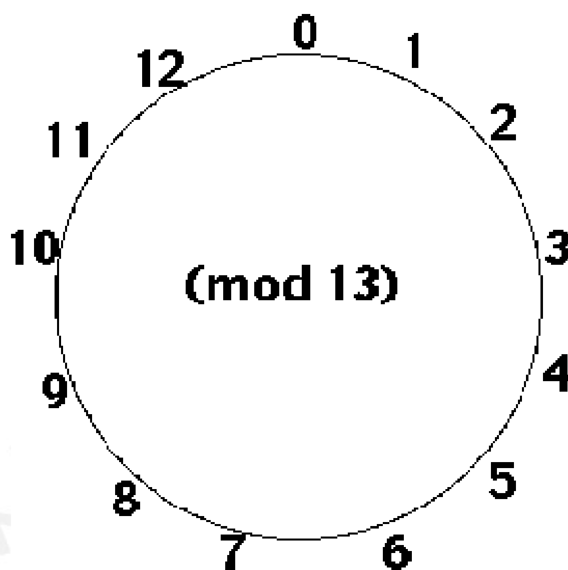
El RSA utiliza aritmética modular. Es similar a la aritmética convencional pero solo utiliza enteros positivos menores a un valor determinado llamado el módulo o modulus. Operaciones como la suma, resta y multiplicación trabajan como en las matemáticas convencionales pero no existe la división. Se puede utilizar cualquier valor como módulo[7]. También es conocida como la aritmética del reloj.

Por ejemplo: asumiendo que los días de la semana se numeran del 0 al 6, éstos utilizan módulo 7 pues al llegar al valor 6 el siguiente día vuelve a tomar el valor 0. El horario utiliza un módulo 60 para los segundos y minutos y un módulo 12 para las horas.

[7] "Mathematics", Paul Johnston, tomado de http://pajhome.org.uk/crypt/rsa/maths.html el 08 de marzo del 2005. Ruta completa Paj's Home: Cryptography: RSA: Mathematics

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ



El diagrama anterior utiliza módulo 13, por lo que la cuenta empieza en 0 y continua 1, 2, ..., 11, 12, 0, 1, ... La notación utilizada para expresiones de aritmética modular es: x = y mod m. Esto se lee como x es equivalente a y módulo m, significa que x e y dejan el mismo residuo cuando son divididos entre m. Por ejemplo: 7 = 23 mod 8 y 22 = 13 mod 9.

La siguiente expresión es un principio básico de aritmética modular:

$$a + kp = a \bmod p$$

Se puede visualizar esto en el diagrama pues cada vez que se suma p, se gira alrededor del círculo de regreso al punto de inicio. No es importante dónde se empieza ni cuán grande es el círculo ni cuántas veces se hizo, sólo que la igualdad siempre se cumple. En la expresión anterior por tanto el valor de k no es significativo.

- **Primalidad y Coprimalidad**

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Un número es primo si los únicos números por los que puede ser dividido exactamente son el 1 (o –1) y él mismo. Por ejemplo, 17 es primo pero 15 no lo es porque es divisible por 3 y 5[8]. De manera equivalente podemos decir que dos números serán primos si su Máximo Común Divisor (MCD) es 1.[9]

Un par de números son coprimos si el número más grande que divide a ambos es el 1.Los números en realidad no necesitan ser primos. Por ejemplo, 8 y 9 son coprimos pero ninguno de los dos es primo, 8 y 10 no son coprimos pues ambos son divisibles por 2 y mucho menos primos. El número 1 es coprimo con todos los enteros y 0 solo es coprimo con 1 y –1.

Es de notar que si se tiene un par de números primos distintos, ellos siempre serán coprimos entre sí.

Para determinar si dos números son coprimos, bastará hallar el MCD y comprobar que éste es 1. Para esto se utilizará el Algoritmo de Euclides o el Algoritmo Extendido de Euclides.

- **Prueba de Primalidad de Miller-Rabin**

Antes de referirnos a esta prueba, indicaremos que una prueba de primalidad es un algoritmo para determinar si un número es primo. No es lo mismo verificar la primalidad que factorizar el número en sus componentes enteros. La verificación de primalidad es una operación relativamente sencilla de realizar mientras que la factorización, en la actualidad 2005, es un problema difícil de

---

[8] "Mathematics", Paul Johnston, tomado de http://pajhome.org.uk/crypt/rsa/maths.html el 08 de marzo del 2005. Ruta completa Paj's Home: Cryptography: RSA: Mathematics

[9] "Coprime", http://en.wikipedia.org/wiki/Coprime tomado el 16 de febrero del 2005

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

resolver computacionalmente.[10]

Existen diversos métodos para la comprobación entre métodos determinísticos y probabilísticos. El método más sencillo entre los determinísticos, es el comprobar si el número en cuestión es divisible por alguno de los primeros números enteros empezando desde el 2 hasta la raíz cuadrada del mismo número. Una mejora que se puede aplicar a este algoritmo es considerar sólo los números primos existentes dentro de ese rango.

Sin embargo, las más populares son las pruebas probabilísticas. No determinan con certeza si un número es primo o no, pero son aceptables para aplicaciones prácticas tales como la criptografía que dependen de manera crítica de números primos grandes. La prueba probabilística más simple es la prueba de Fermat. Las pruebas de Miller-Rabin y Solovay-Strassen son variantes más sofisticadas que detectan todos los números compuestos.

La versión original de la prueba de Miller-Rabin fue creada por G.L.Miller y era determinística. M.O.Rabin la modificó para obtener un algoritmo probabilístico incondicional que se basa en la hipótesis de Riemann[11].

La prueba consiste en la aplicación de un conjunto de igualdades que sólo son verdaderas para números primos con la ayuda de un testigo que no es nada mas que un número generado aleatoriamente. Esta prueba se ejecuta las veces que se estime conveniente y si todas ellas con superadas, entonces el número es primo.

---

[10] "Primality Test", http://en.wikipedia.org/wiki/Primality_test tomado el 16 de febrero del 2005

[11] "Primality Test", http://en.wikipedia.org/wiki/Primality_test tomado el 16 de febrero del 2005

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Sea n un primo impar, puede escribirse n –1 como 2^{s} r donde s es un entero y d es impar. Esto es lo mismo que factorizar 2 repetidamente. Entonces alguna de las siguientes igualdades deben ser ciertas para algún a

$a^{r} = 1 \bmod n$

o

$a^{2jr} = -1 \bmod n$, para algún 0 <= j <= s-1

Esto puede demostrarse utilizando el Pequeño Teorema de Fermat, lo cual escapa del alcance de este trabajo. Si todas las pruebas ejecutadas para la verificación de primalidad son superadas, se puede concluir que el número es primo, es caso contrario es compuesto.

A continuación presentamos el algoritmo computacional más difundido.[12]

Entrada: n: valor a ser probado primo; t: número de veces a probar la primalidad

Salida: Compuesto si n es compuesto o de lo contrario retorna Primo

Algoritmo

```
Escribir n - 1 = 2^s*r tal que r es impar
Repetir de  1 a t
        Escoger un entero aleatorio que satisface 1 < a < n - 1
        Calcular y = a^r mod n
        Si (y <> 1) && (y <> n-1) entonces
        j := 1
        Mientras (j < s) && (y <> n - 1) entonces
                y := y2 mod n
                Si y = 1 entonces devolver COMPUESTO
                j := j + 1
        FinMientras
        Si (y <> n - 1) entonces devolver COMPUESTO
        FinSi
Devolver PRIMO
```

[12] “Miller-Rabin Primality Test”, http://www.cryptomathic.com/labs/rabinprimalitytest.html tomado el 21 de febrero del 2005

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

- **Algoritmo Extendido de Euclides**

El Algoritmo Extendido de Euclides es una versión del algoritmo Euclideano; sus entradas son dos enteros a y b y el algoritmo calcula su Máximo Común Divisor así como enteros x e y tales que ax + by = MCD(a,b). Esta ecuación es particularmente útil cuando a y b son coprimos, es decir MCD (a,b) será igual a 1 y x será entonces la inversa multiplicativa de a módulo b.[13]

La idea básica el algoritmo de Euclides es reemplazar repetidamente los números originales por números más pequeños que tienen el mismo MCD hasta que alguno de los números se convierte en cero. El algoritmo Extendido de Extendido de Euclides permite obtener junto con el MCD, los coeficientes u y v en las sucesivas iteraciones que son los que se convierten en x e y de la igualdad indicada en el párrafo anterior.

Los coeficientes u y v se calculan en base a resultados anteriores y a los cocientes obtenidos y con las siguientes fórmulas:

$$u = u_{n-2} - q_n * u_{n-1}$$

$$v = v_{n-2} - q_n * v_{n-1}$$

donde q representa a los cocientes obtenidos y n es el número de iteración.

Ejemplo:

Sean a=23, b=120

n	q	r	$u = u_{(n-2)} - q_{(n)} * u_{(n-1)}$	$v = v_{(n-2)} - q_{(n)} * v_{(n-1)}$
-2	-	23	1	0
-1	-	120	0	1
0	0	23	1	0
1	5	5	-5	1

[13] http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm tomado el 23 de julio del 2005

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

2	4	3	21	-4
3	1	2	-26	5
4	**1**	1	**47**	-9
5	2	0	-120	23

Las iteraciones –2, -1 y 0 se cargan con valores iniciales tales como los valores de a y b, colocando el menor primero, y los valores 1 y 0 para los coeficientes u y v. A partir de la iteración 1 se procede a calcular los cocientes, restos y coeficientes u y v correspondientes. Este procedimiento se repite hasta obtener 0 como resto. El MCD e Inversa serán el resto y el coeficiente v correspondiente a la penúltima iteración.

MCD (a,b) = MCD(23,120)=1 (resto de iteración 4)

= 47*23-9*120

Inversa = 47 (coeficiente u de iteración 4)

En caso el coeficiente u tome un valor negativo, debe tomarse el complemento con respecto al número mayor para la inversa.

- **Potencia Modular**

La potencia modular está representada por la expresión: $c = x^{e} \bmod n$. Esta operación consiste en devolver el residuo en n de elevar x a la e. Para esto se utiliza el algoritmo square-and-multiply (elevar al cuadrado y multiplicar) que aplica para el cálculo de potencias de enteros grandes. También es conocido como exponenciación binaria debido a que el exponente es convertido a su equivalente binario. Se aplica no sólo a la aritmética estándar sino también a la modular.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Se aplica el siguiente algoritmo recursivo[14]

Potencia (x,n)

Entrada: x base de potencia, n potencia

Salida

x,	si n = 1
Potencia (x^2, n/2),	si n es par
x * Potencia (x^2, (n-1)/2),	si n > 2 es impar

Comparado con un método ordinario para multiplicar x consigo mismo n-1 veces, este algoritmo permite multiplicaciones más rápidas reduciendo el tiempo casi a la mitad. La misma idea se aplica a potencias modulares.

Para ilustrar cómo se reduce el número de operaciones requeridas, veamos el cálculo de 13789^{722341} mod 2345

Si aplicamos el método tradicional de cálculo, primero hallaríamos la potencia 722341 de 13789 para luego hallar el resto de dividir dicho número entre 2345. Aún con un método más efectivo, el cálculo tomará mucho tiempo: elevar 13789 al cuadrado, tomar el resto de la división entre 2345, multiplicar el resultado por 13789 y así sucesivamente. Esto tomará 722340 multiplicaciones modulares. El algoritmo elevar al cuadrado y multiplicar se basa en la observación de que $13789^{722341} = 13789\,(13789^{2})^{361170}$. Así si calculamos 13789^{2}, el cálculo completo solo tomaría 361170 multiplicaciones modulares. Esta es la ganancia de un factor dos pero como el nuevo problema es del mismo tipo, podemos aplicar la misma observación de manera sucesiva y cada vez reduciendo la cantidad de iteraciones

[14] "Exponentiation by squaring", http://en.wikipedia.org/wiki/Square-and-multiply_algorithm tomado el 23 de julio del 2005

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

a la mitad.

La aplicación repetida de esta algoritmo es equivalente a descomponer el exponente, previamente convertido a binario, en una secuencia de cuadrados y productos. Así tendríamos:

$x^{13} = x^{1101}$, 13 es 1101 en binario

$= x^{(1*2^3 + 1*2^2 + 0*2^1 + 1*2^0)}$

$= x^{1*2^3 * x1*2^2 * x0*2^1 * x1*2^0}$

$= x^{2^3 * x2^2 * 1 * x2^0}$

$= x^8 * x^4 * x^1$

$= (x^4)^2 * (x^2)^2 * x$

$= (x^4 * x^2)^2 * x$

$= ((x^2)^2 * x^2)^2 * x$

$= ((x^2 * x)^2)^2 * x$

Este algoritmo sólo necesita 5 multiplicaciones en lugar de 13-1=12. Esta demostración lleva a formular el siguiente algoritmo

Entrada: x base de potencia, e potencia, n módulo

Salida: c = Potencia e de x modulo n

Algoritmo

Obtener la representación binaria del exponente e

c = x

Para i = n-2 hasta 0 hacer

$c=c^2$ mod n

Si e_i =1 entonces c = c.x mod n

FinPara

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Devolver c

Aquí e_i representa cada dígito del número binario que representa el exponente binario.

Ejemplo

Calcular 123^{27} mod 678

27 = (11011)

Número binario 11011	resultado	módulo	Operación
1	15129	**213**	c=c2 mod n
	26199	**435**	c=c.x mod n
0	189225	**63**	c=c2 mod n
1	3969	**579**	c=c2 mod n
	71217	**27**	c=c.x mod n
1	729	**51**	c=c2 mod n
	6273	**171**	c=c.x mod n

Resultado= 171 mod 678

Ataques

Para analizar la seguridad del sistema, se supone que el criptoanalista tiene una cantidad ilimitada de pares (m,c) de mensajes originales y sus correspondientes criptogramas. Las posibles maneras que tiene de atacar el sistema son las siguientes.

1. Factorizar *n*.
2. Calcular $\Phi(n)$.
3. Ataque por iteración.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

4. Ataque de Blakley y Borosh.

5. Ataque por tiempo

6. Ataque adaptable de texto encriptado seleccionado

Vamos a analizar estos posibles ataques:

1. Factorizar *n*:

De esta forma obtiene el número Φ(*n*)=(*p*-1)*(*q*-1), y con el la clave privada *d*, puesto que *e* es pública y se cumple: *e***d*-1(mod Φ(*n*)).

Al ser *n* el producto de solo dos números primos, un algoritmo de factorización requiere como máximo ‘*n* pasos, pues uno de los dos factores es necesariamente un primo menor que ‘*n*. Sin embargo, si *n* fuera el producto de N>2 primos, un algoritmo de factorización necesitaría como máximo *n*^(1/N) pasos, que es una cota menor que ‘*n*, por lo que se concluye que es adecuada la obtención de *n* como producto de solo dos números primos.

Con respecto al estudio del problema de la factorización, hay que mencionar al precursor de la moderna factorización, el algoritmo de fracciones continuas de Morrison-Brillhart, ya que es uno de los más rápidos. Sin embargo, los dos algoritmos de factorización que resultan más prácticos para grandes enteros corresponden al de factorización con curvas elípticas de Hendrik Lenstra y al de factorización con filtro cuadrático de Carl Pomerance. Ambos algoritmos convierten el problema de la factorización de un entero *n* en el problema de encontrar soluciones no triviales ('x <> y(mod *n*)' y 'x <> -y(mod *n*)') de la ecuación: x^3-y^3 (mod *n*). Si se supone que ni (x+y) ni (x-y) son múltiplos de *n* enteros, se deduce que el m.c.*d*.(x+y,*n*) o bien el m.c.*d*.(x-y,*n*) es con seguridad un factor no trivial de *n*, por lo que se resuelve el problema de la factorización.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Por ejemplo, si *n*=97343, entonces la ecuación x^2-y^2 (mod 97343) es fácilmente resoluble por ser los factores de *n* dos números primos muy cercanos. Como 312^{2-}1(mod 97343) y ni 313 ni 311 son múltiplos de 97343, se concluye que 313 y 311 son los factores de *n*.

2. Calcular Φ(*n*):

Como se ve a continuación, esta manera es equivalente a la anterior. Si se tiene Φ(*n*), dado que *p*+*q*=*n*-Φ(*n*)+1 y a partir de la suma se puede calcular $(p\text{-}q)^2$ por coincidir con $(p\text{-}q)^2$-4**n*, luego se consigue la factorización mediante las formulas *q*=[(*p*+*q*)-(*p*-*q*)]/2 y *p*=[(*p*+*q*)+(*p*-*q*)]/2.

3. Ataque por iteración:

Si un enemigo conoce (*n*,*e*,C), entonces puede generar la secuencia:

C1-C^*e*(mod *n*), ..., Ci-[C(i-1)]^*e*(mod *n*),

con lo que si existe algún Cj tal que C=Cj se deduce que el mensaje buscado es M=C(j-1) pues [C(j-1)]^*e*=Cj=C. Ahora bien, en cuanto la igualdad Cj=C se cumple solo para un valor de j demasiado grande, este ataque se vuelve impracticable. Con respecto a esto, Rivest demostró que si los enteros *p*-1 y *q*-1 contienen factores primos grandes, la probabilidad de éxito mediante este procedimiento es casi nula para grandes valores de *n*.

4. Ataque de Blakley y Borosh:

El sistema RSA, además, tiene una característica muy peculiar, advertida por Blakley y Borosh, y es que no siempre esconde el mensaje. A continuación vemos un ejemplo que lo muestra.

Si *e*=17, *n*=35 y los mensajes a cifrar son M1=6 y M2=7, entonces se obtiene que 6^17-6(mod 35) y 7^17-7(mod 35). Una situación más peligrosa para

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

el sistema aparece, por ejemplo, con los valores *p*=97 *q*=109 y *e*=865, ya que el criptosistema resultante no esconde ningún mensaje, pues M^865-M(mod 97*109)

En general, lo ocurrido en el ultimo ejemplo ocurre siempre que *e*-1 es múltiplo de *p*-1 y *q*-1, pues en ese caso M^*e*-M(mod *p***q*). Además, se tiene que para cualquier elección de *n*=*p***q* siempre existen al menos 9 mensajes M que no se cifran en realidad, ya que verifican la ecuación M^*e*-M(mod *n*). De esos 9 mensajes hay tres fijos, que son M perteneciente a {0,1,-1}. Para hacer que el sistema RSA sea resistente contra ataques basados en este hecho, es conveniente elegir como claves privadas números primos de la forma *p*=2**p*'+1, donde *p*' es un primo impar.

5. Ataque por tiempo

Kocher describió un ingenioso y nuevo ataque contra RSA en 1995: si el atacante Eve conoce el hardware de Alice en suficiente detalle y es capaz de medir los tiempos de decriptación para diversos textos encriptados conocidos, entonces ella podrá deducir la clave de decriptación d rápidamente. Este ataque también puede ser aplicado al esquema de firma RSA. Una manera de evitar este ataque es asegurarse que las operaciones de decriptación tomen una cantidad constante de tiempo para cada texto crifrado. Otra manera es usar la propiedad multiplicativa del RSA. En lugar de calcular $c^d \bmod N$, Alice primero escoge un valor aleatorio r secreto y calcula $(r^e c)^d \bmod N$. El resultado de este cálculo es *rm mod N* y así el efecto de *r* puede ser removido multiplicando es resultado por su inversa. Un nuevo valor de r es elegido por cada texto cifrado.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

6. Ataque adaptable de texto encriptado seleccionado[15]

Un ataque adaptable de texto encriptado seleccionado es una forma interactiva de ataque según el cual un atacante envía un cierto número de textos cifrados a ser decriptados, y luego utiliza los resultados de estas decriptaciones para seleccionar subsecuentes textos encriptados. La meta de este ataque es revelar gradualmente información sobre el mensaje encriptado, o sobre la clave de decriptación. Para sistemas de clave pública, este método es generalmente aplicable sólo cuando se tiene la posibilidad de manejar el texto encriptado. – esto es, un texto encriptado puede ser modificado de maneras específicas para que tengan efectos predecibles en la decriptación de ese mensaje.

Ataques Prácticos

Los ataques adaptables de texto encriptado seleccionado fueron por mucho tiempo considerados como cuestiones meramente teóricas hasta 1998, cuando Daniel Bleichenbacher de Bell Laboratories demostró un ataque práctico contra sistemas que utilizaban encriptación RSA y la función de codificación enunciada en el PKCS # 1, incluyendo además una versión del protocolo SSL (Secure Socket Layer) utilizado por miles de servidores web por ese entonces.

Los ataques de Bleichenbacher tomaron ventaja de defectos del PKCS # 1 para gradualmente revelar el contenido de un mensaje encriptado en RSA y potencialmente revelar las claves de la sesión[16]. Hacer esto requiere enviar muchos millones de textos encriptados de prueba al dispositivo de decriptación. En términos prácticos, esto significa que la clave de una sesión SSL puede ser

[15] "Adaptive chosen ciphertext attack",, tomado el 16 de febrero del 2005 de http://en.wikipedia.org/wiki/Adaptive_chosen_ciphertext_attack
[16] "RSA", tomado el 16 de febrero del 2005 de http://en.wikipedia.org/wiki/RSA

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

expuesta en una razonable cantidad de tiempo, tal vez un día o menos.

Previsión de ataques

Para prevenir ataques con este método, es necesario utilizar un esquema de encriptación o codificación que limite el manejo del texto cifrado. Se han propuesto numerosos esquemas de codificación; el estándar más común para encriptación RSA es el OAEP (Optimal Asymmetic Encryption Padding) que es un esquema de relleno probablemente seguro. RSA Laboratories ha publicado también nuevas versiones de PKCS # 1 que no son vulnerables a estos ataques.

CONCLUSION

Para la selección de los parámetros del sistema el usuario debe comprobar las siguientes condiciones para intentar evitar los ataques mencionados:

- Los números primos p y q deben tener una diferencia grande (ataque 1).
- Los números primos p y q deben ser del orden de 100 dígitos (ataque 1).
- Los enteros (p-1) y (q-1) deben contener grandes factores primos (ataque 3).
- Los números primos p y q deben elegirse de manera que $p=2*p'+1$ y $q=2*q'+1$ con p' y q' números primos impares (ataque 4).

Los primos p y q del RSA deben ser de la forma x=2*x'+1 (siendo x' un número primo impar) tal que x-1 tiene grandes factores primos, con aproximadamente 100 dígitos y de forma que p-q sea grandes.

No obstante, es lógico pensar que a medida que se profundiza en la

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

investigación se añadirán y descubrirán nuevas propiedades de los parámetros que deban verificar para que el sistema resultante se fortalezca.

Uno de los más recientes ataques al RSA fue realizado por Wiener. Llevo a cabo un criptoanalisis en tiempo polinomial de un sistema RSA con claves privadas pequeñas. Para ello, utilizo un algoritmo que representa los números racionales como fracciones continuas finitas.

De todo lo anterior se concluye que para que la seguridad del sistema RSA quede perfectamente salvaguardada es necesario escoger cuidadosamente las claves a utilizar.

Finalmente se presentan algunos resultados que versan sobre la seguridad del sistema. Un algoritmo que calcule d se puede convertir en un algoritmo probabilístico que factorice n.

Si el RSA se aplica en un entorno en el que el módulo n y los exponentes de cifrado y descifrado d son distribuidos por una agencia de manera que esta proporciona un módulo n común a todos los usuarios, los exponentes de cifrado de cada usuario eA, eB, ... y distribuye entre los usuarios las claves privadas dA, dB, ..., pero en todo caso mantiene para si los números p y q, entonces cualquier usuario puede determinar en tiempo cuadrático determinístico la clave de descifrado secreta de otro usuario.

Si dos usuarios usan el mismo módulo en un sistema RSA y lo saben, entonces cada usuario puede descifrar los criptogramas enviados al otro.

Luego queda claro que esas condiciones no resultan muy propicias para la seguridad de las comunicaciones.

A pesar de las ventajas evidentes de este esquema frente a los sistemas de

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

clave secreta, hay que subrayar que en la actualidad la principal desventaja del RSA estriba en que es mucho más lento que el DES y que los cifrados en flujo. Como cifras ilustrativas, se puede señalar que el DES trabaja a unos 20 megabits por segundo, mientras que el RSA funciona a una velocidad 1000 veces menor, y aunque la investigación para acelerar el proceso es intensa, es de esperar que esta proporción se mantenga, ya que también se realiza para los cifrados simétricos.

El punto más débil del RSA es su velocidad (comparándola con la de otros cifrados, como el cifrado en flujo).

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

CAPITULO III

DISEÑO E IMPLEMENTACION

Para la implementación de este proyecto nos basamos en el algoritmo definido en el Estándar de Criptografía de Clave Pública (PKCS) públicado por RSA Laboratories en el año 2000. En este capítulo enunciamos el contenido del estándar PKCS para una mejor comprensión del trabajo efectuado y a continuación describimos el diseño e implementación del sistema de encriptación RSA

El algoritmo RSA fue creado por Ronald L. Rivest, Adi Shamir y Leonard Adleman en 1977 en MIT. Las letras RSA son las iniciales de sus apellidos. El algoritmo fue patentado en 1983 en los Estados Unidos de América expirando la patente el 21 de Setiembre del 2000.

La seguridad de este algoritmo esta basada en la dificultad de factorizar un número muy grande. En el 2004, el número más grande factorizado por métodos de propósito general fue de 174 dígitos decimales de largo (en binario fue de 576 bits). Típicamente, las claves RSA tienen 1024-2048 bits de longitud. Algunos expertos creen que una clave de 1024 bits puede ser quebrada en el corto plazo pero nadie cree que la clave de 2048 bits pueda serlo en un futuro previsible.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Algoritmo del RSA

Una clave pública RSA consiste en un par (n, e) de enteros, donde n es el módulo y e es el exponente público. El módulo n es un número compuesto grande (una longitud en bits de por lo menos 1024 es el tamaño recomendado actualmente), mientras que el exponente público e es normalmente un primo pequeño tal como 3, 17 o 65537. En esta especificación, el módulo (n) es el producto de dos primos distintos.

Una clave privada RSA puede tener alguna de las dos siguientes representaciones. Ambas representaciones contienen información sobre un entero d que satisface

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

para todos los enteros x. Esto quiere decir que la clave privada puede ser usada para resolver la ecuación

$$x^{e} \equiv c \pmod{n}$$

en x, i.*e*., calcular la e-ésima raiz de c módulo n.

La primitiva RSAEP de encriptación RSA toma como entrada la clave pública (n, e) y un entero positivo $m < n$ (una representación de mensaje) y retorna el entero $c = m^{e} \bmod n$.

La primitiva RSADP de decriptación RSA toma como entrada la clave privada y un entero $c < n$ (una representación de mensaje encriptado) para retornar el entero $m = c^{d} \bmod n$, donde d es un entero con las propiedades especificadas antes.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Claves RSA

La clave pública RSA

La clave pública RSA tiene 2 componentes

n el módulo, un entero no negativo

e el exponente público, un entero no negativo

En una clave pública RSA válida, el módulo *n* es un producto de dos distintos primos impares *p* y *q*, y el exponente público *e* es un entero entre 3 y *n*-1 que satisface MCD(*e*,*p*-1)= MCD(*e*,*q*-1)=1

La clave privada RSA

Una clave privada RSA puede tener alguna de las siguientes dos representaciones

1. La primera representación consiste en el par (*n*,*d*) donde los componentes tienen los siguientes significados

n El módulo, un entero no negativo

d el exponente privado, un entero no negativo

En una clave privada RSA válida, el módulo *n* es el mismo que el de la correspondiente clave pública y es el producto de dos primos *p* y *q*, y el exponente privado *d* es un entero positivo menor que *n* que satisface

$e \cdot d = 1 \pmod{MCD(p-1,q-1)}$

en donde *e* es el correspondiente exponente público

2. La segunda representación consiste en un quíntuple (*p*,*q*,*dP*,*dQ*,*qInv*), donde los componentes tienen el siguiente significados

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

p el primer factor, un entero no negativo

q el segundo factor, un entero no negativo

dP el exponente del primer factor, un entero no negativo

dQ el exponente del segundo factor, un entero no negativo

qInv el coeficiente CRT (Chinese Remainder Theorem), un entero no negativo

En una clave privada RSA válida con esta representación, los dos factores *p* y *q* son los factores primos del módulo *n*, los exponentes *dP* y *dQ* son enteros positivos menores que *p* y *q* respectivamente satisfaciendo

$e \,.\, dP \equiv 1 \pmod{p - 1};$

$e \,.\, dQ \equiv 1 \pmod{q - 1}.$

y el coeficiente CRT *qInv* es un entero positivo menor que *p* satisfaciendo

$q \,.\, qInv \equiv 1 \pmod{p}$

Primitivas

RSAKG Generación de Clave RSA

Este algoritmo produce un par de claves con el módulo de la longitud en dígitos decimales deseada y un exponente público predeterminado. Se requieren 2 números primos, cada uno de los cuales es generado por una subrutina PG que toma un intervalo y el exponente público como datos de entrada y devuelve un primo aleatorio *p* que pertenece al intervalo, de tal manera que *p* - 1 es relativamente primo con el exponente público.

RSAKG (*L* , *e*)

Entrada: L la longitud en dígitos decimales deseada para el módulo

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

e el exponente público, un entero impar mayor a 1

Salida: *K* una clave privada valida en cualquiera de las dos representaciones estándar con un módulo de longitud de L bits.

(*n* , *e*)una clave pública válida

1. Generar un primo *p* tal que $[2^{(L-1)/2}] + 1 \leq p \leq [2^{L/2}] - 1$ y tal que MCD(*p* , *e*) = 1 usando PG con input ($[2^{(L-1)/2}] + 1$, $[2^{L/2}] - 1$, *e*).
2. Generar un primo *q* tal que $[2^{(L-1)/2}] + 1 \leq q \leq [2^{L/2}] - 1$ y tal que MCD(*q* , *e*) = 1 usando PG con input ($[2^{(L-1)/2}] + 1$, $[2^{L/2}] - 1$, *e*).
3. Asignar $n = pq$. La clave pública es (*n* ,*e*)
4. Hacer los cálculos necesarios para la clave privada K
5. Mostrar las claves pública y privada.

PG Generación de primos

El algoritmo utiliza la prueba probabilística de primalidad de Miller-Rabin, no verifica que los primos sean "fuertes"

PG (*r* , *s* , *e*)

Input *r* limite inferior para el primo a ser generado

s limite superior para el primo a ser generado

e un entero positivo impar

Output *p* un primo impar uniformemente escogido del intervalo [*r*,*s*] tal que MCD(*p* - 1 , *e*) = 1

1. Generar un número aleatorio impar *p* uniformemente en el intervalo [*r* - 2 , *s* -

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

2].

2. Asignar $p \leftarrow p + 2$
3. Si p es divisible por digamos cualquiera de los 2000 menores primos, retornar al paso 2.
4. Si el MCD(p - 1 , e) ≠ 1, entonces retornar al paso 2
5. Asignar v, w tal que w es impar y $p - 1 = w2^v$
6. Escoger un entero positivo t tal que la prueba de primalidad del paso 8 es exitoso con una suficientemente amplia probabilidad.
7. Asignar $i \leftarrow 1$
8. Mientras $i \leq t$ hacer
   - 8.1. Generar un entero aleatorio a de modo uniforme del intervalo [1, p - 1]
   - 8.2. Asignar $b \leftarrow a^w \bmod p$
   - 8.3. Si $b = 1$ o $b = p - 1$, entonces ir al paso 8.6
   - 8.4. Asignar $j \leftarrow 0$
   - 8.5. Mientras $b \neq p - 1$ hacer
     - 8.5.1. Asignar $j \leftarrow j + 1$
     - 8.5.2. Si $j = v$, entonces retornar al paso 2 (p es compuesto)
     - 8.5.3. Asignar $b \leftarrow b^2 \bmod p$ (= $a^{w2^\wedge j} \bmod p$)
     - 8.5.4. Si $b = 1$, entonces retornar al paso 2 (p es compuesto)
   - 8.6. Asignar $i \leftarrow i + 1$
9. Si p es mayor que s, retornar al paso 1
10. Mostrar p

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Encriptación RSA

<u>RSAEP</u> PRIMITIVA DE ENCRIPTACION CON RSA

RSAEP ((*n*,*e*), m)

Entrada

(*n*,*e*) clave pública RSA

m representación del mensaje, un entero entre 0 y *n*-1

Salida

c representación del texto cifrado, un entero entre 0 y *n*-1

Errores

"*Representación* de mensaje fuera de rango"

Supuestos

La clave pública (*n*,*e*) es válida

1. Si la representación del mensaje m no está entre 0 y *n*-1, mostrará "representación de mensaje fuera de rango" y se detendrá

2. Sea c= m^e mod *n*

3. Mostrar c

Desencriptación RSA

<u>RSADP</u> PRIMITIVA DE DESENCRIPTACION CON RSA

RSADP (K,c)

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Entrada K clave privada RSA donde K tiene una de las siguientes formas

1. un par (*n*,*d*)
2. un quintuple (*p*,*q*,*dP*,*dQ*,*qInv*)

e *Representación* de mensaje, un entero entre 0 y *n*-1

Salida m *Representación* de mensaje, un entero entre 0 y *n*-1

Errores "*Representación* de texto cifrado fuera de rango"

Supuesto La clave privada K es valida

1. Si la representación del texto cifrado c no está entre 0 y *n*-1, mostrará "*Representación* de texto cifrado fuera de rango" y se detendrá

2. Si la primera forma es *(n,d)* de *K* es usada:

2.1 Sea $m = c^d \bmod n$

Si no, si la segunda forma (*p, q, dP, dQ, qInv*) de *K* es usada:

2.2 Sea $m_1 = c^{dP} \bmod p$

2.3 Sea $m_2 = c^{dQ} \bmod q$

2.4 Sea $h = (m_1 - m_2) . qInv \bmod p$

2.5 Sea $m = m_2 + q. h$

3. Mostrar m

Implementación

El Sistema de Encriptación RSA permite generar claves, encriptar y desencriptar archivos de texto bajo el algoritmo RSA que hemos presentado en las secciones anteriores. Aquí mostraremos la organización interna bajo la cual se ha construido el sistema, los algoritmos aplicados y otras consideraciones que se han debido tomar en cuenta para el desarrollo del sistema. Muchos casos son

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

ilustrados con ejemplos y corridas de escritorio y para un mejor entendimiento también se adjuntan datos para prueba.

El sistema se ha desarrollado en Java aplicando conceptos de programación orientada a objetos. Podemos referirnos brevemente al sistema indicando que cuenta con una ventana de entrada de parámetros que permite al usuario accionar cada una de las funciones de encriptación, decriptación y generación de claves de manera intuitiva. La funcionalidad de la interfase de usuario se muestra en el Anexo A correspondiente al Manual de Usuario.

Requerimientos

Para la construcción del sistema se han tenido en cuenta los siguientes requerimientos mínimos

- Permitir el manejo de números enteros de por lo menos 128 dígitos para incrementar la seguridad de la claves de encriptación.
- Implementar consideraciones de seguridad mínimas para hacer un sistema resistente a ataques en lo referente a características de los números primos y otros valores que intervienen en el cálculo de las claves.
- Considerar los algoritmos más eficientes disponibles en la teoría matemática con el fin de que el uso de la herramienta computacional sea óptima.
- Interfase gráfica de fácil uso que permita seleccionar los archivos directamente del árbol de directorios.
- Es deseable un adecuado tiempo de respuesta aunque los algoritmos son lentos debido a la gran cantidad de cálculos involucrados

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Herramienta utilizada

Este sistema se desarrolló en Microsoft Java 6.0. El aplicativo se distribuye como un archivo de aplicación de extensión .exe que puede ser ejecutado bajo cualquier entorno Windows.

Organización interna

El sistema ha sido construido en Java y aplica programación orientada a objetos, por tanto se ha definido un número de clases, métodos y atributos para este fin. Esta compuesto por cuatro clases para el almacenamiento ordenado y lógico de los diversos componentes que son las siguientes:

- *Entero.java:* Permite el manejo de números enteros grandes a través de operaciones matemáticas básicas tales como suma, resta, división, multiplicación así como operaciones referidas a aritmética modular como cálculo de potencia e inversa multiplicativa.
- *Archivo.java:* Permite el manejo de archivos de entrada y salida de datos definiendo una serie de métodos que simplifican el manejo de datos, tales como lectura de caracteres desde un archivo de entrada, almacenamiento de datos en archivos, entre otros.
- *RSA.java:* Permite la encriptación y decriptación de archivos así como la generación de claves pública y privada. Se implementa con el uso de primitivas. Utiliza los métodos definidos en las clases Entero y Archivo.
- *Form1.java:* Permite el manejo de la interfase gráfica de usuario. En ella se realiza la invocación a los métodos de encriptación, decriptación y generación de claves según la opción elegida. Asimismo permite la

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

selección y visualización de archivos así como la lectura del archivo de configuración rsa.ini.

Más adelante detallaremos el contenido de cada clase.

Visión General del Sistema

El Sistema de Encriptación RSA permite generar claves de encriptación y decriptación, encriptar archivos de texto y decriptar archivos previamente encriptados en el sistema. Cada uno de estos procesos tiene algoritmos de naturaleza matemática que han sido implementados en el sistema durante la construcción del mismo. A continuación comentaremos algunos de las ideas más importantes plasmadas en el sistema.

Números Grandes

El concepto más importante para la seguridad de este sistema es el de tener la posibilidad de manejar números muy grandes debido a que el tamaño de los mismos significará la mayor o menor dificultad para el quiebre de la clave. Para esto se implementa una estructura denominada Entero que permitirá almacenar un número de cualquier longitud así como efectuar operaciones sobre ellos. La bibliografía encontrada recomienda utilizar números de por lo menos 1024 bits, es decir, 128 dígitos decimales para el cálculo de las claves que además han de ser primos.

Asimismo, la necesidad de efectuar cálculos sobre los números enteros, requiere que la estructura no sólo pueda almacenar los números generados a los cuales nos referimos en el párrafo anterior, sino también los números resultantes de operaciones entre ellos. Debido a esto la estructura debe tener una capacidad mucho mayor para poder almacenar los números en su totalidad.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Números Primos

Los números primos están muy ligados a temas encriptación pues ellos son utilizados no sólo en algoritmos de encriptación RSA sino en muchos otros. Son la base para la generación de claves de encriptación y decriptación. Deben satisfacer ciertos requisitos para que la clave sea sólida que enunciaremos más adelante. En nuestro sistema estos números son generados de manera aleatoria y con la cantidad de dígitos especificada.

Generación de Claves

Para la generación de claves se utiliza números primos aleatorios debidamente seleccionados. La clave privada y pública se generan a la vez y se requiere que sea así pues para una clave pública determinada sólo existirá una clave privada capaz de decriptar lo que la primera encriptó. Sin embargo, el uso de claves podría invertirse debido a que en todos los casos cualquiera de ellas obtiene los valores originales que la otra ocultó a través de la encriptación.

El usuario emisor de la clave remitirá su clave pública a todos aquellos que deseen intercambiar información con él y ellos remitirán la versión encriptada del archivo de texto que deseen intercambiar. Por lo general la clave pública es un valor pequeño al cual corresponderá un valor grande de clave privada, por tanto la encriptación será rápida pero la decriptación será lenta.

Encriptación de Archivos

La encriptación de archivos permitirá modificar el contenido del archivo de entrada de forma tal que su contenido original no pueda ser recuperado. Existen diversos algoritmos con este propósito pero el RSA se caracteriza por ser el más

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

seguro. Para la encriptación se utiliza la clave pública que por lo general es un valor sencillo que hará que la encriptación sea rápida

Decriptación de Archivos

La decriptación de un archivo permitirá recuperar el valor original del texto, se utilizará para esto la clave privada, es decir, un archivo encriptado recepcionado por el emisor podrá ser decriptado sólo por él aplicando la correspondiente clave privada. Este proceso tomará un tiempo inversamente proporcional al tamaño de la clave pública,

Aritmética Modular

Otro aspecto importante es la aplicación de la aritmética modular. Debe recordarse que la clave privada es la inversa multiplicativa de la clave pública. Se aplican diversos algoritmos matemáticos entre los que destacan el algoritmo Extendido de Euclides pues se emplea tanto en el cálculo de la inversa multiplicativa como en el del Máximo Común Divisor. Ambos resultados se aplican tanto en la encriptación como en la decriptación.

Interfase de Usuario

Esta visión general no podría estar completa si no mencionamos el manejo de la interfase de usuario y manejo de archivos. Se permite la fácil selección de archivos a encriptar y decriptar asi como de archivos de claves. El manejo de las extensiones permite identificar el tipo de contenido.

Se cuenta también con el archivo de configuración rsa.ini que contiene 11 valores que pueden ser ingresados por el usuario modificando el comportamiento del sistema. Entre los diferentes valores se tiene la cantidad de dígitos de números

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

primos, la cantidad de dígitos por celda y las extensiones por tipo de archivo, entre otros.

En las próximas secciones describiremos con detalle los algoritmos empleados y otras definiciones requeridas para la implementación de este trabajo.

Descripción del Algoritmo RSA

Respecto a la generación de las claves de encriptación (e,n) y decriptación (d,n), éstas se generan a la vez pues como vimos, para una clave pública determinada sólo existe una clave privada capaz de decriptar lo que la primera encriptó. El punto de partida de las claves es la generación de números primos p y q que deben cumplir con ciertas características para que la clave sea sólida y que mencionamos a continuación

1. p y q son primos generados aleatoriamente
2. Cada uno debe tener 128 dígitos decimales como mínimo
3. p y q deben ser primos relativos
4. $z = (p\text{-}1)*(q\text{-}1)$ debe ser primo con la clave pública e

   El sistema contiene un método para la creación de números primos aleatorios de nTotalDigPrimo dígitos. Cada número es generado dígito a dígito utilizando funciones random de java y para el cual, una vez conformada la totalidad de dígitos requeridos, se verifica la primalidad del mismo aplicando el algoritmo de Miller-Rabin para este fin.[17]

En base a los números p y q obtenidos se construye los valores $n = p*q$ y $z = (p\text{-}1)*(q\text{-}1)$. La clave pública e asume el valor indicado en archivo de

[17] Lo indicado aquí aplica al algoritmo general, sin embargo en la opción de Demo del sistema, el usuario ingresará los números primos y el sistema los asumirá primos pero verificará que sean coprimos

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

configuración aunque se recomienda que tome el valor 3.[18] La clave privada *d* se calcula como la inversa de *e* en módulo *n*.

Tanto la operación aritmética de multiplicación como la de resta, suma y división, entre otras, han sido implementadas con la utilización de la estructura para almacenamiento de números grandes. Asimismo, se desarrollo un método para la asignación de valores a dicha estructura.

Una vez generadas las claves, estas son almacenadas en sendos archivos y en forma de pares para su posterior utilización en encriptación y decriptación. El archivo de clave pública contendrá el par (*e*,*n*) y se utilizará para la encriptación mientras que el de clave privada almacenará el par (*d*,*n*) que será utilizado para decriptación. El uso de claves podría invertirse debido a que en todos los casos cualquiera de ellas obtiene los valores originales que la otra ocultó a través de la encriptación.

Como hemos visto la clave pública es un valor sencillo que permitirá que la encriptación sea rápida. Ante un valor pequeño de clave pública, el valor de la clave privada será grande. Siendo d la inversa modulo n de e, la multiplicación modular de d y e será 1 modulo n.

Para encriptar, el archivo de texto es convertido a números utilizando la codificación ASCII, es decir, cada caracter es convertido a un número de 3 dígitos. A continuación, la secuencia de dígitos generada es leída en bloques conformando un número m más grande. La cantidad de dígitos leídos por bloque depende del valor de n ya que como máximo se tomará tantos dígitos como cifras

---

[18] En la generación de claves demos *e* variará hasta encontrar un valor tal que sea primo con z.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

tenga n-1 de manera tal que ningún valor numérico del bloque conformado sea mayor a n-1.

Cada bloque será enviado a la primitiva de encriptación la cual ejecutará la siguiente operación modular

$c = m^e$ modulo n

donde c es la representación del mensaje cifrado y m el mensaje original. Cada valor de c obtenido es grabado en el archivo encriptado.

Para decriptar, cada número almacenado en el archivo encriptado es leído y enviado a la primitiva de desencriptación que ejecutará la siguiente operación modular

$m = c^d$ modulo n

donde m es el bloque de dígitos correspondiente al mensaje cifrado. Es decir, que una vez concluida la decriptación de todos los números del archivo de encriptación, se tendrá el equivalente a la secuencia de dígitos que conformaban la representación ASCII del texto original. A continuación un método adicional convertirá la secuencia de dígitos en caracteres tomados convenientemente de 3 en 3.

Estructura de datos

Un lenguaje de programación común no permite manejar números enteros de más de 10 ó 12 dígitos. Dado que el éxito de un sistema de encriptación depende en gran medida de la cantidad de dígitos empleados en la generación de claves, se hace necesario definir una estructura que permita almacenar completamente un número entero grande que, de manera confiable, asegure que no habrá pérdida de precisión.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Esto es muy importante pues de no considerarse se corre el riesgo de que los cálculos provoquen un desbordamiento de las variables utilizadas y que por tanto la encriptación y decriptación basados en cálculos matemáticos produzcan resultados incorrectos, impidiendo que se lleve a buen término alguna de las funcionalidades del sistema.

Se define un arreglo de números enteros dobles en el cual el número entero grande será almacenado fraccionado en números más pequeños que no excedan la capacidad de almacenamiento de las celdas del arreglo. Tanto la cantidad de elementos del arreglo como el valor máximo permitido por celda es configurado en el archivo RSA.ini. Este archivo ha de estar ubicado junto con los demás archivos del sistema. Durante la inicialización del sistema este archivo es leído y los valores almacenados en los correspondientes atributos de la clave.

Para el almacenamiento de un número en esta estructura se considera que cada celda que compone el arreglo tendrá un valor numérico máximo y que ningún valor almacenado podrá, bajo ninguna consideración, superar este valor máximo. Las cifras más significativas del entero grande se almacenan en las posiciones mayores del arreglo. Las celdas en exceso de las requeridas para almacenar el número, conservan el valor 0 como parte del manejo estándar de java.

Asimismo se conserva el atributo nUltElem con el índice de la última posición ocupada del número.

Se ilustra a continuación el manejo bajo el formato b-Radix decimal

Valor Entero $a = a_1a_2a_3 \ldots a_{n-2}a_{n-1}a_n$, tal que $n > 0$

Valor Máximo $b = b_1b_2b_3b_4b_5$

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Almacenamiento de Entero

0	1		n-1	n
$a_{n-4}a_{n-3}a_{n-2}a_{n-1}a_n$	…	…	$a_4a_5a_6a_7a_8$	$a_1a_2a_3$

Ejemplo

Para el número 1234567890123456 y siendo 99999 el mayor valor, se tendrá lo siguiente:

El número se fracciona en 4 partes que serían las siguientes

1 23456 78901 23456

Estas se almacenarán en el arreglo de la siguiente manera:

Ejemplo

0	1	2	3		n-1	n
23456	78901	23456	1	...	0	0

nUltElem = 3

Definición de cantidad máxima de celdas

En el siguiente cuadro se presenta la cantidad de dígitos que como máximo puede ocupar cada uno de los números más significativos del sistema.

Variable	Cantidad máxima de dígitos
p	128
q	128
n=p*q	256
z=(p-1)*(q-1)	256

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

e	1
d	256
Potencia Modular	512 (como mínimo)

Es importante estimar estos valores para optimizar la performance del sistema. Si se define la estructura con celdas en exceso ocupará memoria innecesariamente causando que la ejecución sea lenta. Si por el contrario se utilizan menos celdas que las requeridas, el proceso abortará o producirá resultados incorrectos por problemas de precisión.

Por esta razón el total de celdas del arreglo se define en función de la cantidad de dígitos de los números primos a manejar, habiéndose elegido después de muchas pruebas, el factor 4.5. Por ejemplo, si p tiene 128 dígitos, el arreglo tendrá 576 celdas.

Estándar de codificación

Para este trabajo se ha adecuado la “Especificación del Lenguaje Java” de Sun Microsystems, Inc. (http://java.sun.com/docs/codeconv/) como estándar de codificación. Este ha sido aplicado en la construcción del todo el sistema en los aspectos concernientes a documentación, nomenclatura y codificación. Entre sus principales características podemos indicar:

- Los nombres de los métodos se expresan como acciones o verbos en infinitivo. Su codificación empieza con letra minúscula. Se utilizan nombres en español

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

- Todas las variables incluyendo los parámetros cuentan con un prefijo de una letra que indica el tipo de datos. Adicionalmente, los parámetros llevan una p como prefijo para identificarlos como parámetros de métodos.
- Los arreglos llevan un prefijo adicional al de tipo de dato para identificar que se trata de un conjunto de valores y uno simple.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Descripción del Clases del Sistema

Clase ENTERO

Packages utilizados

java.io
java.text

Atributos de Clase

NDIGITOS

static final int **nDigitos**

Cantidad de dígitos por celda, es leído de rsa.ini

---

nMaxElem

static final int **nMaxElem**

Cantidad de celdas, calculado en base a la cantidad de dígitos de primos.
Se inicializa con el siguiente valor:

(int) Math.ceil((double) RSA.nTotalDigPrimo/nDigitos*45./10.)

---

nOrden

static final double **nOrden**

Valor de una unidad de la siguiente celda. Toma el siguiente valor inicial:

Mah.pow(10,nDigitos);

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Atributos de Instancia

nUltElem

int **nUltElem**

Indice del último elemento utilizado. Toma 0 como valor inicial

---

nArrValor

double **nArrValor** []

Número Entero almacenado. Toma 0 como valor inicial y se crean instancias con nMAXELEM elementos

---

Método Constructor

Entero

public void **Entero** (String pcValor)

El constructor de la clase Entero permite inicializar la estructura con el valor recibido como parámetro.

Parámetros

pcValor   Cadena con el valor a asignar a la estructura de tipo Entero, puede contener espacios en blanco.

Retorno

Ninguno

Algoritmo

Invoca al método asignarValor(String) con el parámetro recibido

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Métodos

hallarAcarreo

double **hallarAcarreo**(double pnValor)

Devuelve la porción numérica del valor recibido como parámetro que excede el valor máximo de una celda. Se utiliza en las operaciones aritméticas.

Parámetros

pnValor Número doble al cual se halla el acarreo

Retorno

Retorna el valor en exceso

Algoritmo

Retornar el resultado de truncar (pnValor / 10^nDigitos)

---

devolverCadena

String **devolverCadena()**

Permite obtener el valor del número actual en forma de cadena

Parámetros

No tiene

Retorno

Retorna una cadena conteniendo el valor numérico almacenado como Entero

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Algoritmo

Inicializar Cadena en nulo
Repetir desde último hasta primer elemento de estructura Entero
Si es el último elemento,
Cadena += elemento formateado con un carácter como mínimo
Sino,
Cadena += elemento formateado con máximo número de dígitos por celda

---

asignarValor

void **asignarValor** (String pcValor)
Permite asignar valor al número Entero

Parámetros

pcValor Cadena conteniendo un valor numérico que será asignado al objeto

Retorno

Ninguno

Algoritmo

Inicializar estructura de objeto Entero en 0
i=0
Desde el último hasta el primer carácter
Saltar todos los espacios en blanco
Cadena += nDigitos caracteres siguientes, sin considerar espacios en blanco
Asignar Cadena a elemento i++ de estructura Entero
Ultimo Elemento = --i

---

asignarValor

void **asignarValor** (Entero peValor)
Permite asignar un valor al objeto actual copiando el valor de otro objeto Entero

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Parámetros

peValor — Contiene el valor del otro objeto Entero cuyo valor será copiado al actual

Retorno

Devuelve el valor del objeto copiado

Algoritmo

Para cada elemento del Entero recibido como parámetro
    Asignar valor correspondiente a objeto actual
Ultimo Elemento = Ultimo Elemento de parámetro

---

multiplicar

Entero **multiplicar** (Entero peFactor)

Permite multiplicar el número entero recibido como parámetro con el valor almacenado en el objeto actual.

Parámetros

PeFactor — Número con el cual se multiplicara el objeto actual

Retorno

Retorna el resultado de multiplicar ambos números

Algoritmo

---

sumar

public Entero **sumar**(Entero peSumando)

Permite sumar el número entero recibido como parámetro al valor almacenado en el objeto actual.

Parámetros

peSumando — Número con el cual se sumara el objeto actual

Retorno

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Retorna el valor de la suma de ambos números

Algoritmo

Inicializar acarreo en 0
Para cada elemento de Entero
    Sumar elementos correspondientes + acarreo
    Si valor>máximo posible
        Calcular acarreo
    Sino
        acarreo=0

RESTAR

Entero **restar** (Entero peSustraendo)

Permite restar el número entero recibido como parámetro del valor almacenado en el objeto actual.

Parámetros

peSustraendo   Número que se restará al objeto actual

Retorno

Devuelve resta de los dos números

Algoritmo

Inicializar acarreo en 0
Para cada elemento de Entero
    Si acarreo=1
        Restar 1 a elemento de objeto actual
        acarreo = 0
    Restar elemento objeto actual menos elemento parámetro
    Si resultado < 0
        Asignar complemento
        acarreo=1

---

dividir

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

double **dividir**(double pnDivisor)

Esta función devuelve el resto de dividir el valor almacenado en el objeto actual entre el número recibido como parámetro.

Parámetros

pnDivisor Número entre el cual se dividirá el objeto Entero

Retorno

Devuelve el resto de la división, adicionalmente actualizará el contenido del objeto actual con el cociente correspondiente.

Algoritmo

Para cada elemento empezando en el mas significativo
Asignar al elemento actual la división entera de el mismo entre pnDivisor
Sumar resto a elemento de posición anterior

---

dividir

Entero **dividir**(Entero peModulo,Entero peResto)

Permite dividir el valor del objeto actual entre el valor otro objeto Entero. Devuelve el cociente y el residuo.

Parámetros

peModulo Divisor entre el que se divide el valor actual
peResto Este es un parámetro de salida en el cual se retorna el resto de la división

Retorno

Cociente de la división, además el parámetro peResto contendrá el resto de la división

Algoritmo

---

comparar

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

int **comparar** (Entero peValor)

Permite comparar los valores almacenados en el objeto actual y del objeto recibido como parámetro.

Parámetros

peValor Valor a ser comparado con el Número actual

Retorno

Devuelve 0 cuando ambos son iguales, 1 cuando el valor del objeto actual es mayor y 2 cuando el valor del parámetro es mayor.

Algoritmo

Para cada elemento de Entero
    Si elemento actual > elemento parámetro, retorna 1
    Sino Si elemento actual < elemento parámetro, retorna 2
Retorna 0, son iguales

---

potenciarModulo

Entero **potenciarModulo** (Entero peA,Entero peB,Entero peN)

Permite calcular la potencia b en modulo n de a, o potencia modular: a^b mod n. Se aplica el algoritmo Square & Multiply (Elevar al cuadrado y multiplicar) que luego de convertir el exponente a binario, eleva a al cuadrado en modulo n y por cada digito 1 del exponente multiplica en modulo n el numero anterior por el original

Parámetros

peA Base, número al cual se desea elevar
peB Exponente de la potencia modular
peN Módulo para el cual se hallará la potencia

Retorno

Retorna el valor de la potencia b de a en modulo n

Algoritmo

Convertir exponente peB a número binario b
c = peA
Para cada dígito de b, excluyendo más significativo
    c=c^2 modulo peN
    Si dígito de b=1, c=c*peA modulo peN
Fin-Para

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Devolver c

INVERTIR

Entero **invertir** (Entero peZ)

Permite calcular la inversa multiplicativa del numero actual en modulo n. Se conoce como inversa al numero que multiplicado por el correspondiente valor actual, dara la unidad en modulo n como resultado (u*m=1 mod n) Se aplica el Algoritmo de Euclides para hallar el Maximo Comun Divisor de m,n pues se cumple que
Si r=MCD(x,y) ==>> existe u,v tales que r=u*x+v*y

Para m primo relativo de n se cumple que
MCD(m,n)=1 ==>> 1=u*m+v*n ==>> u*m = 1 mod n

Parámetros

peZ Módulo para el cual se hallara la inversa del valor almacenado en el objeto actual

Retorno

Inverso multiplicativo del valor actual en estructura de Entero

Algoritmo

Devolver retorno de calcularEuclides

---

hallarMCD

Entero **hallarMCD** (Entero peY)

Retorna el valor del MCD Maximo Comun Divisor de dos números. Aplica el Algoritmo de Euclides

Parámetros

peY Numero para el cual, junto con el valor del objeto actual, se hallará el Maximo Comun Divisor

Retorno

Devuelve el Maximo Comun Divisor del numero del parámetro y del valor actual

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Algoritmo

Devolver resto de calcularEuclides

---

corregirValor

Entero **corregirValor** (Entero peValor)

Devuelve el valor complementario en caso el numero almacenado en la estructura Entero sea negativo

Parámetros

peValor	Numero al cual se verifica su valor

Retorno

Valor complementario del valor recibido

Algoritmo

Si último elemento=Valor Máximo por celda (solo 9s, señal de número negativo)

Retorna suma de orden siguiente y valor actual.

---

calcularEuclides

Entero **calcularEuclides** (Entero peNum1,Entero peNum2, Entero peResto)

Desarrolla el Algoritmo Extendido de Euclides que permite el cálculo del Maximo Comun Divisor y de la inversa modular

Parámetros

peNum1	Primer factor
peNum2	Segundo factor
peResto	Parámetro de salida que retorna la Inversa Modular de peNum1 modulo peNum2

Retorno

Maximo Comun Divisor de peNum1 y peNum2, además retorna la inversa modular en el parámetro de salida peResto

Algoritmo

Idea Básica: Reemplazar los números originales con números más pequeños que tienen el mismo MCD hasta que uno de ellos es cero.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

El MCD de dos números puede ser expresado como la suma de algún múltiplo de cada número, es decir, si r=MCD(x,y), existen dos enteros u y v tales que r = ux + vy

```
Inicializar u1,v2 en 1
Inicializar u2,v1 en 0

Mientras x!=0 && y!= 0
        Hallar cociente q y resto r de dividir x / y
        u=u2-q*u1
        v=v2-q*v1
        x=y
        y=r
        u2=u1
        u1=u
        v1=v
Fin-Mientras
Si U2<0, tomar complemento
Asignar resto anterior a parámetro de salida, Inversa Modular
Retornar U2, Máximo Común Divisor
```

---

convertirBinario

String **convertirBinario** (Entero peDecimal)

Convierte el valor del parámetro recibido en su equivalente binario. Devuelve una cadena de 0s y 1s que representan dicho valor binario. El parámetro debe estar expresado en el sistema decimal.

Parámetros

peDecimal   Número Entero que será convertido a binario

Retorno

Cadena con el numero binario correspondiente con el digito mas significativo en la posicion 0 del String

Algoritmo

```
Inicializar Cadena en nulo
Mientras valor sea mayor a 1
        Hallar cociente q y resto r de dividir peDecimal / 2
        Concatenar resto r a Cadena por la izquierda
Concatenar ultimo q a Cadena por la izquierda
Devolver Cadena
```


SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

---

formatear

String **formatear** (double pnValor,int pnDigitos)

Permite dar formato a un numero, es decir, rellenar el número recibido con ceros a la izquierda hasta lograr la cantidad de digitos indicada

Parámetros

pnValor	Numero al cual se aplica el formato
pnDigitos	Cantidad de caracteres del número con formato resultante

Retorno

Cadena conteniendo el numero recibido en el formato indicado

Algoritmo

Conformar Cadena con pnDigitos 0s
Devolver valor con formato aplicado

---

escribir

void **escribir** (FileWriter pfFile,char pcIndAvance,char pcIndSeparar,
int pnCarBloque, char pcIndRelleno)

Permite grabar el valor del objeto actual en un archivo, este archivo debe encontrarse abierto

Parámetros

pfFile	Identificador de archivo que debe encontrarse abierto
pcIndAvance	El valor S indicará que antes de imprimir el valor recibido, se imprimirá un cambio de línea.
pcIndSeparar	El valor S indicará que se imprima un espacio en blanco después de escribir el valor recibido.
pnCarBloque	Contiene la cantidad de dígitos que se imprimirán por celda. Según el siguiente parámetro se rellenará o no con ceros a la izquierda
pcIndRelleno	El valor S indica que el número a imprimir se rellenará con ceros a la izquierda hasta alcanzar pnCarBloque digitos

Retorno

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Ninguno

Algoritmo

Escribir valor de Entero recibido en archivo según parámetros recibidos

---

LEER

void **leer** (Archivo paFile)

Permite leer un número de un archivo y almacenarlo en la estructura de Entero

Parámetros

paFile Archivo del cual se leerá el numero que se almacenará en la estructura actual

Retorno

Ninguno

Algoritmo

Obtener siguiente token de archivo de entrada (ya debe estar abierto)
Mientras no sea EOF o EOL
Asignar valores a estructura Entero

Clase RSA

Packages utilizados

java.io
java.lang.Math
java.text
java.util.Random

Atributos de Clase

nTotalDigPrimo

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

static int **nTotalDigPrimo**

Total de Dígitos de los números primo a ser generados, este valor es leído de rsa.ini.

---

eCERO

static final Entero **eCERO**

Valor Constante 0

---

eUNO

static final Entero **eUNO**

Valor Constante 1

---

eDOS

static final Entero **eDOS**
Valor Constante 2

---

eDIEZ

static final Entero **eDIEZ**

Valor Constante 10

---

eDIEZ

static final Entero **eDIEZ**

Valor Constante 10

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

NARRMATRIZPRIMOS

static final int **nArrMatrizPrimos**

2000 primeros números primos

Atributos de Instancia

eP

Entero **eP**

Número Primo p utilizado para la generación de claves RSA

---

eQ

Entero **eQ**

Número Primo q utilizado para la generación de claves RSA

---

eN

Entero **eN**

Módulo de la clave RSA n=p*q

---

eZ

Entero **eZ**

Función de Euler z=(*p*-1)*(*q*-1)

---

eD

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Entero **eD**

Clave Privada d utilizada para decriptar archivos

---

eE

Entero **eE**

Clave Pública e utilizada para encriptar, es leída de archivo rsa.ini

---

Método Constructor

Utiliza el método constructor por Defecto

Métodos

Entero

public **Entero** PG()

Este método permite la generación aleatoria de un número primo, la cantidad de dígitos (también generados aleatoriamente) con la que se creará este número es especificada rsa.ini. Se aplica el algoritmo de Miller-Rabin para la verificación de primalidad del numero generado. Se recomienda utilizar 128 o 256 dígitos para asegurar la inviolabilidad de la clave, sin embargo es posible utilizar valores menores para pruebas o con efectos didácticos.

Parámetros

No tiene

Retorno

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Numero primo generado en estructura de Entero

Algoritmo

El algoritmo utiliza la prueba probabilística de primalidad de Miller-Rabin, no verifica que los primos sean "fuertes"

```
Inicializar Primo en 0
Mientras Primo no supera prueba Miller-Rabin
    For i=0 hasta nTotalDigPrimo-1
        Generar un numero aleatorio para posición
        Generar un numero aleatorio para valor
        Extraer digito de valor de posición
        Si es primer digito no puede ser 0
        Agregar a Primo
Retornar Primo
```

---

verificarPrimalidad

boolean **verificarPrimalidad**(Entero peN)

Este método aplica el algoritmo de Miller-Rabin para verificación de primalidad de un número supuestamente primo. Se utiliza el parámetro t leido de rsa.ini para indicar la cantidad de aplicaciones de la prueba.

Parámetros

peN   Numero n supuestamente primo a ser verificado

Retorno

Devuelve verdadero si el número supera el test de primalidad y falso en otro caso

Algoritmo

```
Hallar r tal que: n - 1 = 2^s * r tal que r es impar, siendo n primo impar, n-1
es par
Repetir de  1 a t  (factor de repetición)
Escoger un entero aleatorio que satisface 1 < a < n - 1
Calcular y = a^r mod n
Si (y <> 1) && (y <> n-1) entonces
    j := 1
    Mientras (j < s) && (y <> n - 1) entonces
```


SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

```
                y := y2 mod n
                Si y = 1 entonces devolver FALSO
                j := j + 1
        FinMientras
        Si (y &lt;&gt; n - 1) entonces devolver FALSO
FinSi
Devolver VERDADERO
```

RSAKG

public void **RSAKG**(Entero peP, Entero peQ)

Este método permite manejar dos posibles tipos de generación de claves. La primera permite recibir números aleatorios primos p y q en base a los cuales se verifica que la clave publica e sea primo relativo con cada uno de ellos para de no ser así, generar un nuevo e. La otra forma permite generar aleatoriamente los números primos hasta que sean relativamente primos con e. En ambos casos, estos valores permiten efectuar el calculo de la clave privada d

Parámetros

peP Número primo aleatorio p, si es 0 se generara aleatoriamente
peQ Número primo aleatorio q, si es 0 se generara aleatoriamente

Retorno

No tiene

Algoritmo

```
Mientras MCD de z=(p-1)*(q-1) y e no sea 1
        Si peP!=0 && peQ!=0
                Asignar parámetros a p y q
                Verificar que sean relativamente primos
                Si e no cumple, sumarle 2 para siguiente vez
        Sino
                Mientras MCD (p,q)!=1
                        Generar p
                        Generar q
        z = (p-1) * (q-1)
        Calcular Clave Privada d como inversa de e modulo z
n=p*q
```

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

RSAEP

public Entero **RSAEP** (Entero peN, Entero peE, Entero peM)

Primitiva para la encriptación de un mensaje. Recibe como parámetros el número n, la clave pública e y el número m o mensaje a encriptar.

Parámetros

peN Número n
peE Clave Pública e
peM Mensaje a encriptar, numero entre 0 y n-1

Retorno
Mensaje encriptado, numero entre 0 y n-1

Algoritmo
Verificar que mensaje a encriptar sea mayor a 0 y menor a n
Encriptar mensaje m recibido y obtener c aplicando la formula
c= m^e mod eN

RSADP

public Entero **RSADP**(Entero peN, Entero peD, Entero peC)
Primitiva para la decriptación de un número mensaje encriptado con el procedimiento anterior. Recibe como parámetros el número n, la clave privada d y el número c o mensaje a decriptar.

Parámetros
peN Numero n
peD Clave privada d
peC Mensaje a decriptar, numero entre 0 y n-1

Retorno
Mensaje decriptado, numero entre 0 y n-1

Algoritmo
Verificar que mensaje a decriptar sea mayor a 0 y menor a n
Encriptar mensaje m recibido obteniendo c aplicando la formula m = c^d mod n

---

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

GUARDARCLAVE

void **guardarClave** (Entero peElem1,Entero peElem2, String cNombreArchivo)

Permite almacenar en el archivo del nombre indicado los Enteros recibidos. Cada numero se guarda en una línea separando cada elemento del arreglo con un espacio en blanco.

Parámetros

peElem1	Primer Entero a almacenar en archivo
peElem2	Segundo Entero a almacenar en archivo
cNombreArchivo	cNombreArchivo Nombre a asignar al archivo de claves

Retorno

No tiene

Algoritmo

Crear archivo de claves con nombre proporcionado
Escribir las claves
Cerrar Archivo

GENERARCLAVES

void generarClaves (Entero peP,Entero peQ,String pcNomClavePublica,
String pcNomClavePrivada)

Permite generar las claves RSA privada y publica y almacenarlas en un archivo, invoca la primitiva de generación de claves.

Parámetros

peP	Primo p a utilizar en la generacion de claves, si se envia 0 sera generado aleatoriamente
peQ	Primo q a utilizar en la generacion de claves, si se envia 0 sera generado aleatoriamente
pcNomClavePublica	Nombre del archivo de clave publica a crear, o dejar nulo para no crearlo
pcNomClavePrivada	Nombre del archivo de clave privada a crear, o dejar nulo para no crearlo

Retorno

No tiene

Algoritmo

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Generar claves invocando RSAKG
Si pcNomClavePublica!=""
guardarClave (e,n, pcNomClavePublica)
Si pcNomClavePrivada!=""
guardarClave (d,n, pcNomClavePrivada)

GENERARCLAVESDEMO

void **generarClavesDemo** (String pcPrimop, String pcPrimoq,
String pcNomClavePublicaDemo,
String pcNomClavePrivadaDemo)

Permite asignar los números primos p y q leídos de pantalla para la generacion de claves RSA

Parámetros

pcPrimop	Primo p asignado para la creación de claves
pcPrimoq	Primo q asignado para la creacion de claves
pcNomClavePublicaDemo	Nombre del archivo de clave publica leido de pantalla, puede ser nulo
pcNomClavePrivadaDemo	Nombre del archivo de clave privada leido de pantalla, puede ser nulo

Retorno

No tiene

Algoritmo

Asignar valores leídos de pantalla a valores p y q
Generar Claves RSA en modo demo

ENCRIPTARARCHIVO

void **encriptarArchivo** (String pcArchivoEncriptar,String pcArchivoClavePublica)

Método que permite encriptar el archivo de texto recibido como parámetro aplicando la clave pública recibida como segundo parámetro.

Parámetros

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

pcArchivoEncriptar	Nombre del archivo a encriptar
pcArchivoClavePublica	Nombre del archivo de clave publica

Retorno

No tiene

Algoritmo

Abrir archivo de clave publica
Leer componentes de clave publica
Cerrar archivo de clave publica
Calcular numero de caracteres de bloque a ser encriptados por vez
Encriptar el archivo

ENCRIPTAR

void **encriptar** (String pcArchivoEncriptar, int pnCarBloque,Entero peE, Entero peN)

Recibe como parámetro el archivo a encriptar en formato numérico decimal. Este archivo será leído formando bloques de pnCaracteres y cada bloque encriptado aplicando las claves e y n a la correspondiente primitiva. Se crea otro archivo con el contenido encriptado del primero.

Parámetros

pcArchivoEncriptar	Nombre de Archivo a encriptar
pnCarBloque	Cantidad de caracteres por bloque a encriptar
peE	Componente e de Clave Publica
peN	Componente n de Clave Publica

Retorno

No tiene

Algoritmo

Convertir archivo a encriptar a numérico
Abrir archivo numérico
Conformar nombre de archivo encriptado de salida, se usa extensiones leídas de rsa.ini
Abrir archivo numérico
Mientras no sea EOF
    Leer siguiente entero
    Encriptar bloque
    Escribir bloque encriptado en archivo de salida, un bloque

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

por linea
Cerrar archivos
Abrir archivo encriptado en Bloc de Notas

---

decriptarArchivo

void **decriptarArchivo**(String pcArchivoDesencriptar,
String pcArchivoClavePrivada)

Método que permite decriptar el archivo encriptado recibido como parámetro aplicando la clave privada recibida como segundo parámetro.

Parámetros

pcArchivoDesencriptar	Nombre del archivo a decriptar
pcArchivoClavePrivada	Nombre del archivo de clave privada

Retorno

No tiene

Algoritmo

Abrir archivo de clave privada
Leer cada componente de la clave privada
Cerrar el archivo
Calcular numero de caracteres de bloque que seran desencriptados por vez
Decriptar el archivo

---

calcularTamano

int **calcularTamano**(Entero peN)

Función que determina la cantidad de dígitos que conformarán el bloque a encriptar o desencriptar, tal que 10^(nBloque-1) < n < 10^nBloque

Parámetros

peN	Número n

Retorno

Tamaño del bloque

Algoritmo

nAux = 1
Bloque=0

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

```
Hacer
        Bloque++
        nAux=nAux*10
Mientras nAux<peN
Retornar Bloque--
```

DECRIPTAR

void **decriptar**(String pcArchivoDesencriptar,int pnCarBloque, Entero peD, Entero peN)

Recibe como parámetro el archivo a decriptar en formato encriptado numérico decimal. Este archivo será leído línea por línea y desencriptado aplicando las claves d y n a la correspondiente primitiva. Se obtiene un bloque de pnCaracteres por cada línea decriptada. Se crea otro archivo con el contenido desencriptado del primero.

Parámetros

pcArchivoDesencriptar	Nombre de Archivo a decriptar
pnCarBloque	Cantidad de caracteres de cada bloque desencriptado
peD	Componente e de Clave Privada
peN	Componente n de Clave Privada

Retorno

No tiene

Algoritmo

```
Conformar nombre de archivo decriptado de salida, se usa extensiones leídas de rsa.ini
Abrir archivo de salida numérico con nombre leído de rsa.ini
Abrir archivo numérico
Mientras no sea EOF
        Leer bloque encriptado
        Decriptar bloque leído
        Escribir bloque desencriptado en pnCarBloque caracteres
                por vez, se rellena con 0s de ser necesario
```

Clase Archivo

Packages utilizados

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

java.io
java.text
java.util

Atributos de Clase

No tiene

Atributos de Instancia

EOF

int **EOF**

Señal de Fin de Archivo, se inicializa en 0.

FR

FileReader **fR**

Puntero a archivo manejado por esta clase

---

sT

StreamTokenizer **sT**

Contenido del archivo actual tokenizado

Método Constructor

Archivo

public **Archivo** (String pcNomArchivo)

Método constructor de la clase Archivo. Permite inicializar el puntero al

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

archivo e inicializar la variable de control de fin de archivo EOF.

Parámetros

pcNomArchivo Nombre de archivo de lectura a abrir

Retorno

No tiene

Algoritmo

EOF=0
Inicializa fR

Métodos

CERRAR

void **cerrar** ()

Permite cerrar el archivo actual

Parámetros

No tiene

Retorno

No tiene

Algoritmo

Cierra archivo

INICIARTOKEN

void **iniciarToken** ()

Permite leer en forma de tokens el archivo actual, inicializa el atributo de instancia sT

Parámetros

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

No tiene

Retorno

No tiene

Algoritmo

Inicializa tokens

OBTENERSIGUIENTETOKEN

String **obtenerSiguienteToken** (char pcIndEOL)

Permite obtener el siguiente token del archivo actual

Parámetros

pcIndEOL 'S' indica que se considerara el cambio de línea como token, otro valor que no

Retorno

Devuelve el token leído en forma de char, los números son convertidos a entero

Algoritmo

Obtener siguiente token
Retornarlo de tipo String

LEERLINEA

int **leerLinea** (StringBuffer pcCadena) throws Exception

Lee una línea del archivo de entrada hasta encontrar un cambio de línea. Devuelve la línea leída en el parámetro cadena.

Parámetros

PcCadena Devolverá la cadena leída sin incluir el cambio de línea

Retorno

Devuelve el valor de EOF

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Algoritmo

Lee una línea del archivo actual
Retorna valor del ultimo carácter leído

LEERENTERO

int **leerEntero** (Entero peNum)

Lee una línea del archivo de entrada y asigna el valor leído a la estructura pcBloque. Se asume que el contenido de esa línea es numérico.

Parámetros

PeNum Número que almacenara el valor leido del archivo actual

Retorno

Devuelve el valor de EOF

Algoritmo

Leer línea actual
Asignar valor a Entero
Devolver valor de EOF

---

convertirDecimal

void **convertirDecimal**(String pcNomArchivoDecimal,int pnCarBloque)

Crea el archivo pcNomArchivoDecimal con el equivalente en código ASCII de cada carácter del archivo de entrada. Los números se escriben de manera consecutiva considerando pnCarBloque dígitos por línea

Parámetros

pcNomArchivoDecimal Nombre que recibirá el archivo numérico
pnCarBloque Caracteres a ser escritos en cada línea

Retorno

No tiene

Algoritmo

Abrir archivo

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

```
Inicializar Cadena en nulo
Mientras no sea EOF
        Leer un caracter
        Si longitud Cadena+3<pnCarBloque
                Cadena += ASCII de carácter
        Sino
                Cadena += Subcadena de ASCII hasta completar
                                longitud pnCarBloque
                Escribir Cadena en archivo
                Cadena = Subcadena de ASCII sobrante
Escribir Cadena
Completar pnCarBloque con 0s
Cerrar Archivo
```

---

convertirCaracter

void **convertirCaracter**(String pcNomArchivoCaracter

Crea el archivo pcNomArchivoCaracter con el carácter ASCII equivalente de cada código leído del archivo de entrada. Se asume que cada código ocupa 3 dígitos.

Parámetros

pcNomArchivoCaracter   Nombre del archivo de caracteres resultante

Retorno

No tiene

Algoritmo

```
Abrir archivo
Mientras no sea EOF
        Leer 3 caracteres
        Escribir en archivo el equivalente ASCII
Cerrar archivo
```

ABRIRBLOC

void **abrirBloc** (String pcNomArchivo)

Abre el archivo indicado en el bloc de Notas

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Parámetros

pcNomArchivo Nombre del archivo a abrir

Retorno

No tiene

Algoritmo

Invocar Bloc de Notas

Clase Form1

Packages utilizados

com.ms.wfc.app
com.ms.wfc.core
com.ms.wfc.ui
com.ms.wfc.html
java.awt.FileDialog
java.awt.Dialog
java.awt.Frame

Atributos de Clase

NDIGITOSTMP

static int **nDigitosTmp**

Digitos por celda leído de archivo rsa.ini

---

nTotalDigPrimoTmp

static int **nTotalDigPrimoTmp**;

Cantidad de digitos de numeros primos leído de archivo rsa.ini

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

---

cETmp

static String **cETmp**

Valor por defecto de componente e de clave leído de archivo rsa.ini

---

nTTmp

static int **nTTmp**

Cantidad de repeticiones de algoritmo de Miller Rabin leído de archivo rsa.ini

---

cExtensionTextoTmp

static String **cExtensionTextoTmp**

Extensión de archivos de texto por defecto leído de archivo rsa.ini

---

cExtensionEncripTmp

static String **cExtensionEncripTmp**

Extensión de archivos encriptados por defecto leído de archivo rsa.ini

---

cExtensionDecripTmp

static String **cExtensionDecripTmp**

Extensión de archivos decriptados por defecto a ser leido de rsa.ini

---

cExtensionClaveEncripTmp

static String **cExtensionClaveEncripTmp**

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Extensión de archivos de clave publica por defecto leído de archivo rsa.ini

---

cExtensionClaveDecripTmp

static String **cExtensionClaveDecripTmp**

Extensión de archivos de clave privada por defecto leído de archivo rsa.ini

---

CNOMBREENCRIPTMP

static String **cNombreEncripTmp**

Nombre de archivo numérico temporal de encriptación leído de archivo rsa.ini

---

cNombreDecripTmp

static String **cNombreDecripTmp**

Nombre de archivo numérico temporal de decriptacion leído de archivo rsa.ini

---

Atributos de Instancia

No tiene

Método Constructor

Form1

public **Form1**()

Presenta la ventana principal del sistema

Parámetros

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

No tiene

Retorno

No tiene

Algoritmo

Inicializar forma
Leer Parámetros de rsa.ini

Métodos

LEERPARAMETROS

void **leerParametros**()

Permite leer variables desde rsa.ini y asignarlas a los atributos correspondientes

Parámetros

No tiene

Retorno

No tiene

Algoritmo

```
Abrir Archivo rsa.ini
Inicializar Tokens
Obtener siguiente token
Mientras no es EOF
        Si operador '='
                Asignar Valor de Variables a atributos temporales
        Obtener siguiente token
Cerrar archivo
```

---

button2_click

private void **button2_click**(Object source, Event e)

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Es invocada desde el botón Mostrar Archivo Clave Publica, permite visualizar el contenido del archivo indicado en el campo Archivo Clave Publica

Parámetros

source, e	Parámetros automaticamente proporcionados por la herramienta

Retorno

Indicador de status de término

Algoritmo

Invoca bloc de notas

ABRIRARCHIVO

String **abrirArchivo** (String pcNombre,String pcFiltro,String pcExt)

Permite seleccionar un archivo del árbol de directorios

Parámetros

pcNombre	Nombre inicial a ubicar en árbol de directorios
pcFiltro	Filtro por defecto
pcExt	Extensión por defecto

Retorno

Retorna el nombre del archivo elegido por el usuario o valor del parámetro recibido si se canceló la acción

Algoritmo

Asignar filtros y opciones de clase OpenDialog
Mostrar ventana de diálogo
Si resultado=OK
    Devuelve nombre de archivo seleccionado
Sino
    Devuelve parámetro recibido

---

btEjecutar_click

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

private void **btEjecutar_click**(Object source, Event e)
Es invocada desde el botón Ejecutar de la pantalla principal, permite ejecutar las diferentes rutinas que componen el Sistema de Encriptacion RSA

Parámetros

source, e   Valores automaticamente proporcionados por la herramienta

Retorno
Indicador de status

Algoritmo
Obtiene nombre del tab activo
Según el tab elegido, invoca rutina correspondiente

---

button17_click

private void **button17_click**(Object source, Event e)
Es invocada desde el boton Seleccionar Archivo Encriptar, permite seleccionar un archivo del arbol de directorios ubicando por defecto el archivo indicado en el campo Archivo a Encriptar.

Parámetros

source, e   Valores automaticamente proporcionados por la herramienta

Retorno
Indicador de status

Algoritmo
Invocar abrirArchivo

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

ANEXOS

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Sistema de Encriptación RSA

Manual de Usuario

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

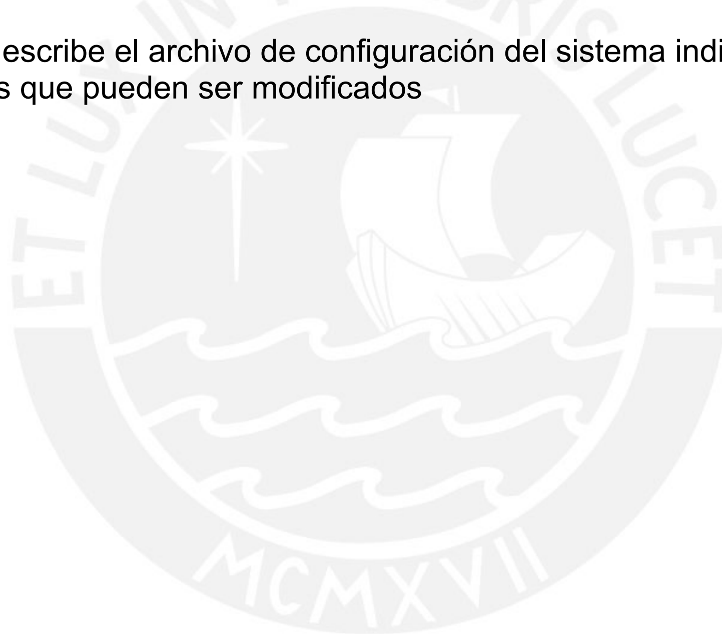
PRESENTACIÓN

Este documento muestra la forma de operar el Sistema de Encriptación RSA desarrollado como tesis para obtener el grado de Magíster en Informática con el título “Encriptación de Archivos de Texto bajo algoritmo RSA” para la Pontificia Universidad Católica del Perú

En la primera parte se presentan las generalidades del sistema en la que se describe aspectos comunes en la operación de todas las opciones e instalación del sistema.

A continuación se describe la funcionalidad de cada una de las opciones que compone el sistema entre las que se cuentan, Encriptar, Desencriptar, Generar Claves y Generar Claves Demo.

Por último, se describe el archivo de configuración del sistema indicando los diversos valores que pueden ser modificados



SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Sistema de Encriptación RSA

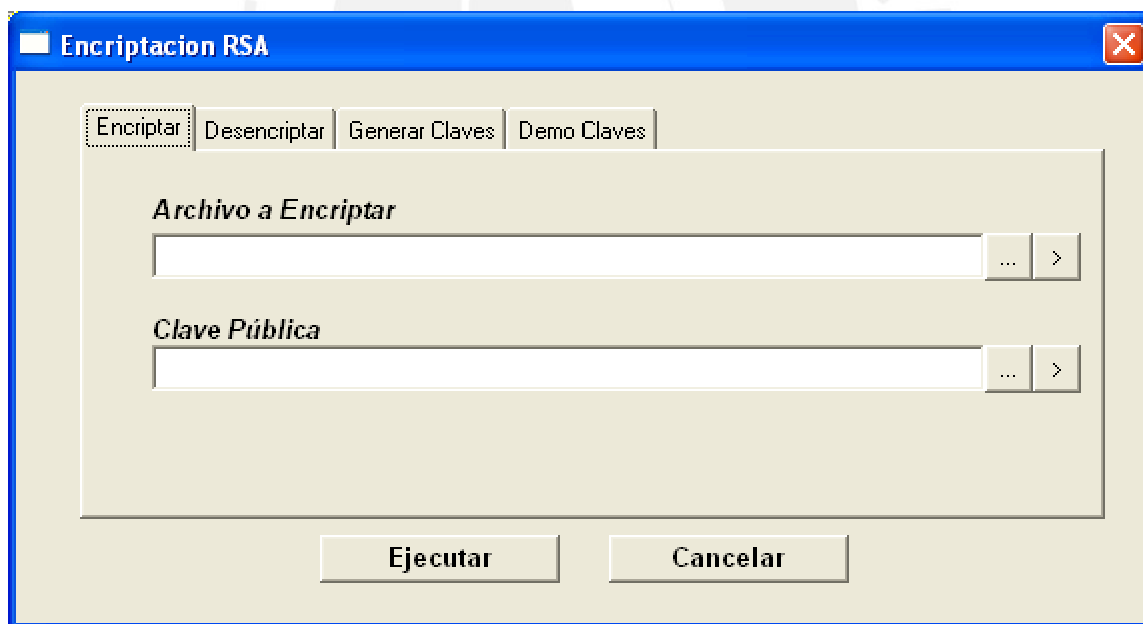
Este sistema permite encriptar archivos de texto, desencriptar archivos previamente codificados y generar claves públicas y privadas bajo el algoritmo de encriptación RSA. Para la utilización de cada una de ellas se requiere el ingreso de algunos pocos valores para la selección de archivos y para el inicio de la acción.

Generalidades

El sistema cuenta con 4 posibles cursos de acción:

Encriptar
Desencriptar
Generar Claves
Demo Claves

Como puede verse en esta imagen, cada acción esta relacionada a una pestaña en las cuales aparece el nombre de la acción a efectuar.

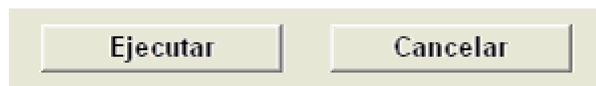


Para acceder a cualquiera de estas opciones bastará hacer clic sobre la pestaña correspondiente.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

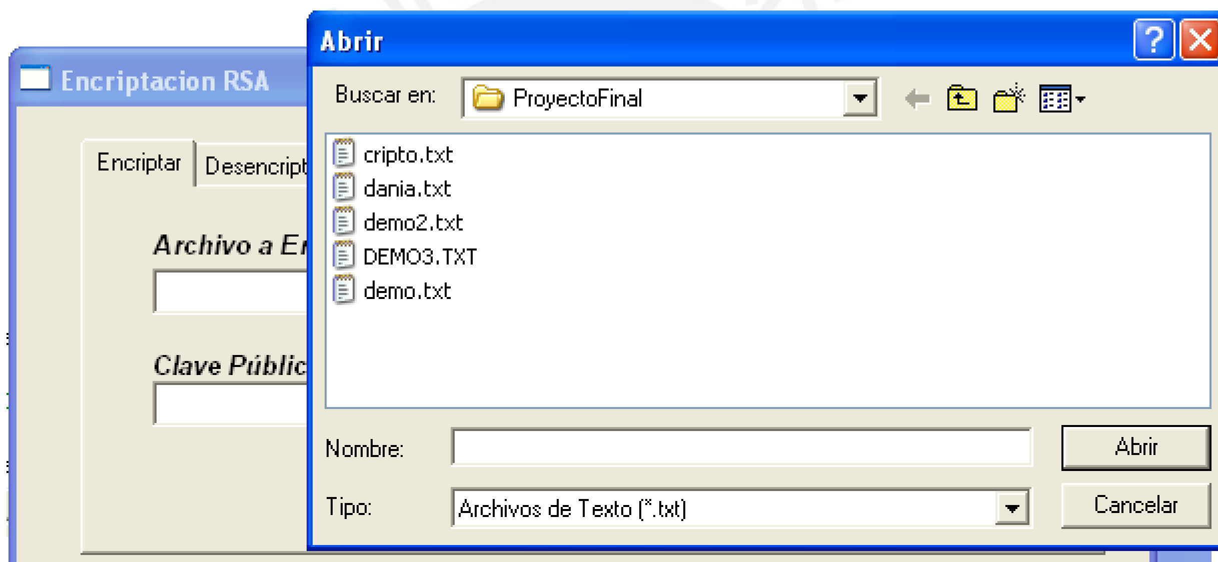
Los botones Ejecutar y Cancelar de la parte inferior aparecerán permanentemente sin importar que pestaña se seleccione:



El botón **Ejecutar** permite dar inicio al proceso seleccionado mientras que el botón **Cancelar** permite terminar con la ejecución del programa y cerrar la ventana.

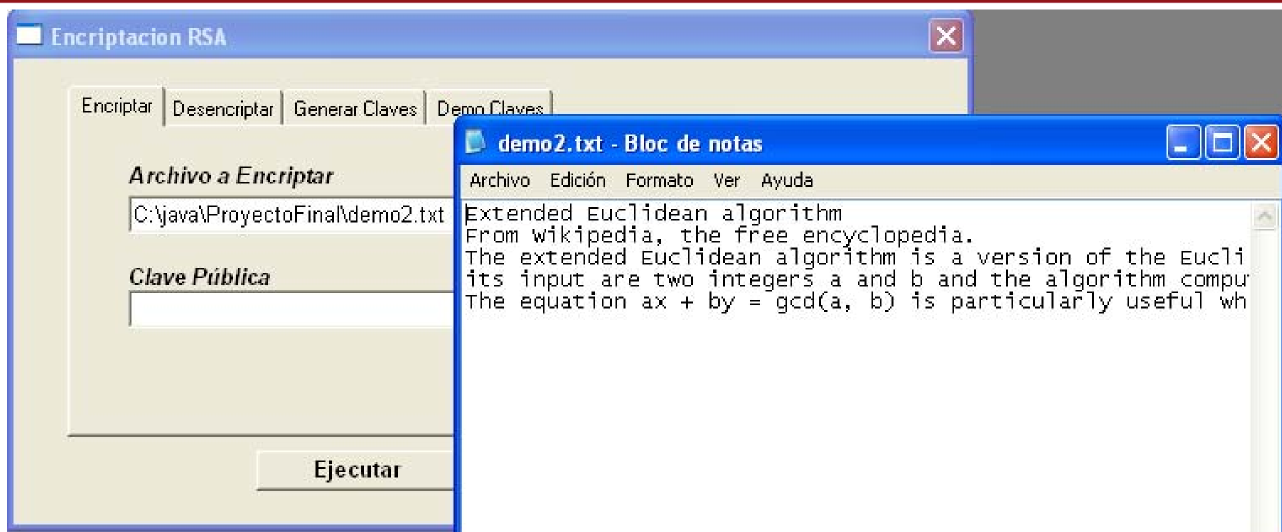
Asimismo, en todas las ventanas se dispondrá de los siguientes botones que permitirán seleccionar archivos del árbol de directorios y visualizarlo.

Permite seleccionar un archivo del disco. Presenta la ventana estándar de Windows para este fin.



Permite visualizar el contenido del archivo en el bloc de notas como se muestra a continuación

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis



Requerimientos del Sistema

Este sistema puede ser ejecutado bajo cualquier entorno Windows.

Instalación

El sistema consta de las siguientes partes:

RSA.exe	Aplicación
RSA.ini	Archivo de Configuración

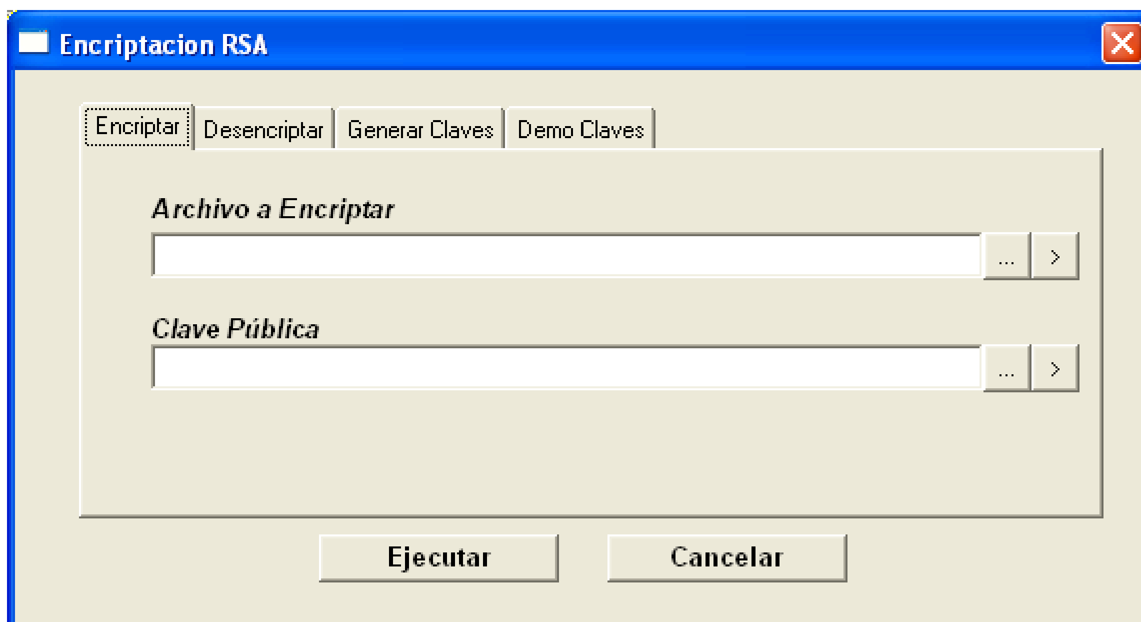
Ambos deben ser copiados del CD de instalación en el mismo directorio

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Opciones del Sistema

1. Encriptar

Al seleccionar la pestaña Encriptar, se visualizará esta ventana. Permite el ingreso de datos para encriptar archivos de texto con aplicación del algoritmo de encriptación RSA. Se solicitan los siguientes datos



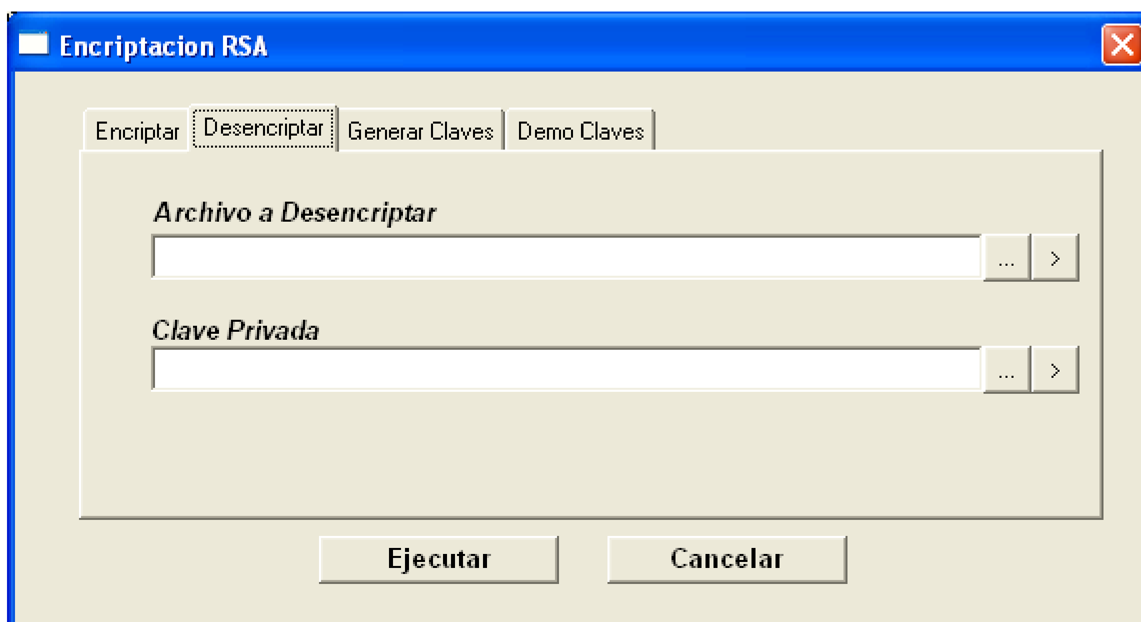
Archivo a Encriptar:
Permite ingresar el nombre del archivo de texto a encriptar. Con los botones de la derecha puede visualizarse el archivo y seleccionar el mismo de la ventana de archivos.

Clave Pública:
Permite ingresar el nombre del archivo que contiene la clave pública generada en la opción correspondiente de este sistema. Con los botones de la derecha puede visualizarse el archivo y seleccionar el mismo de la ventana de archivos.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

2. Desencriptar

Al seleccionar la pestaña Desencriptar, se visualizará esta ventana. Permite el ingreso de datos para desencriptar archivos de texto con aplicación del algoritmo de desencriptación RSA. Se solicitan los siguientes datos



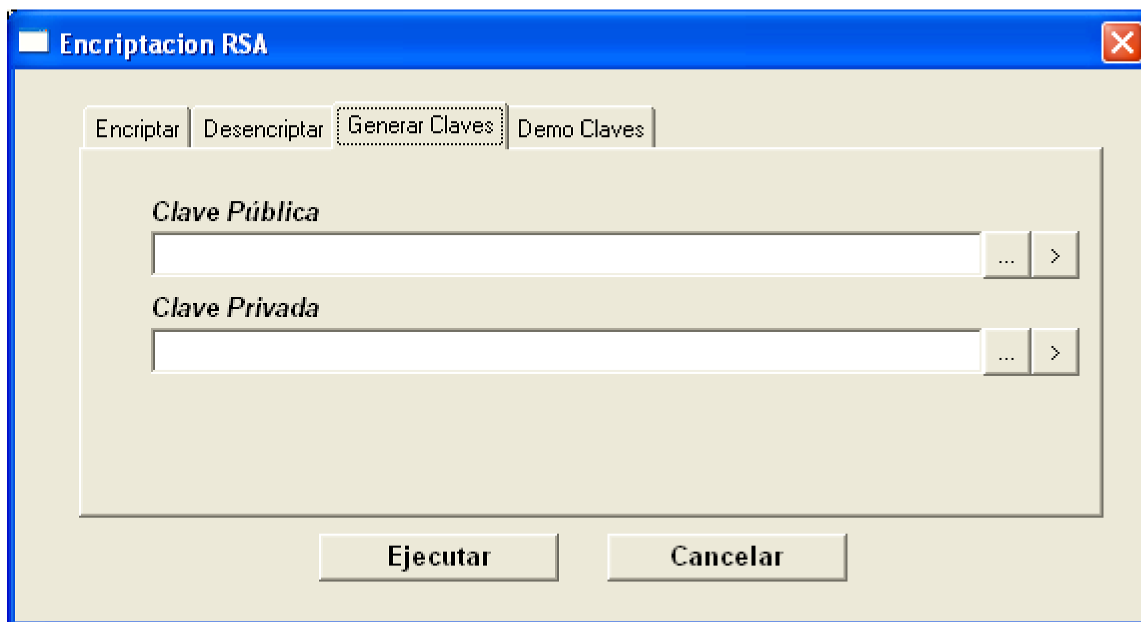
Archivo a Desencriptar:
Permite ingresar el nombre del archivo codificado a desencriptar. Este archivo debe haber sido generado con la opción correspondiente de este sistema. Con los botones de la derecha puede visualizarse el archivo y seleccionar el mismo de la ventana de archivos.

Clave Privada:
Permite ingresar el nombre del archivo que contiene la clave privada generada en la opción correspondiente de este sistema. Con los botones de la derecha puede visualizarse el archivo y seleccionar el mismo de la ventana de archivos.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

3. Generar Claves

Esta ventana permite generar claves de encriptación y desencriptación RSA de forma automática. Se solicitan los siguientes datos



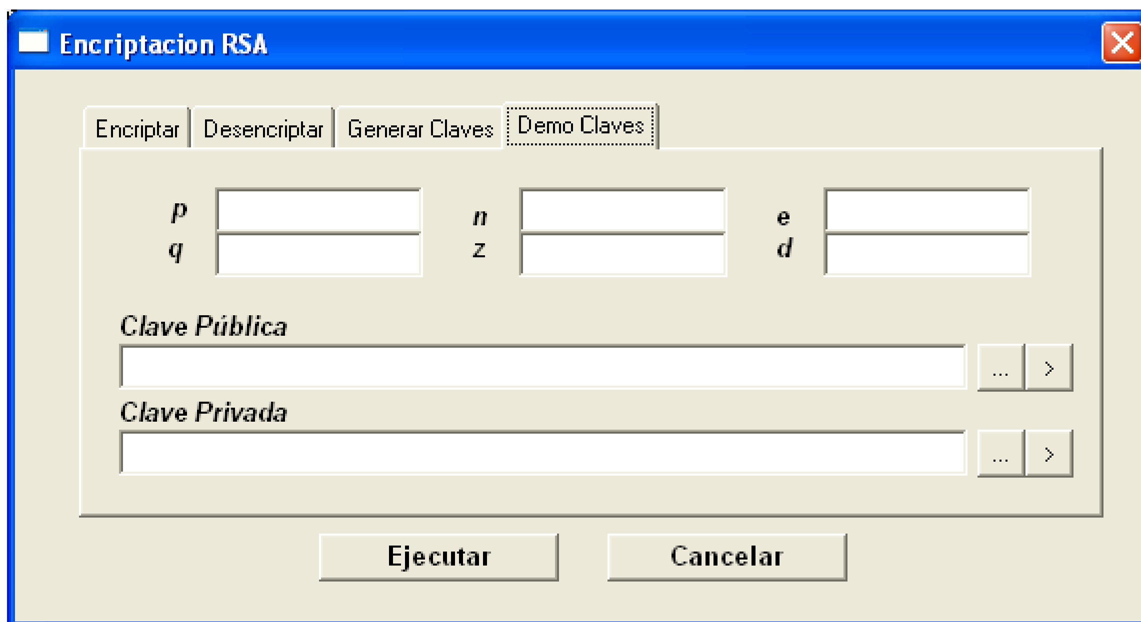
Clave Pública:
Permite ingresar el nombre del archivo que contendrá la clave pública. Con los botones de la derecha puede visualizarse el archivo y seleccionar el mismo de la ventana de archivos.

Clave Privada:
Permite ingresar el nombre del archivo que contendrá la clave privada. Con los botones de la derecha puede visualizarse el archivo y seleccionar el mismo de la ventana de archivos.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

4. Demo Claves

Esta ventana permite generar claves de encriptación y decriptación RSA según los valores ingresados para los números primos p y q.



p
Permite el ingreso del número primo p. Se asume que el valor ingresado es primo y no se verificará.

q
Permite el ingreso del número primo q. Se asume que el valor ingresado es primo y no se verificará.

n
Este valor es calculado por el sistema y representa el cálculo del módulo n=p*q

z
Este valor es calculado por el sistema de la siguiente manera z=(p-1)*(q-1)

e
Es la clave pública utilizada para encriptar. Es calculada por el sistema en base a los valores proporcionados como un número primo que inicia en 3 y que debe ser primo con n.

d
Es la clave privada utilizada para desencriptar. Es calculada por el sistema en base a los valores proporcionados como la inversa de e n módulo n.

Clave Pública:

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Permite ingresar el nombre del archivo que contendrá la clave pública. Con los botones de la derecha puede visualizarse el archivo y seleccionar el mismo de la ventana de archivos. Si este campo se deja en blanco no se almacenará la clave pública pero si se visualizará el resultado en la pantalla

Clave Privada:
Permite ingresar el nombre del archivo que contendrá la clave privada. Con los botones de la derecha puede visualizarse el archivo y seleccionar el mismo de la ventana de archivos. Si este campo se deja en blanco no se almacenará la clave privada pero si se visualizará el resultado en la pantalla

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Archivo de Configuración RSA.ini

El sistema cuenta con el archivo de configuración rsa.ini que permite parametrizar los siguientes valores:

Nombre	Valor sugerido	**Descripción**
Digitos	6	Dígitos por celda
TotalDigPrimo	128	Cantidad de dígitos de números primos
E	3	Valor por defecto de componente e de clave publica
T	17	Cantidad de Encriptación de algoritmo de Miller Rabin
ExtensionTexto	txt	Extensión de archivos de texto por defecto
ExtensionEncrip	rsae	Extensión de archivos encriptados por defecto
ExtensionDecrip	rsad	Extensión de archivos decriptados por defecto a ser leído de rsa.ini
ExtensionClaveEncrip	kge	Extensión de archivos de clave publica por defecto
ExtensionClaveDecrip	kgd	Extensión de archivos de clave privada por defecto
NombreEncripTmp	wrkfrsae.tmp	Nombre de archivo numérico temporal de Encriptación
NombreDecripTmp	wrkfrsad.tmp	Nombre de archivo numérico temporal de decriptación

En esta tabla se indican los valores por defecto pero pueden ser modificados por el usuario con el uso de cualquier editor de texto como el bloc de notas.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

TESIS PUCP

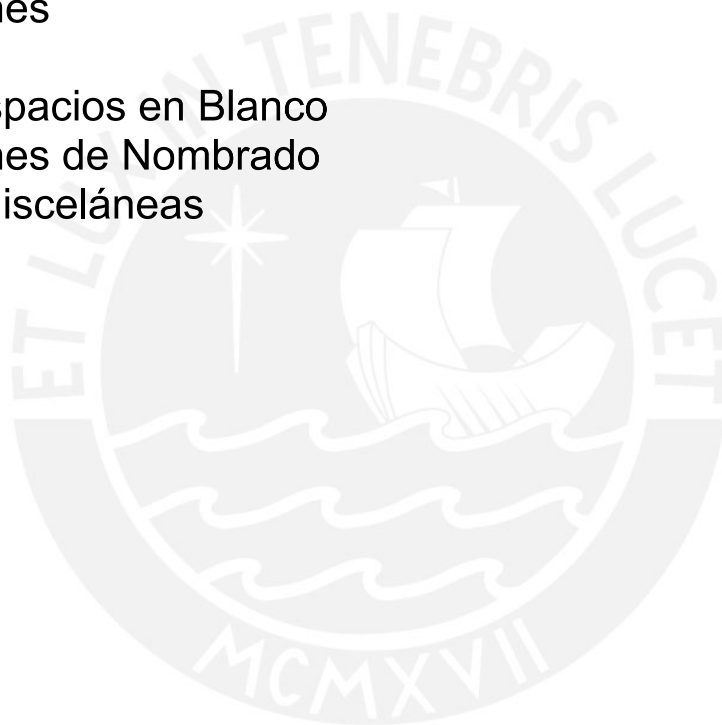
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Estándar de codificación JAVA

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

CONTENIDO

Introducción
Organización de archivos JAVA
Identación
Comentarios
Declaraciones
Sentencias
Líneas y Espacios en Blanco
Convenciones de Nombrado
Practicas Misceláneas



SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Introducción

Este documento es una adaptación del "Estándar de Codificación del Lenguaje Java" presentados en la "Especificación del Lenguaje Java" de Sun Microsystems, Inc. http://java.sun.com/docs/codeconv/

Es necesario contar como requisito para el entendimiento de la presente guía de conceptos de programación orientada a objetos (OOP)

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Organización de archivos JAVA

Los archivos java deben tener las siguientes partes:

- Comentario de Clase

Todos los archivos deben empezar con una cabecera de comentario que se describe a continuación:

```
/*
* Nombre de Clase y Fecha
*
* Propósito
* Desarrollado por
*
*/
```

- Paquete y Sección de Importación

  El nombre del paquete debe estar siempre escrito en minúscula.
  Ejemplo:

```
package java.awt;                    // paquete
import java.awt.peer.CanvasPeer;   // sección de Importación
```

- Clase

  Descripción de las partes de una clase

Comentarios de la clase (/**...*/)	
Sección clase	
Variables estáticas	Orden de declaración: 1. Por ámbito: Primero las variables de clase públicas, luego las protegidas y por ultimo las privadas 2. Por Tipo de dato: Agrupar las variables por tipo de dato
Variables de Instancia	Orden de declaración: a. Por ámbito:

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

	Primero las variables de clase públicas, luego las protegidas y por ultimo las privadas b. Por Tipo de dato: Agrupar las variables por tipo de dato
Constructores	
Métodos	Los métodos deben ser agrupados por funcionalidad en vez que por alcance y accesibilidad. Ejemplo: Un método privado puede estar entre dos métodos públicos. El objetivo es hacer más fácil y comprensible el código.

Identación

Se debe usar un tab como unidad de identación.

- Longitud de Línea

  Evitar líneas mayores de 80 caracteres.

- Ruptura de Líneas

  - Ruptura en declaración de clases y métodos

  Se identa a 8 espacios, romper después de una coma o antes de un operador.

  Ejemplo

```
someMethod(longExpression1, longExpression2, longExpression3,
        longExpression4, longExpression5);

var = someMethod1(longExpression1,
            someMethod2(longExpression2,
                    longExpression3));

longName1 = longName2 * (longName3 + longName4 - longName5)
                    + 4 * longname6;            // Preferido

longName1 = longName2 * (longName3 + longName4
                   - longName5) + 4 * longname6; // Evitar
```

  - Ruptura en implementaciones de métodos y constructores

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Se identa a 4 espacios, romper después de una coma o antes de un operador.

Ejemplo

```
public String desencriptarPassw( String p_cOrigen ) {

    String cCadAux   = "";
    String cCadResul = "";
    String cCadena1  = "";
    String cCadena2  = "";
    int    j         = 0 ;

    // Ejemplo de ruptura antes de un operador
    cCadena1="ABCDEFGHIJKLMNÑOPQRS380WXYZabcdefghijklmnñopqrstuvwyz123456"
        + "1203";

    for (j=0; j < p_cOrigen.length(); j++) {
        cCadAux=p_cOrigen.substring(j,j+1);

        // Ejemplo de ruptura despues de una coma
        cCadResul=cCadResul+cCadena2.substring(cCadena1.indexOf(cCadAux),
            cCadena1.indexOf(cCadAux)+1);

    }
    cCadResul="p"+cCadResul.substring(1);
    return  cCadResul;
}
```

Para sentencias if tenemos:

```
/* Usar identación a 8 espacios para rupturas de líneas en la sección de
 * condición del IF y 4 espacios para las sentencias dentro del bloque IF.
 */
if ((condicion1 && condicion2) || (condicion3 && condicion4)
        ||! (condicion5 && condicion6)) {
    ejecutarAccion();
}
```

Comentarios

Los comentarios no deben encerrarse en grandes cajas dibujadas con asteriscos u otros caracteres. Los comentarios nunca deberían incluir caracteres especiales como saltos de página, etc.

- Tipos de comentarios

  - Comentarios de implementación

    Están delimitados por **/*...*/** y **//**.
    Creado para comentar código o una implementación particular.

  - Comentarios de documentación (para generar Javadocs)

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Los comentarios de documentación son únicos de Java y están delimitados por **/**...*/**
Se pueden extraer a archivos HTML usando la herramienta Javadoc.

- Formatos de Implementación de Comentarios

Los programas pueden tener cuatro estilos de implementación de comentarios:

- Bloque de Comentarios

Los bloques de comentarios se usan para proporcionar descripciones de archivos, métodos, estructuras de datos y algoritmos. Los bloques de comentarios deben usarse al principio de cada archivo y antes de cada método. También pueden usarse en otros lugares, como dentro de los métodos. Los bloques de comentarios dentro de una función o métodos deben estar identados al mismo nivel que el código que describen. Un bloque de comentario debe ir precedido por una línea en blanco para configurar un apartado del resto del código:

```
/*
 *  Aquí hay un  bloque de comentario
 */
```

- Comentarios de una línea

Los comentarios cortos pueden aparecen como una sola línea identada al nivel del código que la sigue. Si un comentario no se puede escribir en una sola línea, debe seguir el formato de los bloques de comentario. Un comentario de una sola línea debe ir precedido de una línea en blanco.
Ejemplo:

```
if ( condicion ) {

    /* Manejo de la condición. */
    ...
}
```

- Comentarios finales

Los comentarios muy cortos deben aparecer en la misma línea que el código que describen, pero deben separarse lo suficiente de las sentencias. Si aparece más de un comentario en el mismo trozo de código, deben estar identados a la misma altura.

Ejemplo:

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

```
if (a == 2) {
    return true;                /* comentario..*/
} else {
    return esPrimo(a);          /* comentario..*/
}
```

- Comentarios de final de línea

  El delimitador de comentario “//” puede comentar una línea completa o una línea parcial.
  No debería usarse en líneas consecutivas para comentar texto; sin embargo, si puede usarse en líneas consecutivas para comentar secciones de código.
  Ejemplo

```
if (foo > 1) {
    // Comentando la línea completamente
    ...
}
else {
    return false;  // comentando parcialmente la línea.
}
```

- Comentarios de Documentación

  Los comentarios de documentación describen clases Java, interfaces, constructores, métodos y campos. Cada comentario de documentación va dentro de los delimitadores de comentario **/**...*/**, con un comentario por clase, interfase o miembro.
  Este comentario debe aparecer justo antes de la declaración:

```
/**
 *  1.- Detallar aquí una descripción Funcional
 *  2.- El doble asterisco significa que este bloque de comentario aparecerá
 *      en el Javadoc generado
 */
public class Ejemplo {
    ...
}
```

Declaraciones

- Número de declaraciones por línea

  - Se debe realizar solo una declaración por línea.
  - No debemos poner diferentes tipos sobre una misma línea.
  - No es recomendable declarar una lista sucesiva de variables.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

- Agrupar por tipo de datos el orden de las declaraciones.
- Alinear las columnas de las declaraciones.

Ejemplo:

```
/* Declaraciones correctas */
int       nNivel       = -1;
int       nCantidad    =  0;
String    cCadenaAux   = "";
String    cIdDocumento = "";
Hashtable colTabla     = null;

/*
 *  Declaraciones incorrectas:
 *  1.- Variables no inicializadas.
 *  2.- Se debe declarar una variable por línea.
 *  3.- Se debe agrupar por tipo de dato.
 */
int nNivel, nPeso;
float nAncho, arrListaAux [ ];
int nCantidad;
```

- Características para declaraciones

    - Inicialización

      Debemos intentar inicializar las variables locales donde son declaradas. La única razón para no inicializar una variable donde es declarada es si el valor inicial depende de algún cálculo que tiene que ocurrir antes.

    - Situación

      Ponemos las declaraciones sólo al principio de los bloques (Un bloque es cualquier código encerrado entre corchetes “{” y “}”.)

      Ejemplo:

```
void ejecutarProceso() {
    int nNumero = 0;          // comienzo del bloque
    if (condicion) {
        int nNumero2 = 0;     // comienzo del bloque "if"
        ...
    }
}
```

La única excepción a esta regla son las declaraciones en bucles, que en Java pueden ser declarados en la sentencia for:

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

```
for (int i = 0; i < 5; i++) { ... }
```

- Declaraciones de Clases e Interfaces

  Cuando codificamos clases e interfaces Java, se deben seguir las siguientes reglas de formateo:

  - Para la declaración de métodos de las clases, no debe haber ningún espacio entre el nombre y el paréntesis "(" que inicia la lista de parámetros.
  - El corchete abierto "{" aparece en la misma línea que la declaración.
  - El corchete cerrado "}" empieza una línea por sí mismo, identado a su correspondiente sentencia de apertura, excepto cuando es una sentencia nula en la que el "}" debería aparecer inmediatamente después del "{"
  - Los métodos están separados por una línea en blanco.

  Ejemplo:

```
class Sample extends Object {
    private int nNumero1 = 0;
    private int nNumero2 = 0;

    //declaración de método con dos parámetros de entrada
    public void Sample(int p_iValor1, int p_Valor2) {

        nNumero1 = p_Valor1;
        nNumero2 = p_Valor2;
    }

    private int metodoVacio() { }  //declaración de método con sentencia nula

    ...
}
```

Sentencias

- Sentencias Simples

  Cada línea debe contener como máximo una sentencia. Por ejemplo:

```
nArgv++;            // Correcto

nArgc++;            // Correcto

nArgv++; nArgc--;   // Incorrecto
```


SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

- Sentencias Compuestas (Secciones de código)

  Las sentencias compuestas son sentencias que contienen listas de sentencias encerradas entre corchetes “{ sentencias }”

  El corchete abierto “{” debe estar al final de la línea que empieza la sentencia compuesta; el corchete cerrado ”}” debe empezar en una nueva línea y estar identado con el principio de la sentencia compuesta.

  Si dentro de una sentencia if – else, for, etc se aplica una sola sentencia esta de igual modo debe estar encerrada dentro de corchetes, esto hace más fácil la adición de sentencias sin introducir errores debido al olvido de los corchetes. Ejemplo:

```
if ( nNumGuia <= 1000 ) {
    nContador++;
    agregarPedido(nContador);
    if ( nContador > 10 ) {
        generarCorte( lista ) //Ejemplo de bloque if con una sola sentencia
    }
}
```

- Una sentencia de retorno no debería usar paréntesis a menos que sea necesario.

Ejemplo:

```
return;                // retorno sin devolución de dato
return nNumImpresos;   // retorno con devolución del valor de una variable
/* retorno con devolución condicionada, aquí se uso paréntesis */
return ( nNumImpresos > 10 ? nNumImpresos * 3.2 : nNumImpresos * 4);
```

- Sentencias if, if-else, if else-if else

  Las sentencias del tipo if-else deberían tener la siguiente forma:

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

```
if ( condicion) {
    sentencias;
}
if ( condicion) {
    sentencias;
} else {
    sentencias;
}
if ( condicion) {
    sentencias;
} else if ( condicion) {
    sentencias;
} else {
    sentencias;
}
```

- Sentencias for

  Una sentencia for debe tener la siguiente forma:

```
for (inicializacion; condicion; actualizacion) {
    sentencias;
}
```

  Una sentencia for vacía (una en la que todo el trabajo se hace en las cláusulas de inicialización, condición y actualización) debería tener la siguiente forma:

```
for ( inicializacion; condicion; actualizacion) { }
```

  Cuando usamos el operador coma en las cláusulas de inicialización o actualización de una sentencia for, debemos evitar la complejidad de usar más de tres variables y separar las sentencias con un espacio dentro de estas cláusulas.

  Ejemplo:

```
for (i=1, j--: i > 10; i++, muestra(j)) { }
```

- Sentencias while

  Una sentencia while debería tener la siguiente forma:

```
while (condicion) {
   sentencias;
}
```

  Una sentencia while vacía debería tener la siguiente forma:

```
while (condicion) { }
```


SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

- Sentencias do-while

Una sentencia do-while debería tener la siguiente forma:

```
do {
    sentencias;
} while (condicion);
```

- Sentencias switch

Una sentencia switch debería tener la siguiente forma:

```
switch( nType ) {
    case Types.TINYINT: {
        cClassName = "java.lang.Byte";
        break;
    }
    case Types.SMALLINT: {
        cClassName = "java.lang.Short";
        break;
    }
    case Types.INTEGER: {
        cClassName = "java.lang.Integer";
        break;
    }
    case Types.OTHER: case Types.JAVA_OBJECT: {
        cClassName = "java.lang.Object";
        break;
    }
    default: {
        return super.getColumnClass(p_nColumn);
    }
}
```

Toda sentencia switch debe incluir un valor default.

- Sentencias try-catch

Estos bloques permiten la captura y control de excepciones, se recomienda su continuo uso.
Una sentencia try-catch debería tener la siguiente forma:

```
try {
    sentencias;
} catch (ExceptionClass e) {
    sentencias;
}
```

Una sentencia try-catch también podría ir seguida de un bloque finally, que se ejecuta sin importar si se ha completado con éxito o no el bloque try.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

```
try {
   sentencias;
} catch (ExceptionClass e) {
   sentencias;
} finally {
            sentencias;
}
```

Líneas y Espacios en Blanco

- Líneas en Blanco

  Las líneas en blanco mejoran la lectura separando secciones de código que están relacionadas lógicamente.

  Se debe usar dos líneas en blanco en las siguientes circunstancias:

  - Entre secciones de un archivo fuente
  - Entre definiciones de clases *e* interfaces

  Se debe usar una línea en blanco en las siguientes circunstancias:

  - Entre métodos
  - Entre las variables locales de un método y su primera sentencia
  - Antes de un bloque de comentarios o un comentario simple
  - Entre secciones lógicas dentro de un método para mejorar su lectura

- Espacios en Blanco

Los espacios en blanco deben usarse en las siguientes circunstancias:

- Una palabra clave seguida por un paréntesis debe estar separados por un espacio en blanco, ejemplo:

```
while (true) {
    ...
 }
```

- Observar que no se debería usar un espacio en blanco entre un nombre de método y su paréntesis de apertura. Esto ayuda a distinguir las palabras claves de las llamadas a métodos.

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Ejemplo:

```
public String sendMessage(
        String[][] values ) throws BusException {
```

- Después de las comas en una lista de argumentos debe aparecer un espacio en blanco.

  Ejemplo:

```
public void cargarResultados( String p_cTes, String p_cCes,
        String p_cAfiliado, String p_cProv,
        String p_cSucProveedor, String p_cBeneficio, String p_cEspecialidad,
        String p_cProcedimiento)
        throws SQLException, IOException {
```

- Todos los operadores binarios excepto "." deben estar separados de sus operandos por espacios. Los espacios en blanco nunca deben separar los operadores unarios como incremento ("++") y decremento ("--") de sus operandos.

  Ejemplo:

```
a += c + d;
a = (a + b) / (c * d);
while (d++ = s++) {
    n++;    //operador unario, no lleva espacio en blanco
}
```

- Las expresiones de una sentencia deben estar separadas por espacios, ejemplo:

```
for (expr1; expr2; expr3)
```

- Los CAST deben ir seguidos de un espacio en blanco, ejemplo:

```
myMethod((byte) aNum, (Object) x);

myMethod((int) (cp + 5), ((int) (i + 3)) + 1);
```

Convenciones de Nombrado

Se detallan las siguientes reglas de acuerdo a los tipos de identificadores:

- Paquetes

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Cada identificador que compone un paquete debe estar escrito en letras minúsculas y en singular.

Ejemplo:

```
package com.apple.quicktime.v2
package  com.sun.eng
```

- Clases

  - Los nombres de clases deben ser mezclas de mayúsculas y minúsculas, con la primera letra de cada palabra interna en mayúsculas.
  - Se debe usar palabras completas, evitar acrónimos y abreviaturas
  - No usar prefijos de tipo, como C para clase, en un nombre de clase. Utilice, por ejemplo, el nombre de clase GestorArchivo en vez de CGestorArchivo
  - No utilice el carácter de subrayado "_"
  - De vez en cuando, es necesario proporcionar un nombre de clase que comience con la letra "I", aunque la clase no sea una interfaz. Esto es correcto siempre que "I" sea la primera letra de una palabra que forme parte del nombre de la clase.

  Ejemplo:

  ```
  class Pedido
  class DocumentoPago
  class IdentityStore        // Nombre de clase que comienza con la letra "I"
  ```

- Interfaces

  - Asignar nombres a interfaces utilizando sustantivos, sintagmas nominales o adjetivos que describan su comportamiento.
  - Los nombres de clases deben ser mezclas de mayúsculas y minúsculas, con la primera letra de cada palabra interna en mayúsculas.
  - Utilice las abreviaturas con moderación.
  - Incluya un prefijo con la letra "I" en los nombres de interfaces para indicar que el tipo es una interfaz.
  - No utilice el carácter de subrayado "_"

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Ejemplo:

interfase IProveedorServicio
interfase IDirectorioMedico

- Métodos

  - Se debe declarar métodos con la primera letra del nombre en minúsculas y con la primera letra de cada palabra interna en mayúsculas.

  - La primera palabra de la declaración debe ser un verbo activo

Ejemplo:

imprimir();
enviarCorreo();
generarDetalleKardex();

- Variables

Se distinguen los siguientes tipos:

- Variables de ámbito y propiedades de clase

Formato: xNombreVariable	
x	Carácter que indica el tipo de dato de la variable: c Carácter *n* Numérico *d* Fecha b Lógico Otros tipos llevan la inicial del tipo
Nombre de la Variable	La primera letra de cada palabra interna va en mayúscula. No usar caracteres "_" y "$" Usar nombres descriptivos

Ejemplo	`//Variables locales`
	`char cIndEstadoFact;`
	`int nAnchoPagina ;`
	
	`//Propiedades o campos de la clase`
	`private float nMontoCobrar;`
	`public int nNumIngresos;`

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

- Variables contadores

Formato: i,j,k	

Ejemplo	`for (i=1; i <= 5 ;i++) {`
	`    mostrarProducto(i);`
	`}`

- Variables de excepciones

Formato: *eNombreExcepcion*	
e	Prefijo que identifica a una excepción
Nombre Excepción	Identificador de la excepción. No usar caracteres "_" y "$"

Ejemplo	`Try {`
	`  …`
	`}`
	`catch (Exception eExcp)  {`
	`      eExcp.StackTrace();`
	`}`

- Parámetros de los métodos

Formato: *p*xNombreVariable	
p	Carácter que indica que la variable es argumento de un método perteneciente a una clase.
x	Carácter que indica el tipo de dato de la variable: c Carácter *n* Numérico *d* Fecha b Lógico Otros tipos llevan la inicial del tipo
Nombre de la Variable	La primera letra de cada palabra interna va en mayúscula. No usar caracteres "_" y "$" Usar nombres descriptivos

Ejemplo	`public void checkUser( String pcLogin, String pcPassw)`
	`      Throws SQLException {`
	`      …`
	`}`

- Constantes en Java

Formato: xNOMBRECONSTANTE	
	Carácter que indica el tipo de dato de la constante:

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

x	c Carácter *n* Numérico *d* Fecha b Lógico Otros tipos llevan la inicial del tipo
Nombre de la Constante	Las constantes se escriben en mayúsculas con las palabras separadas por subrayados "_" Permitido solo caracteres alfanuméricos. Utilizar nombres descriptivos.
Ejemplo	`static final int nMIN_PRECIO = 4 ;` `static final int nMAX_PRECIO = 100;`

-
- Arreglos

Aplica para arreglos unidimensionales o de múltiples dimensiones.

Formato: xArrNombreArreglo	
x	Carácter que indica el tipo de dato de cada elemento del arreglo c Carácter *n* Numérico *d* Fecha b Lógico
Arr	Identificador que indica que la variable es un arreglo Arr array
Nombre del arreglo	La primera letra de cada palabra interna va en mayúscula. No usar caracteres "_" y "$" Usar nombres descriptivos

Ejemplo	`// arreglos unidimesionales`
	`float []    nArrListaPrecios;`
	`boolean []  bArrEstadosFin;`
	
	`// arreglos bidimensional`
	`Object [][] oArrMatriz;`

Anexos

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

- Proporcionar Acceso a Variables de Instancia y de Clase

  No se debe hacer públicas ninguna variable de instancia o de clase sin una buena razón.

- Asignaciones de Variables

  - Se debe evitar asignar a varias variables el mismo valor en una sola sentencia. Es difícil de leer.
    Ejemplo:

```
cCadena1 = cCadena2 = "Teoria"; //Incorrecto
```

No se debe usar el operador de asignación en lugares donde pueda ser fácilmente confundible con el operador de igualdad.

Ejemplo:

```
if ( c++ = d++ ) { // Incorrecto (Java rechaza)
    ...
}

/* Debería escribirse así: */
if (( c++ = d++ ) != 0) {
    ...
}
```

No se debe usar asignaciones embebidas en un intento de mejorar el rendimiento de ejecución. Ese es el trabajo del compilador.

Ejemplo:

```
d = (a = b + c) + r; // Incorrecto
// Debería escribirse como:
a = b + c;
d = a + r;
```

- Palabras reservadas en Java

  Las siguientes son las palabras clave que están definidas en Java y que no se pueden utilizar como identificadores:

```
cast        future      generic     inner
operator    outer       rest        var
```


SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

```
abstract     continue     for          new          switch
boolean      default      goto         null         synchronized
break        do           if           package      this
byte         double       implements   private      threadsafe
byvalue      else         import       protected    throw
case         extends      instanceof   public       transient
catch        false        int          return       true
char         final        interface    short        try
class        finally      long         static       void
const        float        native       super        while
```

Practicas Misceláneas

- Paréntesis
  Se recomienda usar paréntesis para liberar expresiones que incluyen mezclas de operadores para evitar problemas de precedencia de operadores. Incluso si la precedencia del operador parece clara para nosotros.

```
if (a == b && c == d)          // Incorrecto
if ((a == b) && (c == d))      // Correcto
```

- Valores de

  Debemos intentar hacer que la estructura de nuestro programa corresponda con nuestra intención.
  Ejemplo:

```
/* Incorrecto, bloque que incluye una programación de retorno redundante */
if ( condicion ) {
    return true
} else {
    return false
}
/* Correcto */
return condicion;
```

- Expresiones antes del '?' en el Operador Condicional

  Si una expresión que contiene un operador binario aparece antes del '?' en el operador terciario ?: debería ponerse entre paréntesis.
  Ejemplo:

```
(x >= 0) ? x : -x;
```


SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Listado de Programas

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

BIBLIOGRAFÍA

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

RSA Laboratories, "RSA Security Inc. Public-Key Cryptography Standards (PKCS)", tomado de ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf el 28 de noviembre del 2004

RSA Laboratories, "RSAES-OAEP Encryption Scheme", tomado de www.rsasecurity.com el 25 de enero del 2005

Fco. Javier Cobos Gavala. "Introduccion a la Matematica Discreta", tomado de http://ma1.eii.us.es/Docencia/apuntes/ap_imd.pdf el 25 de enero del 2005.

Sun MicroSystems, Inc, “Java Code Conventions” tomado el 02 de mayo del 2005 de http://java.sun.com/docs/codeconv/CodeConventions.pdf

Loidreau, Piere, tomado de http://ldp.rtin.bz/linuxfocus/Castellano/May2002/article243.shtml el 11 de abril del 2005.

http://www.reuna.cl/central_apunte/apuntes/soc_info5.html, tomado el 11 de abril del 2005

Manuel Lucena, "Criptografía y Seguridad en Computadores", 3ra Edición actualiza y ampliada, v2.01 (3 de marzo de 2003) tomado el 11 de abril del 2005 de http://www.criptored.upm.es/guiateoria/gt_m027a.htm.

"RSA", tomado de http://en.wikipedia.org/wiki/RSA el 16 de febrero del 2005

"Public Key Cryptography", Robert Rolland, Diciembre 2001, tomado el 12 de abril del 2005 de http://tlapixqui.izt.uam.mx/cimpa/files/rolland/pub_key.pdf

"Introducción a la Criptografía", http://rinconquevedo.iespana.es/rinconquevedo/criptografia/rsa.htm, tomado el 11 de abril del 2005

"Mathematics", Paul Johnston, tomado de http://pajhome.org.uk/crypt/rsa/maths.html el 08 de marzo del 2005. Ruta completa Paj's Home: Cryptography: RSA: Mathematics

"Mathematics", Paul Johnston, tomado de http://pajhome.org.uk/crypt/rsa/maths.html el 08 de marzo del 2005. Ruta completa Paj's Home: Cryptography: RSA: Mathematics

"Coprime", http://en.wikipedia.org/wiki/Coprime tomado el 16 de febrero del 2005

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

"Primality Test", http://en.wikipedia.org/wiki/Primality_test tomado el 16 de febrero del 2005

"Miller-Rabin Primality Test", http://www.cryptomathic.com/labs/rabinprimalitytest.html tomado el 21 de febrero del 2005

http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm tomado el 25 de julio del 2005

"Exponentiation by squaring", http://en.wikipedia.org/wiki/Square-and-multiply_algorithm tomado el 23 de julio del 2005

"Adaptive chosen ciphertext attack",, tomado el 16 de febrero del 2005 de http://en.wikipedia.org/wiki/Adaptive_chosen_ciphertext_attack

"RSA", tomado el 16 de febrero del 2005 de http://en.wikipedia.org/wiki/RSA

Introducción a la criptografia. Pino Caballero Gil. Editorial RA-MA.

Codificación de la información. Carlos Munuera, Juan Tena. Universidad de Valladolid.

Códigos secretos. Andrea Sgarro. Editorial Piramides.

http://www.Kriptopolis.com

http://www.eff.org/pub/Privacy/Crypto_misc/DESCracker/HTML/19990119_deschallenge3.html

http://www.pgp.com

http://www.gvsu.edu/mathstat/enigma/enigma.html

rinconquevedo.iespana.es/rinconquevedo/Criptografia/contenido.htm, Lenguaje C rsa2000.exe versión 1.0

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis

Garcia de Jalón, Javier y otros, ”Aprenda Java”, Universidad de Navarra, San Sebastián, Febrero 2000. Tomado el 06 de mayo del 2005 de http://www.iua.upf.es/~progiii/practiques/docs/cursojava.pdf

Bruce Eckel, “Thinking in Java”, 2nd Edition, Revision 12, 2000, Bruce Eckel Ed.

Java 2, Editorial San Marcos, 2003, Primera Edición

Lin Zhong, “Public Key Algorithms and the related Mathematics”, Princeton University tomado el 08 de marzo del 2005 de http://www.princeton.edu/~lzhong/talks/ele572.pdf

SOME RIGHTS RESERVED
Tesis publicada con autorización del autor
Algunos Derechos Reservados. No olvide citar esta tesis