UNIVERSIDAD DEL BIO-BIO

Facultad de Ciencias Empresariales
Departamento Sistema de Informaciones

MEMORIA PARA OPTAR A TITULO DE INGENIERO DE EJECUCIÓN EN COMPUTACIÓN E INFORMÁTICA

Implementación de simulador gráfico para algoritmos de firma digital grupal con garantías de anonimato

Alumno
Pablo Torres Osses

Profesor Guía
Patricio Galdames Sepúlveda

Concepción, 2017

Resumen

La firma digital grupal, consiste en una prueba de autenticidad de un documento creado por un grupo de usuarios para que se les reconozca como sus creadores. En diversos contextos se desea además que un subconjunto del grupo usuarios pueda generar una firma grupal bajo la condición que no se conozca la identidad de los firmantes del subconjunto. Tan solo se puede concluir que los firmantes pertenecen al grupo.

En este trabajo de título se presenta la implementación y evaluación de una librería Java para los algoritmos de firma digital grupal con garantías de anonimato propuestos por Bresson et al [2]. Dado un grupo de posibles firmantes, quienes conforman un grupo, Bresson et al. proponen tres algoritmos de firma grupal: 1) Hay un solo un firmante, quien genera mediante la combinación de criptografía simétrica y asimétrica una firma anónima en representación de todo el grupo. 2) la segunda variante también le permite crear a un usuario una firma anónima, pero requiere combinar criptografía asimétrica con funciones de hashing. 3) La ultima variante considera a un subgrupo de firmantes, de tamaño mayor a uno y menor al tamaño del grupo, quienes generan una firma anónima en representación del grupo.

En un trabajo de título previo [1] se implementó solamente la primera versión antes mencionada. Por lo tanto, el aporte de este trabajo es el siguiente: 1) implementación del algoritmo para un firmante anónimo basado en criptografía simétrica y funciones hashing. 2) Implementación de la versión que permite la firma grupal para múltiples usuarios. 3) Se comparan el rendimiento de las versiones de un firmante y se comprueba que el algoritmo basado en funciones de hashing obtiene el menor tiempo de ejecución. 4) Finalmente, las librerías desarrolladas en Java fueron probadas y evaluadas en un simulador gráfico que también describe el funcionamiento de los mismos algoritmos.

Índice

Figuras

Ecuaciones

Tablas

Frame

Casos de uso

1. Introducción

La firma digital es el equivalente de la firma tradicional realizada con un lápiz de tinta, que busca dejar una marca personal y de autoría en un documento que solo pueda ser realizada por un individuo. Por ejemplo, cuando se lleva a cabo un contrato de compra y venta de un automóvil, el contrato debe concluir con la firma del comprador, como prueba de la veracidad de la operación comercial. En la firma digital tal 'marca característica' puede ser creada mediante el cálculo de una ecuación criptográfica que ejecuta el firmante. Tal marca puede ser fácilmente obtenida por el firmante, pero es casi imposible de replicar por otra persona.

Los algoritmos existentes que implementan la firma digital permiten al igual que la firma tradicional que el proceso de verificación de la misma firma sea fácilmente comprobable. En el caso tradicional, la firma del papel es verificada tras compararla con alguna almacenada previamente o aquella registrada en algún documento de identidad oficial como la cedula de identidad o pasaporte. En el caso digital el proceso de verificación es similar.

El algoritmo típico para implementar una firma digital emplea sistemas criptográficos asimétricos, los cuales generalmente requieren de dos parámetros de entrada (además del documento):

- El primer parámetro llamado **llave pública** es conocido por todos, y es utilizado por cualquier usuario que quiera verificar una firma con el fin de concluir con la autenticidad de un documento.
- El segundo parámetro llamado **llave privada** es secreta y por ende conocido solo por el usuario al que le pertenece. La utiliza su dueño para firmar un documento.

Un ejemplo real del uso de la firma digital consiste en una empresa del rubro de encomiendas que necesita proporcionar al remitente una prueba que el destino recibió un paquete. En general, el cartero o el representante de la empresa de encomiendas, le solicita al receptor que firme un certificado de entrega. Tal firma servirá de prueba al remitente que el destino recibió efectivamente el paquete. En el caso que el destino firmase digitalmente el certificado de recepción, el remitente (o cualquier persona) puede

verificar la veracidad de la entrega empleando la llave pública del receptor sobre el certificado de recepción.

Un caso especial de la firma digital es la **firma digital grupal**. Esta consiste en un **grupo de n > 1 usuarios** que desea como grupo proclamar autoría sobre un determinado documento. También es posible permitir que un subgrupo de t integrantes cualesquiera del grupo (t<n) tengan permiso para firmar en representación de todo el grupo. En el caso tradicional, la firma grupal exige que cada uno de los integrantes del grupo o subgrupo deba de estampar su firma en el documento. Por tanto, para verificar que el grupo o subgrupo firmo, tan solo basta con chequear que el documento presenta el número de firmas mínimo requeridas y que estas firmas correspondan a aquellas que pertenezcan a miembros del grupo. Una firma digital grupal que posea características similares a la tradicional puede ser fácilmente implementada con los algoritmos de criptografía asimétrica existentes.

Sin embargo, esta implementación presenta un inconveniente, requiere conocer la identidad de los firmantes. En algunos casos es deseable que las identidades de los usuarios que participaron de la creación de la firma permanezcan anónimas. Es decir, queremos crear una firma que sea fácilmente verificable que pertenezca al grupo pero que la identidad de los firmantes no sea conocida más allá del hecho que sea comprobable que estos firmantes son miembros del grupo original (cuyas identidades si son conocidas).

Los primeros algoritmos de firma digital grupal con anonimato que se desarrollaron, permitieron que del grupo **n usuarios, solo 1 de ellos sea el firmante**. Es decir, cada uno de los miembros del grupo puede generar una firma que represente a todo el grupo. Estos algoritmos permiten preservar la identidad del firmante.

Un ejemplo sobre la firma grupal para un firmante, consiste en que una empresa elige una nueva inversión cada año. Para confirmar legalmente tal inversión, se requiere que uno cualquiera de los miembros de la junta directiva firme el contrato de inversión, para que esta sea válida. En este contexto se desea probar la veracidad de la firma grupal no la identidad de quien la emitió, ya que se entiende que todos los miembros del grupo tienen el poder de firmar documentos en representación de todos los inversionistas.

Existen diversos escenarios prácticos donde es deseable que al menos un número t > 1 de firmantes sea requerido para crear una firma digital grupal. Se acepta que cualquier combinación de t usuarios pueda crear una firma válida y al igual que el caso anterior se desea que las identidades de los “t” firmantes permanezca anónima.

Siguiendo con el ejemplo de la empresa que quiere realizar una inversión, ahora se establece que para aprobar dicho trámite se requiere que al menos un número mínimo de

los integrantes de la directiva firme para aprobar la inversión.

En la figura 1 se describe la taxonomía empleada en este trabajo para la implementación de la firma digital grupal. Para la firma digital con un firmante se emplea la técnica denominada “ring signature” o técnica del anillo [3]. Para la firma digital grupal con t>1 firmantes se implementa una generalización del algoritmo anterior que se le denomina “threshold ring signature” o técnica del anillo con umbral [2].

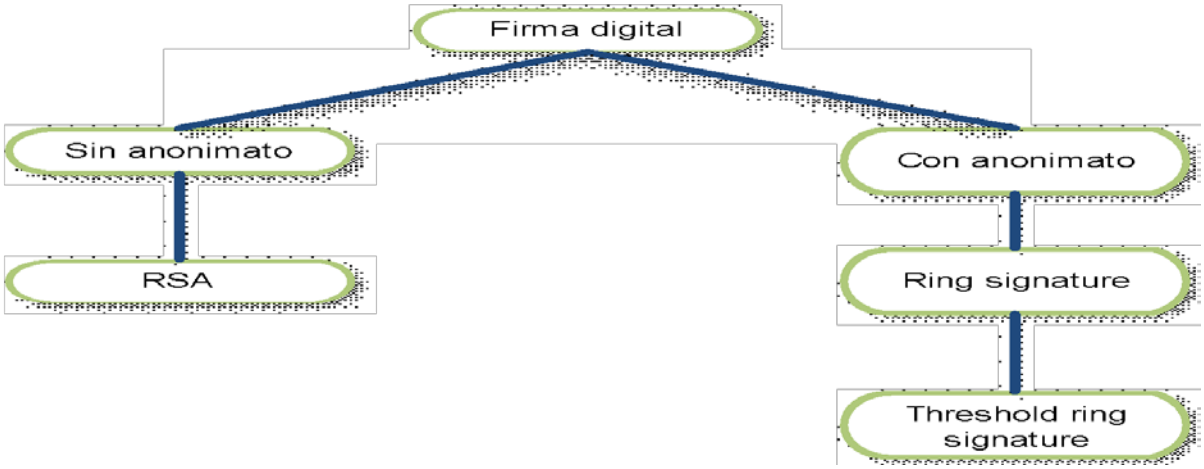


Figura 1: representación de la estructura de la firma digital presentada en el proyecto, don de la firma digital se pueden extraer dos variaciones, una firma anónima y una firma pública. La firma anónima puede ser para múltiples o un solo usuario perteneciente a un grupo.

El aporte de este proyecto de título se presenta en las siguientes implementaciones:

- Se verifica la implementación base del algoritmo ring signature presentado en [1].
- Se evalúan empíricamente 2 implementaciones de ring signature.
- Se implementa ring signature basado en funciones de HASH.
- Se implementa el algoritmo threshold ring signature.
- Se implementa la interfaz gráfica que permite la creación y verificación de una firma digital grupal con anonimato.

Para la simulación gráfica de los algoritmos se ha diseñado una interfaz de usuario que proporciona, a grandes rasgos, tres funciones:

- Herramientas de ejecución de los algoritmos: son encargadas de seleccionar el algoritmo a utilizar, seleccionar las dimensiones de los grupos y firmantes, crear un mensaje para firmar, realizar la operación de cálculo de la firma y realizar la operación de verificación de la firma.
- Representación animada de los algoritmos: encargada de desplegar una

animación de los procesos de la firma mediante las herramientas de gráficas 2D de java.

- Comunicador de estado de los algoritmos: se encarga de entregar el estado de la simulación gráfica y sus componentes.

Este trabajo de título describe lo expuesto con más detalles en las siguientes secciones. La sección 1 del documento contiene la definición del proyecto, la cual indica el origen y estructura general de los algoritmos a implementar, por lo demás también se indican los objetos del proyecto. La sección 2 del documento contiene la taxonomía de la firma digital, por lo demás se contextualiza la firma grupal. Luego en la sección 3 se realiza una definición de conceptos previos fundamentales tales como RSA, la firma digital de un solo firmante y la firma digital de múltiples firmantes. La sección 4 es el inicio experimental del documento, ya que explica en detalle la interpretación de la estructura del algoritmo ring signature. En la sección 5 se mejora la estructura de ring signature. En la sección 6 se da a conocer cómo debe ser entendió el concepto de grupos ad-hoc, para luego en la sección 7 aplicar este conocimiento e implementar threshold ring signature. En la sección 8 se realizan pruebas sobre la implementación del algoritmo threshold ring signature. Finalmente, en la sección 9 se describe brevemente la implantación de la interfaz gráfica encargada de representar los algoritmos implementados, y en la sección 10 y 11 se finaliza con la conclusión y referencias que fueron utilizadas en el documento.

2. Definición del proyecto

2.1. Objetivo general

Desarrollar una aplicación que simule gráficamente tres algoritmos de firma digital grupal que garantizan anonimato [2].

2.2. Objetivos específicos

- Realizar estudio de los algoritmos de firma grupal propuestos en [2]
- Revisión de implementación en java de la variante 1 realizada en proyecto de título previo [1]
- Implementación de librería java de la segunda variante y del algoritmo de firma de múltiples firmantes propuestos en [2].
- Diseñar e implementar mediante método de prototipo una interfaz gráfica para la prueba de los algoritmos de firma digital grupal con anonimato propuestos en [2].
- Evaluación empírica de la implementación en Java de las variantes 1 y de 2 con un solo firmante anónimo propuestos en [2].

3. Antecedentes previos

En el trabajo [1], se investigaron diversas fuentes de información, de modo de cumplir con los requerimientos de anonimato, y se concluyó que el algoritmo más idóneo correspondía a threshold ring signature [1]. Este algoritmo es un tipo de firma grupal con garantías de anonimato, el cual permite a un grupo de personas interactuar para construir una firma digital y, como consecuencia de la interacción, una entidad externa verifica la autenticidad la firma generada sin poder comprobar la identidad de los firmantes. Este algoritmo consta de dos partes.

- Parte 1: ring signature basado en funciones de HASH.
- Parte 2: threshold ring signature, que debe contar con el algoritmo de simulación, algoritmo de la partición justa, algoritmo de súper anillo y el algoritmo de verificación.

En [1] se implementó solo la primera parte del algoritmo ring objetivo, es decir, solo se codificó ring signature simple, basado en criptografía simétrica. En este proyecto se propuso implementar una mejora de este algoritmo basado en funciones de hash propuesto en [2] y realizar la implementación del algoritmo de firma en anillo con umbral propuesto por el mismo artículo académico.

Las modificaciones que se realizarán al algoritmo ring signature son dos:

- En primer lugar, el código del algoritmo implementado en [1] será rediseñado a nivel estructural, ya que no aplica en su totalidad el paradigma de programación orientada a objeto
- En un segundo lugar, el esquema del algoritmo será mejorado, pasando de hacer un recorrido en sentido horario y antihorario a solo realizar un recorrido en sentido horario.

El cambio realizado en el algoritmo ring signature mejora el tiempo de creación de la firma, además de esta mejora también se altera la línea de aprendizaje del algoritmo, ya que se disminuye la cantidad de operación para realizar la firma.

4. Conceptos generales

En esta sección se explicarán las diferencias entre una firma grupal con un solo firmante y con múltiples firmantes, ya que resulta de vital importancia saber diferenciar lo realizado en el trabajo [1] y lo que se realizará aquí. Entonces como consecuencia de lo anterior, esta sección entrega una introducción sobre la diferencia entre el algoritmo ring signature (firma en anillo) y threshold ring signature (firma en anillo con umbral).

La firma digital presenta una serie de características que la pueden hacer variar según los requerimientos que se establezcan al generar un protocolo de comprobación de un documento que contenga información crítica. Por ejemplo, un documento firmado de forma digital, por un grupo de varios usuarios, puede necesitar que todos los integrantes sean anónimos o también que no se pueda saber quién es el usuario o los usuarios que realizan la firma.

En este proyecto, se necesita garantizar el anonimato para los usuarios firmantes de un grupo determinado, es decir, al existir un documento que va a ser firmado por varios usuarios, no se pueda saber la identidad de los palpitantes de la firma.

4.1. RSA

La firma digital sin garantías de anonimato para los usuarios firmantes, son representadas por el sistema criptográfico asimétrico RSA, el cual debe contar con dos parámetros:

- Llave pública: conocida por todos los usuarios.
- Llave privada: conocida sólo por un usuario.

Estas llaves son utilizadas para garantizar la autenticidad o garantías de conservación de los datos, ya que si un usuario firma un documento con su llave privada los que pueden abrirlo con la llave pública están seguros de que el mensaje pudo ser escrito por una sola persona, por otro lado, si un usuario firma un mensaje con la llave pública se garantiza la privacidad del documento ya que solo lo podrá descifrar el usuario con la llave privada.

4.2. Firma digital de un solo firmante con anonimato

Al existir la necesidad, por parte de una persona, de comprobar que digitalmente realiza una firma en el nombre de un grupo de personas, nace la idea de la firma digital grupal.

Para dar un ejemplo pensemos que existe una entidad, de la cual es innecesario ahondar en su origen, la cual quiere comprobar que de un grupo de cuatro gerentes a lo menos uno de ellos está de acuerdo con un documento. uno de los gerentes firma digitalmente dicho documento, una vez generada la firma, existe una comprobación de esta, no obstante, es absolutamente imposible saber cuál de los gerentes firmó dicho documento, al momento de la verificación de la firma.

Este proceso es abordado, en su totalidad, por el algoritmo ring signature, el cual mediante sus operaciones permite que el usuario firmante de un documento tenga privilegios de anonimato. Existen un sin número de aplicaciones para este algoritmo, como por ejemplo banca electrónica, anonimato en geo ubicación, lotería electrónica, etc.

4.3. Firma digital de múltiples firmantes con anonimato

Threshold rin signature, es un algoritmo de firma grupal de múltiples firmantes la cual comprueba que más de una persona, perteneciente a un grupo, sea capaz de crear una firma en representación de un grupo, además, por ningún motivo se sabe sobre la identidad de los firmantes.

Teniendo en cuanta la firma grupal con un solo firmante, vista anteriormente, se puede llegar a pensar que la solución al nuevo problema sería sencilla, ya que por ejemplo, si se crean al mismo tiempo, dos firmas digitales basadas en ring signature en representación de un firmante, se consigue tanto anonimato como también la comprobación de que dos personas están de acuerdo con el documento firmado, pero pensar que cumpliendo estos requisitos es suficiente es erróneo, ya que fácilmente un atacante puede crear las dos firmas sin que exista forma de comprobar que las dos firmas creadas son de distintos usuarios, además de estas complicaciones existen otras las cuales en los capítulos posteriores serán explicadas.

El algoritmo threshold ring signature tiene la capacidad, gracias a su esquema, de entregar una sola firma, es decir, al existir dos usuarios que quieren firmar un documento, y pertenecen a un grupo de por ejemplo cinco usuarios, la firma que se genera mediante

este algoritmo, demuestra que dos firmantes distintos fueron los que estuvieron participando en la construcción de la firma, por lo demás garantiza el anonimato de los usuarios que actúan como firmantes o proclamadores de autoría sobre un documento.

Threshold ring signature se compone de una serie de piezas, entre ellos están:

- Simulación de sub anillos: se encarga de simular un anillo que será utilizado como nodo en la creación de un súper anillo.
- Partición justa de sub anillos: se encarga de calcular un anillo basado en los que son simulados.
- súper anillo: se encarga de generar un súper anillo mediante los anillos simulados y el anillo de la partición justa.
- Verificación de la firma: se encarga de verificar si el súper anillo fue generado por un grupo de usuarios.

5. Firma en anillo

En esta sección se explicará cómo funciona el algoritmo de firma en anillo basado en criptografía simétrica. Para entender con mayor facilidad es importante conocer aspectos generales sobre criptografía simétrica(AES) y asimétrica (RSA), funciones de hash y permutaciones, y de esta forma continuar con mayor facilidad el análisis de una firma en anillo o ring signatura.

Durante el capítulo expuesto a continuación se debe mantener siempre en cuenta que la firma generada es producida para **un solo firmante** que pertenece a un grupo de n usuarios. Además, la firma es producida de tal manera que cuando se verifica la autenticidad de esta, es imposible conocer la identidad del firmante en cuestión.

El concepto de firma en anillo simple es nombrado así ya que se establece un identificador entre este algoritmo y la variación mejorada de la firma basada en funciones de hash, la cual presentan mejoras en rendimiento al reemplazar los conceptos que en este capítulo pueden parecer irrefutables.

5.1. Definición de firma en anillo

Ring signature es un algoritmo de firma grupal que atiende, con mucha eficacia, 2 requisitos que para este trabajo son esenciales:

- El primero es que de un grupo de personas existe solo una responsable de generar la firma digital.
- El segundo requisito es que garantiza el anonimato de la persona firmante.

El algoritmo de firma en anillo está compuesto por dos procesos:

- el primero es el algoritmo de firma
- el segundo es el algoritmo de verificación.

Estos algoritmos o procesos, tienen determinados valores de entrada y salida. Entre los valores utilizados por esto dos procesos existen valores escogidos de forma aleatoria, los cuales son una serie de X y un valor V también llamado GLUE. No obstante, también existen valores que son computados, estos son la serie de valores Y, los cuales son calculados a partir de los X, es decir, los X e Y son un par inseparable. También existe el valor Z el cual es el resultado de procesar mediante una función combinada el valor V y todos los valores Y pertenecientes a cada X. Es muy importante mencionar que el

algoritmo asume que Z debe ser igual al valor V escogido al azar. Por último, el valor de entrada más importante de todos es el mensaje que será firmado digitalmente.

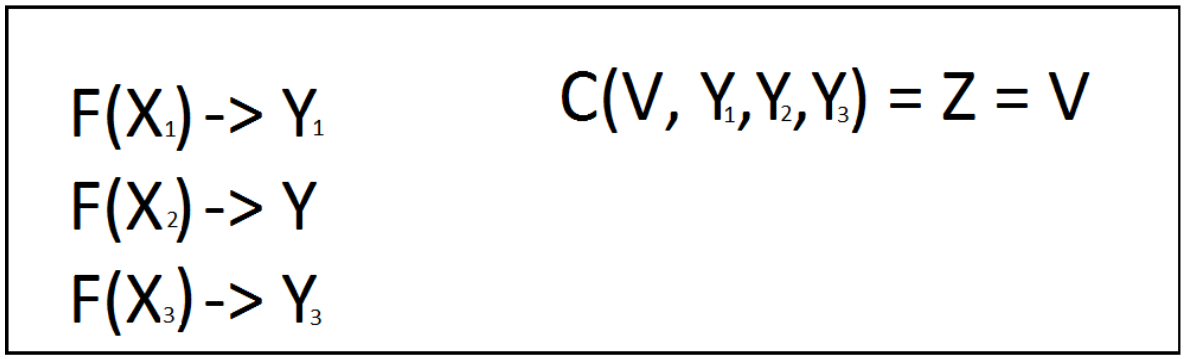


Figura 2: para todos los valores X (escogidos mediante un método de valores Random) son calculados los valores Y, también para la combinación entre V (escogidos mediante un método de valores Random) y todos los valores Y se puede obtener el valor Z, el cual por definición debe ser igual a V.

El algoritmo o **proceso de firma** comienza cuando existe un grupo de usuarios que quieren firmar un mensaje. Entonces el algoritmo recibe como entrada al mensaje que se va a firmar, la llave privada del usuario que firmará el mensaje y las llaves públicas de los otros usuarios participantes. una vez capturados los datos, entrega como salida la firma del mensaje.

El algoritmo o **proceso de verificación** recibe la firma del mensaje, pero para poder verificar la firma se debe contar también con otros valores para comprobar si la firma del mensaje es o no real. Por lo tanto, el proceso debe reunir las llaves públicas de todos los usuarios, los valores escogidos al azar, es decir tanto el valor V como la serie de valores X. una vez reunidos los datos el verificador crea su propia firma del mensaje y la compara con la recibida, si las firmas son la misma, entonces se tiene la certeza que únicamente el grupo de usuario pudo crear la firma y que a lo menos uno de ellos estuvo de acuerdo con el contenido del mensaje.

Como ejemplo de lo explicado anteriormente tenemos que:

Existe un grupo de personas, entre ellos Alice, Bob y Carl, los cuales necesitan entregar un documento muy importante, ese documento requiere que a lo menos uno de los integrantes esté de acuerdo con el contenido para poder ser validado. Alice es el integrante que está de acuerdo con el contenido del documento, entonces quiere enviar la firma, pero es muy importante para Alice que no se sepa que ella fue la que decidió firmar el documento. Entonces, es aquí donde se necesita ring signature. El algoritmo de firma reúne las llaves públicas de Bob y Carl y solicita la llave privada de Alice. Después de esto, el algoritmo de firma se dispone a reunir un valor aleatorio X para Bob y Carl, sucesivamente calcula el valor V. Cuando se reúnen todos los datos el algoritmo se

dispone a calcular el valor incógnito Y correspondiente al usuario firmante, una vez ya calculado aquel valor este se descifra para obtener su respectivo X. Finalmente, una vez calculado el valor incógnito, el algoritmo de firma entrega los valores X de Alice, Bob y Carl, junto al valor V, al algoritmo de verificación, este reúne las llaves públicas de los usuarios y se dispone a comprobar si Z resulta igual a V; el cual es el requisito de comprobación de la firma;

Mientras el algoritmo de firma calculaba el valor de Y incógnito perteneciente a Alice, también se utiliza criptografía simétrica en uno de los tantos procesos. La criptografía simétrica debe contar con una sola llave para encriptar y para des-encriptar, entonces de allí aparece la idea que el valor a utilizar como llave, que debe ser conocido por todos, sea el mensaje, pero dado que el mensaje puede resultar de un tamaño que no es el adecuado para utilizar como llave (se debe utilizar llaves del mismo largo tanto en RSA como en AES), se calcula el **hash del mensaje** para ajustar al tamaño adecuado.

Una vez que el algoritmo de verificación dispone de la firma, que en este caso resulta en una cadena de valores, los cuales son los X de Alice, Bob y Carl, el valor V y el valor Z, entonces se prosigue con la construcción de la firma del mensaje. Con RSA se obtienen los Y de Alice, Bob y Carl y con la combinación que utiliza V y los Y de los usuarios, a su vez cifrando con AES teniendo como llave el Hash del mensaje, es obtenido el valor Z del verificador. Para finalizar si Z = V la firma es real y se entrega la confirmación correspondiente.

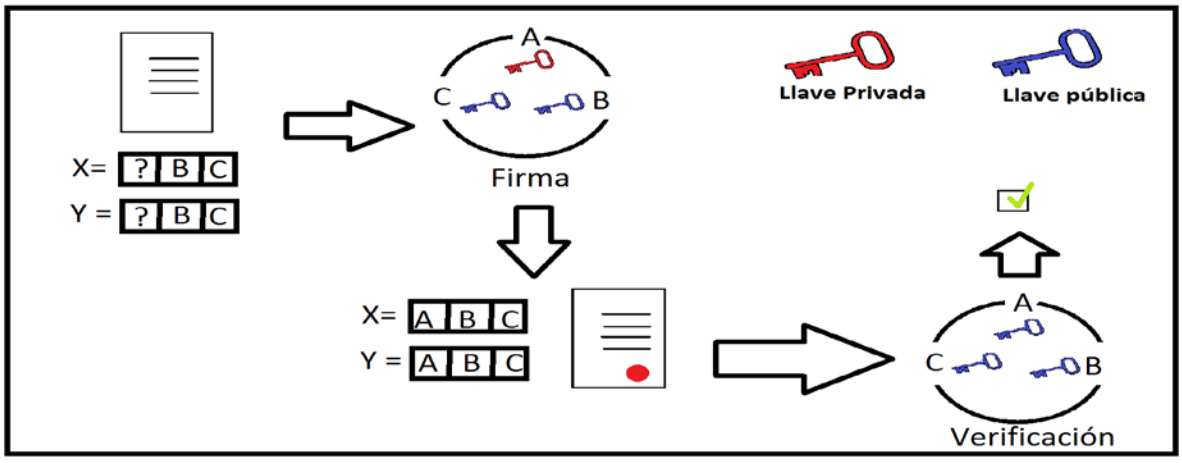


Figura 3: A= Alice, B = Bob y C = Carl. El proceso comienzo con la firma del documento y el cálculo de los valores X e Y de B y C. luego se verifica si la firma es real mediante la reconstrucción del documento firmado por A. si los documentos de entrada y salida son iguales quiere decir que el documento fue firmado por uno de los usuarios, sin conocer la identidad del firmante; la representación debe ser leída desde la esquina superior izquierda.

Así como los números pueden ser combinados de diferentes maneras, las funciones también pueden ser combinadas para formar nuevas funciones, por lo tanto, una función combinada, en el contexto del algoritmo ring signature, es la conformación de una operación a partir de combinar criptografía simétrica (AES) con el operador lógico XOR.

La ecuación que presenta la función combinada en ring signature es la siguiente:

$$z = C_{k,v}(y_1, \ldots, y_n) = E_k\Big(y_n \oplus E_k\Big(y_{n-1} \oplus E_k\big(\cdots \oplus E_k(y_1 \oplus v)\ldots\big)\Big)\Big)$$

Ecuación 1: ecuación de combinación, verificación de la firma. Desde documento [2].

Esta función combinada $C_{k,v}$ (Y_1,…,Y_n) tiene como entrada una llave k, un valor de inicialización v, y una lista arbitraria de valores Y de largo L-bits. Como valor de salida entrega z el cual es una permutación entre {0,1} de largo L-bits, tal que, para algún k, v, existe un valor escogido al azar s que selecciona un número fijo $(Y_i)_{s\ !=\ i}$, es decir el valor s es la posición del usuario firmante en un arreglo de n posiciones (n = número de usuarios, s < n).

Consecutivamente para un índice s, fácilmente se puede verificar que $C_{k,v}$ es una función combinada al reescribir la función en la siguiente:

$$y_s = E_k^{-1}\Big(y_{s+1} \oplus \ldots E_k^{-1}\big(y_n \oplus E_k^{-1}(z)\big)\Big) \oplus E_k\Big(y_{s-1} \oplus \ldots E_k(y_1 \oplus v)\ldots\Big)$$

Ecuación 2: ecuación de combinación, creación de la firma. Recorrido en el sentido horario y antihorario. Desde documento [2].

Esta función demuestra que se puede encontrar un valor incógnito al realizar la permutación entre la operación inversa desde Z hasta el valor Y_{s+1} y la operación normal desde v hasta Y_{s-1}. Dado esto, es aquí en donde se puede notar el recorrido del anillo en sentido horario y antihorario.

La función combinada no solo se compone de criptografía simétrica y la operación lógica XOR, todo lo contrario, para poder obtener los valores Y's es necesario utilizar criptografía asimétrica y para obtener k, es decir, la llave de AES, es necesario utilizar funciones de Hash.

El sistema criptográfico simétrico utilizado en la función combinada es AES, y como llave utiliza el hash del mensaje el cual debe ser de largo L_0-bits. La criptografía simétrica es utilizada como cifrado ideal, es decir, con este tipo de criptografía se puede realizar un recorrido en sentido inverso sobre el anillo, y así, en conjunto con un recorrido normal, encontrar el valor incógnito dentro de la combinación.

El sistema criptográfico asimétrico utilizado en la función combinada es RSA, el cual es utilizado para el cálculo del valor incógnito del usuario firmante; RSA permite cifrar y descifrar tanto con su llave privada como también su respectiva llave pública utilizando su conocido *Trap Door one-way function* ($c = m^e \bmod n$).

Como en la función combinada de ring signature es necesario contar con una serie de valores Y's para cada uno de los usuarios no firmantes, se escoge un X de largo L-bits para cada uno de los participantes del grupo y los Y's son calculados cifrando con la llave pública de los usuarios. A consecuencia de lo anterior, el usuario que sí firma queda con su valor X e Y incognitos.

El valor Y incógnito debe tener una posición dentro del anillo por lo tanto queda demarcado como Y_s, donde s es la posición en el anillo. Cuando este valor es calculado, sólo el usuario firmante puede conseguir su valor X_s al descifrar con su llave privada el valor incognito. Entonces la utilización de RSA en la función de combinación se representa de dos maneras:

- $Y = X^e \bmod n$
- $X = Y^q$ mon n

La e del Trapdoor representa la llave pública, por lo tanto, q representa a la llave privada.

La característica de RSA que permite cifrar o descifrar con la llave privada o pública es de fundamental importancia para garantizar el anonimato del usuario firmante.

El resultado del recorrido inverso que se debe realizar en un anillo debe ser operado, mediante el operador lógico XOR, con el resultado del recorrido hecho de forma normal. Al establecer un recorrido de forma normal (sentido horario) y de forma inversa (sentido anti horario) se debe crear un límite que detiene ambos recorridos. Para obtener el límite, por ejemplo, se puede tener un grupo cuatro firmantes donde la posición del encargado de firmar limita el recorrido, es decir el recorrido inverso es desde el último valor hasta el que sucede al firmante y el recorrido normal es del primero hasta el que antecede al firmante:

- Usuario 1: no firmante.
- Usuario 2: no firmante.
- Usuario 3: firmante.
- Usuario 4: no firmante.

De estos cuatro usuarios el que tiene la posición tres es la que representa al firmante, por lo tanto, para todos el otro existe un Valor X al azar:

- Usuario 1: X_1
- Usuario 2: X_2
- Usuario 3: X_3 = ¿?
- Usuario 4: X_4

Una vez escogidos estos valores al azar tenemos que encontrar sus respectivos Y por lo tanto se opera de la siguiente manera:

- Usuario 1: RSA (X_1, llave pública 1) = Y_1
- Usuario 2: RSA (X_2, llave pública 2) = Y_2
- Usuario 3: X_3 = ¿?
- Usuario 4: RSA (X_4, llave pública 3) = Y_4

Al tener seleccionado estos datos se puede comenzar con los recorridos, del cual se puede encontrar un resultado que revelará el valor incógnito del usuario firmante, es decir:

- $Y_3 = C_{k,\,z}^{-1}(Z, Y_4)$ XOR $C_{k,v}(V, Y_1, Y_2)$

$C_{k,\,v}^{-1}$ representa la combinación en sentido anti horario la cual tiene una llave k para AES y comienza desde el valor z, el cual es el resultado final de la combinación. Por otro lado $C_{k,v}$ representa la combinación en sentido normal el cual tiene una llave k para AES y comienza desde el valor v el cual es el valor de inicialización de la combinación.

En la función combinada se asume que el valor v escogido al azar, debe ser igual al valor z, esto es porque de esta manera se asegurará que el método de verificación al combinar todos los valores Y's obtenga un resultado que pueda ser comprobable; el cual siempre debe resultar V=Z.

5.2. Esquema

En el punto anterior, al entregar las características de la combinación utilizada en ring signature, necesariamente se especifica el funcionamiento del algoritmo de firma de ring signature, pero no solamente existe este algoritmo, si no que tema bien existe el método de verificación, que al igual que el otro cuenta con una combinación la cual realiza un recorrido solo en sentido horario y de esta manera consigue el valor Z el cual por definición debe ser igual a v.

Mediante pseudocódigos se demostrarán los pasos con los que debe contar los algoritmos de firma y verificación de ring signature.

5.2.1. Algoritmo de firma

- Paso 1: Selección del valor GLUE; L es el largo de L-bits del valor.
  - $V = \{0,1\}^L$
- Paso 2: selección de los valores X's para los usuarios no firmantes; s es la posición del firmante, su X queda incógnita.
  - $X_1 = \{0,1\}^L$
  - $X_2 = \{0,1\}^L$
  - …
  - $X_n = \{0,1\}^L$
  - X_s =??
- Paso 3: selección de k; mediante el hash del mensaje de largo L-bits
  - $K = Hash(mensaje)^L$
- Paso 4: cálculo de los Y's para los usuarios no firmantes. (F () representa cifrado RSA con clave pública)
  - $Y_1 = F(X_1)$
  - $Y_2 = F(X_1)$
  - …
  - $Y_n = F(X_1)$
  - Y_s =??
- Paso 5: Cálculo de la combinación en sentido horario, $C_{k,v}(Y_1, …, Y_{s-1})$.(E_k representa cifrado AES con clave k)
  - $R_1 = E_k (Y_1 \text{ XOR } V)$
  - $R_2 = E_k (Y_2 \text{ XOR } R_1)$
  - …
  - $R_{s-1} = E_k (Y_{s-1} \text{XOR } R_{s-2})$

- Paso 6: cálculo de la combinación en sentido anti horario, $C_{k,v}(Y_n, \ldots, Y_{s+1})$. ($E_k^{-1}$ representa descifrado AES con clave k) Z = V
  - $R_n = E_k^{-1}(Z)$
  - $R_{n-1} = E_k^{-1}(Y_n \text{ XOR } R_n)$
  - …
  - $R_{s+1} = E_k^{-1}(Y_{s+1} \text{ XOR } R_{s+2})$
- Paso 7: cálculo de Y_s y X_s;(F^{-1}() representa el descifrado RSA mediante la llave privada del usuario s)
  - $Y_s = R_{s-1} \text{ XOR } R_{s+1}$
  - $X_s = F^{-1}(Y_s)$
- Paso 8: generar tupla que será recibida como entrada por el verificador; la cual puede ser llamada firma.
  - $T = \{X_1, \ldots, X_s, X_n, V\}$

5.2.2. Algoritmo de verificación

- Paso 1: seleccionar elementos de la tupla T.
  - $X_1 = T\{X_1\}$
  - $X_2 = T\{X_2\}$
  - …
  - $X_n = T\{X_n\}$
  - $V = T\{V\}$
- Paso 2: cálculo de los Y's; con las llaves públicas de los usuarios que corresponde cada X's.
  - $Y_1 = F(X_1)$
  - $Y_2 = F(X_1)$
  - …
  - $Y_n = F(X_1)$
- Paso 3: cálculo de Z y comparación con V.
  - $R_1 = E_k(Y_1 \text{ XOR } V)$
  - $R_2 = E_k(Y_2 \text{ XOR } R_1)$
  - …
  - $Z = E_k(Y_n \text{ XOR } R_n)$
- Si Z=V entonces la firma es verdadera, si no, la firma es falsa.

5.3. Implementación de la firma en anillo

Los principales algoritmos de la implementación del trabajo [1]fueron modificados, de modo que cumplieran con el paradigma de orientación a objetos. En el trabajo [1] se muestra el siguiente diagrama que indica los métodos del algoritmo de firma en anillo:

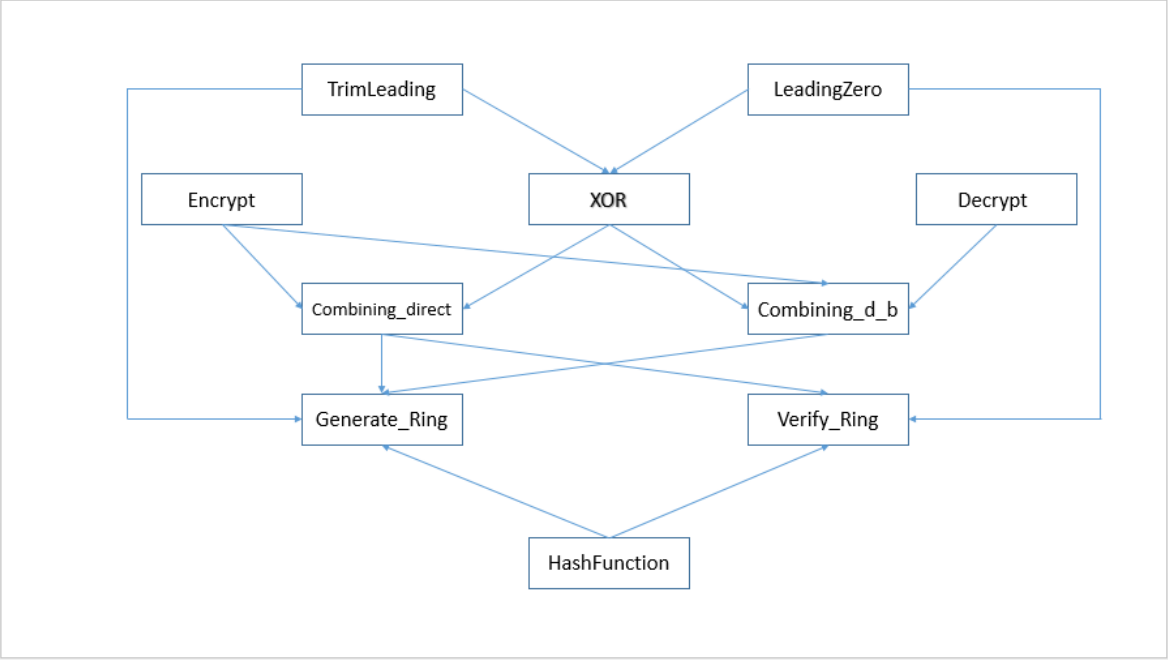


Figura 4: explicación de la estructura del programa, referenciada en el documento [1].

Entonces tomando como base este diagrama y los respectivos métodos implementados, se crea un diagrama de caso de uso, el cual es siguiente:

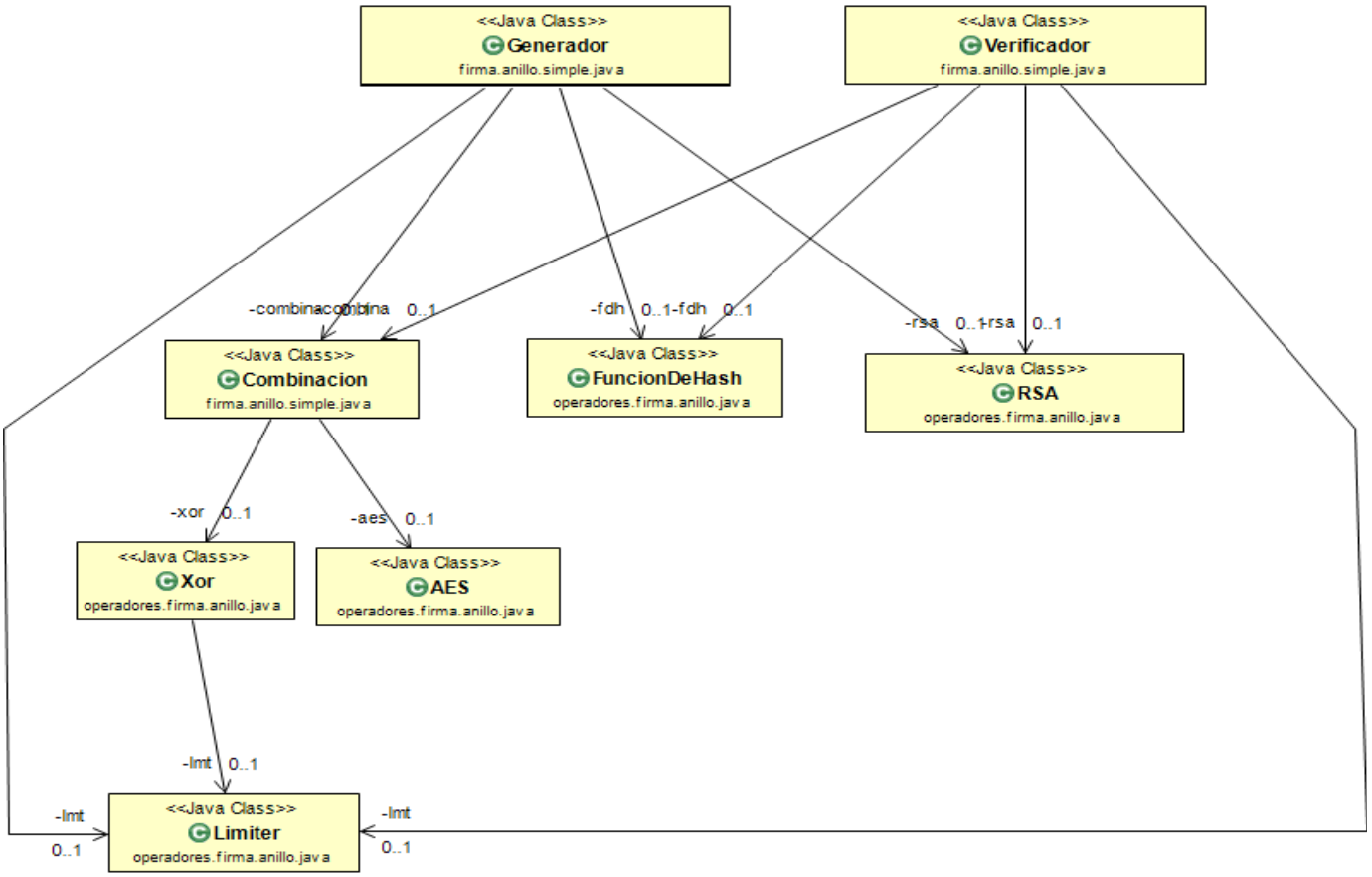


Figura 5: Diagrama de clases del algoritmo de firma en anillo simple. Diagrama automatizado por el plugin de eclipse [5].

5.3.1. Clase combinación

Esta clase representa ambos recorridos que debe realizar el algoritmo para poder reunir el objetivo, por lo tanto, es dividida en dos métodos:

```
public byte[] directa(byte[] key,byte[] v,BigInteger[] y) throws NoSuchAlgorithmException,
                                                                NoSuchPaddingException,
                                                                InvalidKeyException,
                                                                IllegalBlockSizeException,
                                                                BadPaddingException{

    byte[] value = v;


    for(int i =0;i<y.length;i++){
        xor = new Xor(value,y[i].toByteArray());
        value =  xor.getResult();

        value = aes.encripta(key,value); // Valor Z proyección de Ys
    }
    return value;
}
```

Figura 6: Método directa de la clase combinación. Realiza la combinación en sentido horario para la verificación, es decir desde la posición 0 hasta la posición del último usuario que forma parte de un grupo; los usuarios están almacenados en un arreglo.

Este método representa el recorrido en sentido horario a largo de todo el anillo es decir recorre desde la posición cero, pasando por el usuario firmante, para finalizar en la posición n. como parámetros de entrada es necesario contar con la llave AES, es decir, el hash del mensaje, el valor GLUE escogido al azar y todos los valores Y. Como valor de retorno entrega un arreglo de bytes, el cual representa el valor Z del anillo, el cual debe ser igual a V o GLUE. El recorrido indirecto del anillo es necesario para comprobar los resultados válidos calculador por el recorrido indirecto, es decir, con este método podemos saber si el Y incógnito cierra o no el anillo.

El segundo método implementado en la clase representa el recorrido horario y antihorario:

```
public byte[] indirecta(byte[] key,byte[] v, BigInteger[] y,int s) throws InvalidKeyException,
                                                                          NoSuchPaddingException,
                                                                    IllegalBlockSizeException,
                                                                          NoSuchAlgorithmException,
                                                                    BadPaddingException{

    //contenedor temporal de GLUE
    byte[] temp1 = v;
    //segundo contener temporal de GLUE
    byte[] temp2 = v;
    //cantidad de usaurios que pertenecen al anillo
    int r = y.length;


    temp1=aes.desencripta(key,temp1);


    for(int i = r-1; i > s; i--){
        xor = new Xor(temp1,y[i].toByteArray());
        temp1 = xor.getResult();
        temp1=aes.desencripta(key,temp1);
    }
    for(int j=0;j<s;j++){
        xor = new Xor(temp2,y[j].toByteArray());
        temp2 = xor.getResult();
        temp2=aes.encripta(key,temp2);
    }
    xor = new Xor(temp1,temp2);
    return xor.getResult(); //value of ys
}
```

Figura 7: Método indirecta de la clase combinación, realiza el recorrido horario y antihorario en el anillo, cuando se requiere crear una firma grupal.

Como parámetros de entrada se debe contar con las llaves de los usuarios participantes de la firma, el valor GLUE escogido al azar, los valores Y's de los usuarios no firmantes y la posición del usuario firmante. Como valor de retorno entrega un arreglo de bytes que representa el OR-EXCLUSIVO entre el resultado del recorrido horario y antihorario, es decir el Y incógnito del usuario firmante.

5.3.2. Clase Xor

Esta clase representa la operación lógica XOR que se debe realizar en la función de combinación, el algoritmo es:

```
public Xor(byte[] a,byte[] b){


    int len = a.length;
    if (len !=128) { // idéntico que en el método trimLeading
        throw new IllegalStateException("not posible:"+len);
    }
    lmt = new Limiter();
    while (len < b.length) {
        b = lmt.trimLeading(b);
    }

    while (len > b.length) {
        b = lmt.leadingZero(b);
    }
    result = new byte[len];
    for (int i = 0; i < len; i++) {
        result[i] = (byte) (((int) a[i]) ^ ((int) b[i]));
    }
}
```

Figura 8: Constructor de la clase Xor. Encargado de realizar la operación lógica a dos arreglos de bytes.

La clase está dirigida por el constructor, es decir, al instanciar el objeto se realizan las operaciones necesarias para obtener el XOR entre el arreglo de byte a y b, el cual es almacenado en la variable result; esta variable puede ser extraída mediante su método get. Antes de comenzar con la operación lógica de los valores, es necesario conocer el largo de estos, ya que si el valor a es distinto a 128 bytes entonces la operación se detiene y el valor este valor debería ser calculado otra vez; a representa al valor acumulado del recorrido. cuando b es mayor o menor que el largo de a, entonces se deben utilizar los métodos trimLeading y leadingZero para cambiar el largo del arreglo; b representa los valores Y.

5.3.3. Clase limiter

Esta clase es utilizada para limitar los arreglos de bytes cuando estos no respetan el largo requerido en el algoritmo; 128 bytes. La clase no contiene constructor ya que no es necesario inicializar ningún parámetro, por otro lado, estas dividida en dos métodos:

```
public byte[] leadingZero(byte[] b) {
    if(b[0]==0){
        return b;
    }
    result = new byte[b.length + 1];
    for (int i = 0; i < b.length; i++) {
        result[i + 1] = b[i];
    }
    result[0] = 0;
    return result;
}
```

Figura 9: Método leadingZero de la clase limiter.

Cuando un arreglo de bytes es mayor a 128 bytes entonces se utiliza leadingZero, este método cambia el valor más significativo a cero:

```
public byte[] trimLeading(byte[] b) {
    if (b.length % 128 == 0) { // 1024/8 en caso de llaves de 2048 la longitud de los bloques para XOR sería de 256
        return b;
    }
    if (b[0] != 0) {
        throw new IllegalStateException("Attempting to trim " + b[0]);
    }
     result = new byte[b.length - 1];
    for (int i = 0; i < result.length; i++) {
        result[i] = b[i + 1];
    }
    return result;
}
```

Figura 10: Método trimLeading de la clase limiter.

Por otro lado, cuando el arreglo de bytes sobrepasa los 128 bytes, entonces, el método trimLeading traslada todos los bytes en una posición hacia el valor menos significativo y de esta manera el tamaño del arreglo se comprime a 128; esto se lleva a cabo siempre y cuando la posición cero del arreglo es igual a 0.

6. Firma en anillo mejorada

En la sección 3 la firma en anillo que se describe es propuesta para ser utilizada por un grupo de usuarios de los cuales solo uno de ellos desempeña el rol de firmante, por otro lado, en esta sección se presentarán una serie de modificación que se pueden realizar al algoritmo de ring signature con la finalidad de simplificar su esquema y remover la idea de cifrado ideal; el cifrado ideal es presentado por AES, el cual permuta un arreglo de bytes.

La firma de anillo mejorada contiene un elemento, llamado GAP, el cual hace posible el algoritmo threshold ring signature, es decir una firma para múltiples firmantes, no obstante, es presentado en esta sección como modificación del algoritmo presente; este valor se presenta en un anillo como una brecha entre Z y V.

La función combinada de este algoritmo está organizada de forma que incluye el valor GAP y entregar como resultado el valor incógnito del usuario firmante, además de realizar un recorrido en el sentido de la aguja, es decir no es necesario calcular valores inversos para recorrer el anillo en sentido horario y antihorario.

6.1. Modificaciones de la firma en anillo

La idea del algoritmo de firma en anillo mejorado es recorre solamente en sentido horario, comenzando desde un punto conocido del anillo, por ejemplo, el firmante pude poner un índice ique indica al algoritmo el inicio del anillo, es decir la combinación comenzaría con $E_k(Y_i$ XOR v) y así sucesivamente hasta llegar a i-1. Esta leve modificación del recorrido sobre el anillo permite remover la idea de cifrado ideal como una permutación random. Entonces AES puede ser reemplazado por el hash de la concatenación entre el mensaje y el valor de acumulación del recorrido, es decir la operación de la combinación quedaría en resumidas cuentas algo como:

- $E_{k=Hash(m)}$ (x)cambia Por Hash (m, x)

La representación de AES es E_k donde k es la llave a utilizar; la llave es el hash del mensaje.

Todo esto quiere decir que la combinación de la firma en anillo mejorada, ya no debería contener encriptación simétrica, no obstante, se debería contar con una concatenación de valores.

La idea de todo esto es tener un firmante P que calcula valores a lo largo del anillo comenzando desde su propio índice i. Él escoge una semilla ε, para comenzar haciendo hash entre m|| ε, luego a esto realiza un XOR con el valor resultante de $F_{i+1}(X_{i+1})$, para seguir hasta la posición i-1; se puede denotar como:

- $V_{i+1} = H(m, \varepsilon)$
- $V_{i+2} = H(m, V_{i+1} \text{ XOR } Y_{i+1})$
- …
- $V_i = H(m, V_{i-1} \text{ XOR } Y_{i-1})$

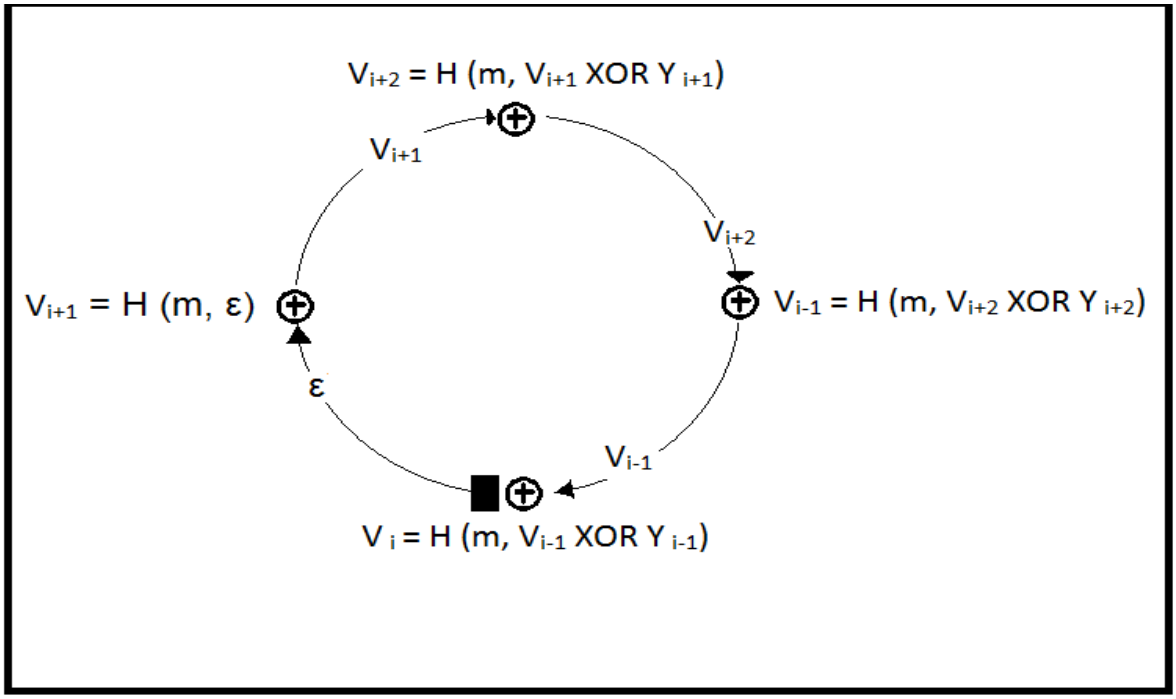


Figura 11: el usuario firmante utiliza el índice i como inicializador, luego se realiza la combinación basada en funciones de HASH; se reemplaza el cifrado AES.

Esta operación nos indica que el valor semilla ε es el XOR entre el GLUE y el Y de la posición anterior en la que nos encontramos, es decir:

- $\varepsilon_{i+1} = V_i \text{ XOR } Y_i$
- $\varepsilon_{i+2} = V_{i+1} \text{ XOR } Y_{i+1}$
- …
- $\varepsilon_i = V_{i-1} \text{ XOR } Y_{i-1}$

mediante esta ecuación también podemos saber que el valor de Y es igual al XOR entre el GLUE de su posición y la semilla de la posición siguiente, es decir:

- V_i XOR $\varepsilon_{i+1} = Y_i$
- V_{i+1} XOR $\varepsilon_{i+2} = Y_{i+1}$
- …
- V_{i-1} XOR $\varepsilon_i = Y_{i-1}$

Con los distintos valores que se pueden conseguir a partir del valor semilla, se obtiene la fórmula para calcular el valor incógnito del usuario firmante; en este caso el valor incógnito seria Y_i. La tupla de valores generada como representación de la firma contiene el valor i, el valor GLUE de la posición i y todos los valores X correspondientes a los usuarios del grupo.

El resultado de todos los procesos realizados en la firma en anillo, prueban que el cifrado ideal (AES) no es superlativo para el funcionamiento de la combinación, por lo tanto, se espera que el algoritmo mejore en sus tiempos de ejecución, tanto en el algoritmo de firma como en el algoritmo de verificación.

6.2. Simulación de un anillo

Otra de las interesantes variantes del esquema ring signature, propuesto en [2], es que la firma se confirma válida solamente cuando se respeta z = v, sin embargo, esto es totalmente arbitrario e incluso innecesario en la seguridad del algoritmo. Esta condición es generada dado el hecho que entre estos valores puede ser incluido un valor GAP, el cual entrega facilidades a la hora de respetar esta simetría del anillo, es decir, si por alguna razón el valor z es un valor distinto a v, al calcular el XOR entre esto valores por ejemplo z XOR v se obtiene como resultado un GAP, el cual al hacer XOR con z se obtiene v, es decir, que a z agregamos lo que le falta para llegar a v. esta modificación propuesta por los autores del documento [2] rompe el paradigma de que z debe ser igual a v, en conclusión todo esta modificación puede ser ejemplificada con el siguiente XOR entre cadena de bits:

Tenemos z y v.

- Z = 01100
- V = 11001

El valor GAP es el XOR entre z y v, con lo cual también se pueden obtener diversos resultados.

- GAP = Z XOR V = 01100 XOR 11001 = 10101
- V = GAP XOR Z = 10101 XOR 01100 = 11001
- Z = V XOR GAP = 11001 XOR 10101 = 01100

El algoritmo de ring signature, dado este cambio de paradigma, presenta una variación al momento de calcular el valor entre el índice n y 1. Ahora que es utilizado el valor GAP, este debe ser incluido cada vez que se recorra el anillo, por lo tanto, cuando el recorrido se encuentre en la última posición del anillo el valor entregado, z, debe ser operado lógicamente con GAP para ajustar todo el resto del recorrido. La siguiente imagen muestra este cometido.

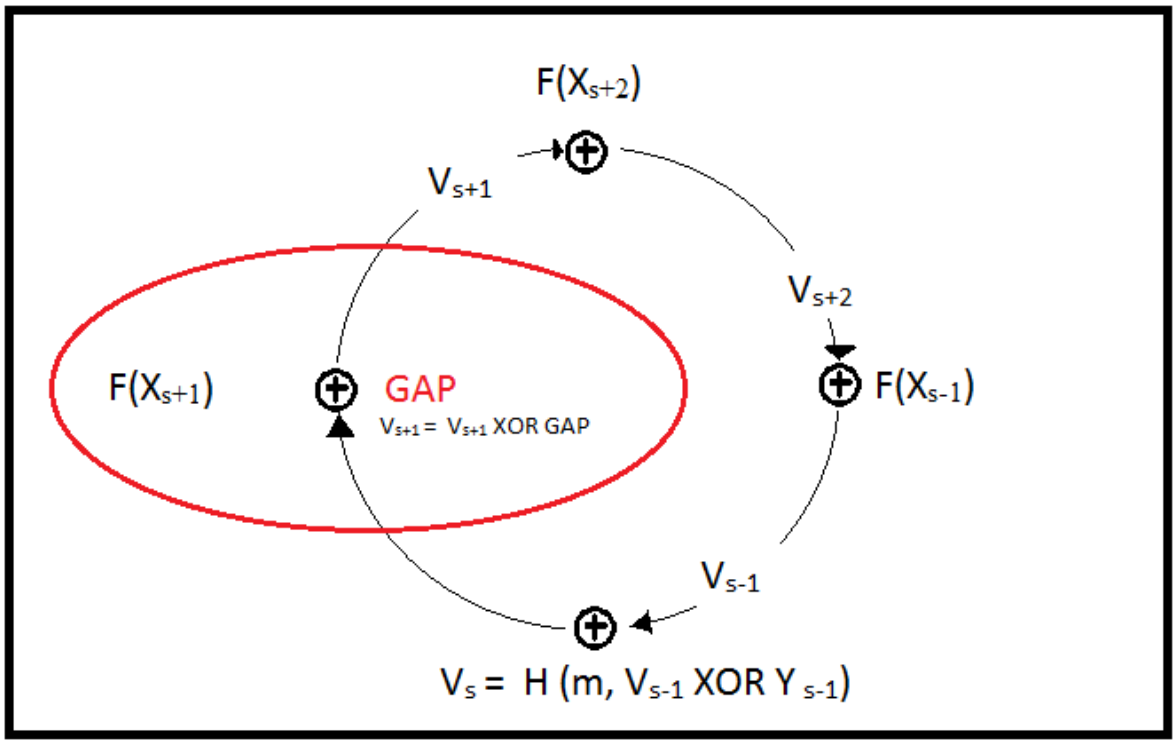


Figura 12: al llegar a la última posición del anillo (encerrada con un círculo), es necesario agregar a la combinación el GAP pre definido.

Esta característica del algoritmo encontrada por los autores del documento [2], permite simular un anillo, ya que ahora el valor GAP puede ser escogido al azar. Entonces cuando son escogidos al azar todos los valores del anillo, es decir, los X de los usuarios, al calcular z se entrega un valor distinto a v, por lo tanto, el XOR entre estos valores permite obtener

el valor GAP que es distando de 0 ya que el valor z es distinto de v. Este resultado permite que utilizar una llave privada para calcular un valor incógnito sea innecesario, es decir un anillo puede ser generado virtualmente sin firmantes.

6.3. Función combinada mejorada

La función combinada vista en los anteriores capítulos, dada todas estas modificaciones en esta subsección se formalizará la nueva fusión combinada, la cual permite simular anillos.

La función combinada se representa como:

- $C_{v,i,m}(GAP, Y_1, ..., Y_n)$

Dado que en la implementación del algoritmo se asume que la posición del gap es la última, no se ingresa como parámetro de la combinación este valor.

La función de combinación está representada por la siguiente operación:

- $H (m, Y_{i-1} \text{ XOR } H (m, Y_{i-2} \text{ XOR } \dots H(m, GAP \text{ XOR } Y_n \text{ XOR} \dots H(m, Y_i \text{ XOR } v)))$

Esta operación indica que se realiza la combinación modificada normalmente hasta el momento que llegamos a la última posición del anillo, y poder ingresar el GAP para que el resultado z cierre con normalidad. esta función de combinación, cómo se puede notar, explica el método de verificación del anillo con GAP al azar.

Para encontrar un valor incógnito Y teniendo un valor GAP escogido al azar, es necesario buscar aleatoriamente los valores Y's para todos los usuarios no firmantes y asignar la posición a encontrar como el índice de inicialización de la combinación, de esta manera se puede crear otra función de combinación que cuenta con la firma de un usuario:

- $Y_s = C^{-1}_{s, Gap,m} (\varepsilon, Y_1, ..., Y_n)$

Esta combinación tiene la siguiente solución:

- $V_{s+1} = H (m, \varepsilon)$
- $V_{s+2} = H (m, Y_{s+1} \text{ XOR } V_{s+1})$
- …
- $V_n = H (m, Y_{n-1} \text{ XOR } V_{n-1}) \text{ XOR GAP}$
- …
- $V_s = H (m, Y_{s-1} \text{ XOR } V_{s-1})$
- $Y_s = V_s \text{ XOR } \varepsilon$

6.4. Esquema

El esquema del anillo mejorado permite crear dos variaciones distintas:

- Anillo simulado
- Anillo ajustado

Esta sub-sección se enfoca en el algoritmo de firma en anillo ajustado, es decir, cuando se conoce de antemano el GAP a utilizar y por lo tanto se desea encontrar un valor incógnito Y para el usuario firmante. Por otro lado, la firma de anillo simulada busca encontrar un valor GAP teniendo como base todos los valores Y, por lo tanto, por obligación el valor z será distinto al valor v; el XOR entre estos valores revela el GAP del anillo.

Es importante hacer esta diferencia ya que más adelante, en threshold ring signature, es de vital importancia generar automáticamente un anillo, y también crear un anillo para los usuarios que van a ser firmante.

6.4.1. Algoritmo generador

- Paso 1: Selección del valor GLUE; L es el largo de L-bits del valor.
  - $V = \{0,1\}^L$
- Paso 2: selección de los valores X's para los usuarios no firmantes; s es la posición del firmante, su X queda incógnita.
  - $X_1 = \{0,1\}^L$
  - $X_2 = \{0,1\}^L$
  - …
  - $X_n = \{0,1\}^L$
  - $X_s = ??$
- Paso 3: cálculo de los Y's para los usuarios no firmantes. (F () representa cifrado RSA con clave pública)
  - $Y_1 = F(X_1)$
  - $Y_2 = F(X_1)$
  - …
  - $Y_n = F(X_1)$
  - $Y_s = ??$
- Paso 4: cálculo del valor semilla ε y el valor GAP.
  - $\varepsilon = \{0,1\}^L$
  - $GAP = \{0,1\}^L$

- Paso 5: Cálculo de la combinación en sentido horario, $C^{-1}_{s, Gap,m}(\varepsilon, Y_1, \ldots, Y_n)$.
  - $V_{s+1} = H(m, \varepsilon)$
  - $V_{s+2} = H(m, Y_{s+1} \text{ XOR } V_{s+1})$
  - …
  - $V_n = H(m, Y_{n-1} \text{ XOR } V_{n-1}) \text{ XOR GAP}$
  - …
  - $V_s = H(m, Y_{s-1} \text{ XOR } V_{s-1})$
- Paso 6: cálculo de Y_s y X_s;(F^{-1}() representa el descifrado RSA mediante la llave privada del usuario s)
  - $Y_s = V_s \text{ XOR } \varepsilon$
  - $X_s = F^{-1}(Y_s)$
- Paso 7: generar tupla que será recibida como entrada por el verificador; la cual puede ser llamada firma.
  - $T = \{X_1, \ldots, X_n, V_0, GAP\}$

6.4.2. Algoritmo verificador

- Paso 1: seleccionar elementos de la tupla T.
  - $X_1 = T\{X_1\}$
  - $X_2 = T\{X_2\}$
  - …
  - $X_n = T\{X_n\}$
  - $V = T\{V\}$
  - $GAP = T\{GAP\}$
  -
- Paso 2: cálculo de los Y's; con las llaves públicas de los usuarios que corresponde cada X's.
  - $Y_1 = F(X_1)$
  - $Y_2 = F(X_1)$
  - …
  - $Y_n = F(X_1)$
- Paso 3: cálculo de Z para encontrar GAP y comparar con el de la tupla.
  - $R_1 = E_k(Y_1 \text{ XOR } V)$
  - $R_2 = E_k(Y_2 \text{ XOR } R_1)$
  - …
  - $Z = E_k(Y_n \text{ XOR } R_n)$
  - $GAP = Z \text{ XOR } V$
  - GAP-tupla = GA

6.5. Implementación de la firma en anillo mejorada

La implementación del ring signature mejorado presenta cambios en las clases de combinación y verificación. Entre otras modificaciones, estas son las más importantes, incluso gracias a estas modificaciones se puede mejorar el rendimiento del algoritmo. El diagrama de clase presenta la siguiente estructura:

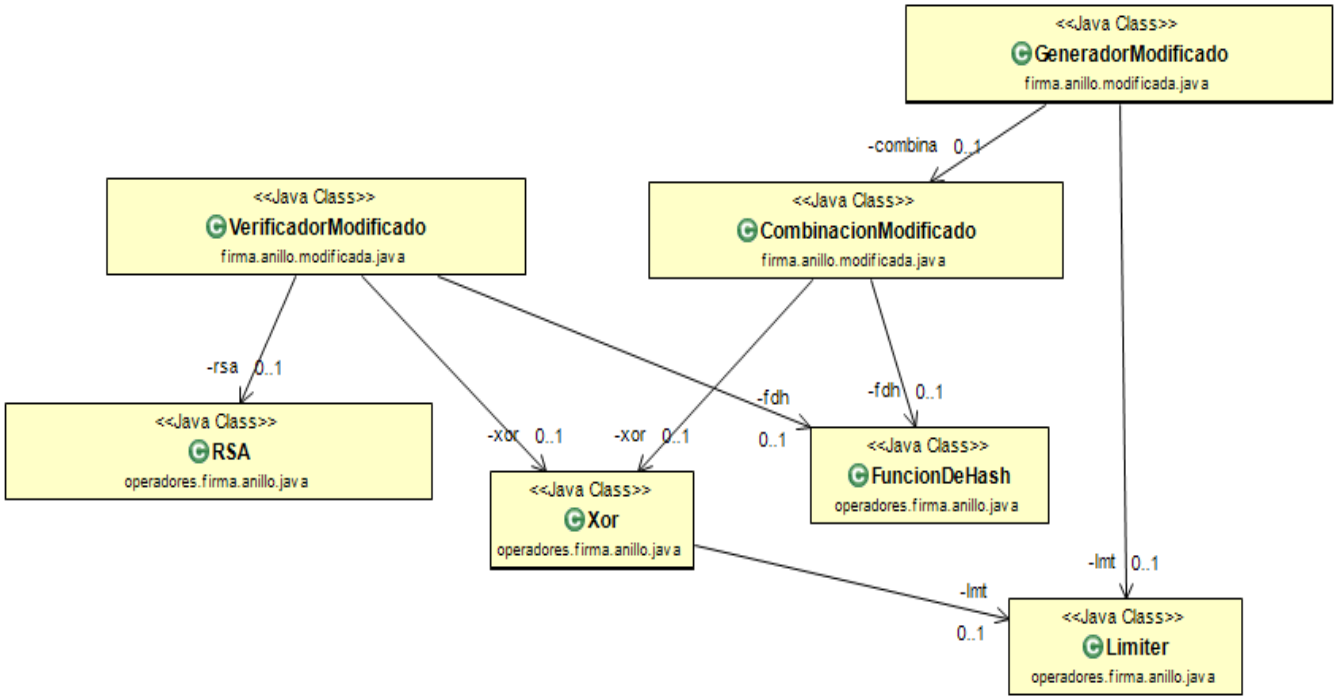


Figura 13: diagrama de clases del algoritmo de firma en anillo mejorado; automatizado en [5].

Como se puede notar la implementación tiene una estructura un tanto menos compleja, debido a que en ella participan menos clases. No obstantes los algoritmos que muestran mayores modificaciones a nivel de código, en la firma de anillo mejorada, son:

```
public void directa(byte[] seed, BigInteger[] y, byte[] mensaje, int s, int users, byte[] gap) throws NoSuchAlgorithmException {
	// TODO Auto-generated method stub
	glue = seed;
	this.seed=seed;
	posFirmante=s;//tremenda falla
	Vs = new byte [users][128];

	//indice para identificar la primera vez que entramos al anillo
	int value =0;
	byte[] c ;//concatenacion
	//concatenacion de la raiz y el mensaje
	for (int i = s+1; true ; i++){
		c = new byte[glue.length + mensaje.length];

		//para el caso inial:
		if (value==0){
			//verificamos si sobrepaza el valor del anillo para comenzar desde 0:
			if (i>=users){
				i=0;
				//si se inicia desde 0 realizamos un xor entre la seed y e gap
				//glue= this.xor(glue, gap);

				//concatenacion entre glue y mensaje:
				System.arraycopy(glue, 0, c, 0, glue.length);
				System.arraycopy(mensaje, 0, c, glue.length, mensaje.length);
				//sacamos el hash de la concatenacion(tamaño 1024, para que el resultado final sea 128)
				fdh= new FuncionDeHash(c,1024);
				xor= new Xor(gap, fdh.getResult());
				Vs[i]=this.xor.getResult();

				//cambiamos glue con el valor de Vs, para ser usado en el siguiente:
				glue=Vs[i];
				value ++;
			}else{
				//si no comenzamos desde 0 no realizar xor entre gap y seed

				//concatenacion entre glue y mensaje:
				System.arraycopy(glue, 0, c, 0, glue.length);
				System.arraycopy(mensaje, 0, c, glue.length, mensaje.length);
				//sacamos el hash de la concatenacion(tamaño 1024, para que el resultado final sea 128)
				fdh= new FuncionDeHash(c,1024);
				Vs[i]=fdh.getResult();

				//cambiamos glue con el valor de Vs, para ser usado en el siguiente:
				glue=Vs[i];
				value ++;
			}



		}else{
			if (i>=users){
				//si se inicia desde 0 realizamos un xor entre la seed y e gap
				i=0;
				//si sobrepasamos el indice entonces comenzamos en 0 y utilizamos el y[n]
				xor= new Xor(glue, y[users-1].toByteArray());
				glue= xor.getResult();

				//concatenacion entre glue y mensaje:
				System.arraycopy(glue, 0, c, 0, glue.length);
				System.arraycopy(mensaje, 0, c, glue.length, mensaje.length);
				//sacamos el hash de la concatenacion(tamaño 1024, para que el resultado final sea 128)
				fdh= new FuncionDeHash(c,1024);
				xor= new Xor(gap, fdh.getResult());
				Vs[i]=xor.getResult();

				//cambiamos glue con el valor de Vs, para ser usado en el siguiente:
				glue=Vs[i];

			}else{

				//si recorremos el anillo con normalidad solo aplicar xor al glue y el indice y[i-1]
				xor= new Xor(glue, y[i-1].toByteArray());
				glue= xor.getResult();

				//concatenacion entre glue y mensaje:
				System.arraycopy(glue, 0, c, 0, glue.length);
				System.arraycopy(mensaje, 0, c, glue.length, mensaje.length);
				//sacamos el hash de la concatenacion(tamaño 1024, para que el resultado final sea 128)
				fdh= new FuncionDeHash(c,1024);
				Vs[i]=fdh.getResult();

				//cambiamos glue con el valor de Vs, para ser usado en el siguiente:
				glue=Vs[i];
			}


		}
		if (i==s)break;

	}//end for
}
```

Figura 14: Método directa de la clase combinación; algoritmo de firma en anillo mejorado.

Por otro lado, en la clase de la combinación mejorada, se presenta el método “directa”, el cual se encarga de recorrer el anillo en sentido horario. Los valores de entrada son el valor semilla, los valores Y’s de los usuarios no firmantes, el mensaje, el índice del usuario firmante (valor inicial del recorrido del anillo), el número de usuarios y el valor GAP. Existe una concatenación de valores, la cual es requerida por el HASH. Por último, las sentencias “if” del código son realizadas para indicar el inicio y final del anillo, es decir, cuando se llega a la última posición se debe cambiar el valor de recorrido a cero, ya que el siguiente valor arrojaría una excepción de valores fuera de rango; las posiciones de los usuarios en el anillo están implementadas mediante un arreglo, es decir, el arreglo de valor Y, el arreglo de las llaves de los usuarios, etc.

Finalmente, la clase de verificación del algoritmo mejorado, presenta las siguientes modificaciones:

```
public VerificadorModificado( byte[][] signature, BigInteger[][] keys,byte[] mensaje) throws NoSuchAlgorithmException{
            System.out.println("");
            System.out.println("Proceso de verificacion mejorado");
            y = new BigInteger[keys.length];
            rsa=new RSA();
            for(int i=0; i<keys.length;i++){
                  y[i]=rsa.encrypt(new BigInteger(signature[i]),keys[i][0],keys[i][1]);
                  //System.out.print("X["+i+"]: "+ new BigInteger(signature[i]).hashCode());
                  //System.out.println(" -> Y["+i+"]: "+ y[i].hashCode());
            }

            glue = signature[keys.length];
            glueComparator = signature[keys.length];
            //SYSO---------------------------------->
            //System.out.println("Glue: "+ new BigInteger(glue).hashCode());
            System.out.println("");
            gap = signature[keys.length+1];

            byte[] c ;//concatenacion
            for (int i = 0; i<keys.length ; i++){
                  c = new byte[glue.length + mensaje.length];
                  //iniciamos glue con el valor de Vs e Ys anterior:
                  xor = new Xor(glue, y[i].toByteArray() );
                  glue= xor.getResult();

                  //concatenacion entre glue y mensaje:
                  System.arraycopy(glue, 0, c, 0, glue.length);
                  System.arraycopy(mensaje, 0, c, glue.length, mensaje.length);

                  //sacamos el hash de la concatenacion(tamaño 1024, para que el resultado final sea 128)
                  fdh = new FuncionDeHash(c,1024);
                  z=fdh.getResult();

                  glue=z;

            }

      }
```

Figura 15: Constructor de la clase VerificadorModificado.

El constructor de esta clase recibe como entrada un arreglo de bytes el cual representa la tupla de los elementos, también recibe las llaves públicas de los usuarios, y el mensaje. Este constructor se encarga de crear la variable z, de la cual dependerá la comprobación de la firma, mediante el método:

```
public boolean signatureIsTrue() {
    xor = new Xor(z , glueComparator );
    byte[] gapToCompare = xor.getResult();
    //System.out.println(" -> gap: "+new BigInteger(gapToCompare).hashCode());
    if(new BigInteger(gapToCompare).equals(new BigInteger(gap))){
        return true ;
    }
    return false;//xor.getResult();
}
```

Figura 16: método para confirmar la operación de verificación de la firma.

Este método de la clase de verificación del algoritmo modificado, calcula el GAP de los elementos de la tupla entregada, si este GAP es igual al GAP que contiene la tupla, esto quiere decir que la firma es verdadera, en caso contrario se retorna falso; Z XOR GLUE = GAP.

6.6. Diferencia con la firma en anillo simple

Como se ha mencionado a lo largo de este capítulo, el algoritmo ring signature, o también traducido como firma en anillo, al presentar variantes que facilitan su comprensión, también se espera que las mejoras ayuden al algoritmo en su tiempo de ejecución. No obstante, comparar los tiempos de ejecución del algoritmo de firma en anillo simple y el modificado, no es la única comparación que se puede establecer, por lo tanto, en esta subsección se pretende comparar el tiempo de ejecución del algoritmo, así como también el peso resultante de las firmas generadas, etc.

6.6.1. Tiempo de ejecución del algoritmo generador

La primera comparación establecida es sobre el tiempo de ejecución del algoritmo de generación de la firma, variando el número de usuarios, manteniendo siempre la misma posición del firmante y un mensaje de mismo peso en bytes.

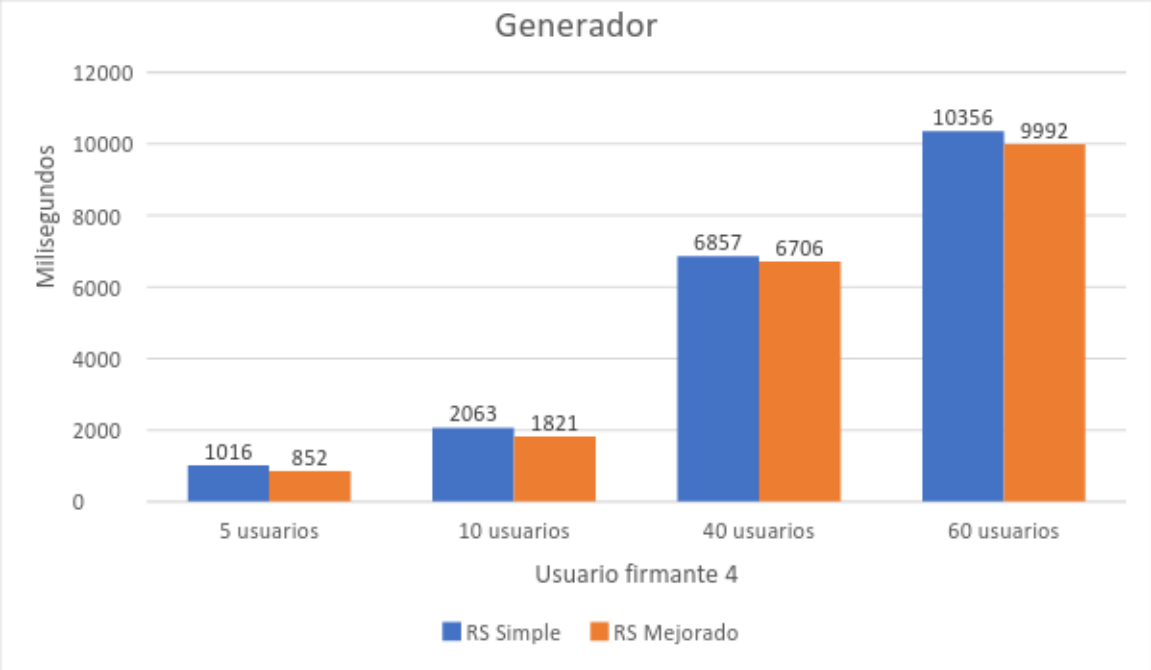


Tabla 1: pruebas de ejecución del algoritmo de firma en anillo; Alternando cantidad de usuarios.

En la segunda comparación se establece una cantidad fija de usuarios para una firma en anillo, esta vez se alterna la posición del firmante dentro de un arreglo, con un mensaje del mismo tamaño para todas las pruebas.

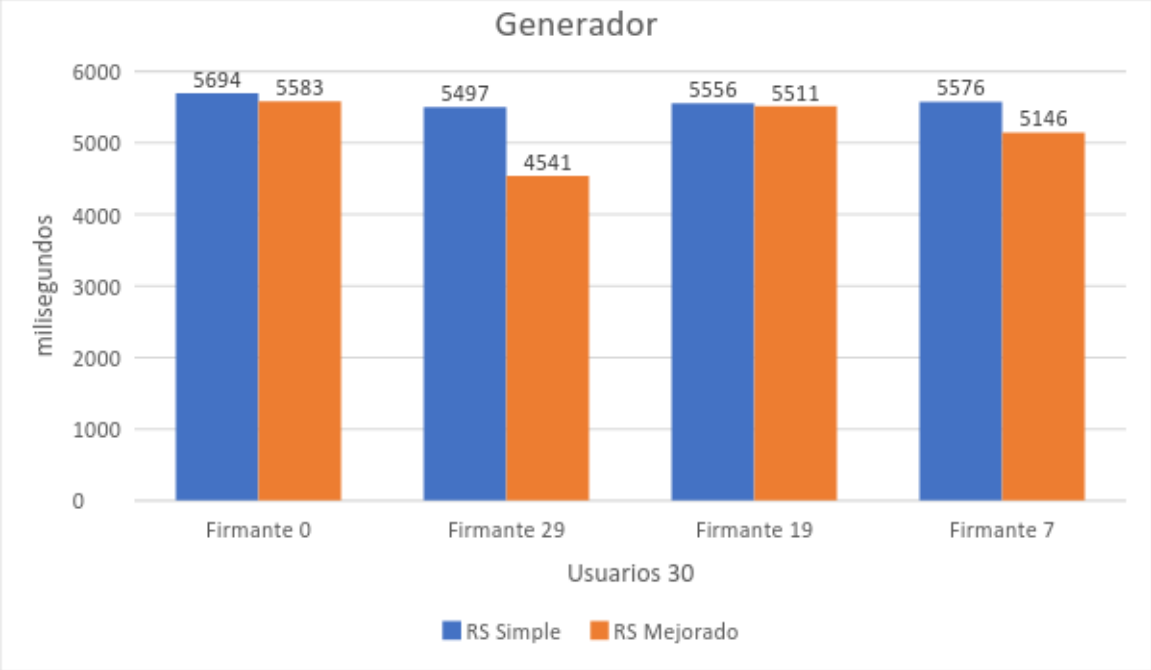


Tabla 2: pruebas de ejecución del algoritmo de firma en anillo; Alternando posición del firmante.

En la tabla 1 y 2 se generan 5 pruebas en cada una de los valores establecidos, de los cuales se consigue un promedio de tiempos de ejecución del algoritmo. Por otro lado, los resultados esperados en la tabla 1 es que siempre el algoritmo de firma en anillo simple tenga un mayor tiempo de ejecución que la mejorada, también este patrón debe ser igual en la tabla 2.

En la tabla 1 las diferencias que se pueden apreciar son relativamente pequeñas, es decir, en la prueba con diez usuarios existe una variación del 16.14% por ciento de mejora, en la prueba 2 un 11.73%, en la prueba 3 un 2.2% y en la última prueba un 3.51%.

En la tabla 2 la diferencia entre el algoritmo simple y el mejorado tienen una diferencia menor, ya que en la prueba 1 es de 1.94%, en la prueba 2 de un 17.39%, en la prueba 3 un 0.8% y en la última prueba un 7.71%. estos resultados indican que el tiempo de ejecución no está estrechado con la cantidad de usuarios, si no que con la posición del usuario en el arreglo en que son almacenados

6.6.2. Tiempo de ejecución del algoritmo verificador

La cuarta comparación es sobre el tiempo de ejecución del algoritmo de verificación,

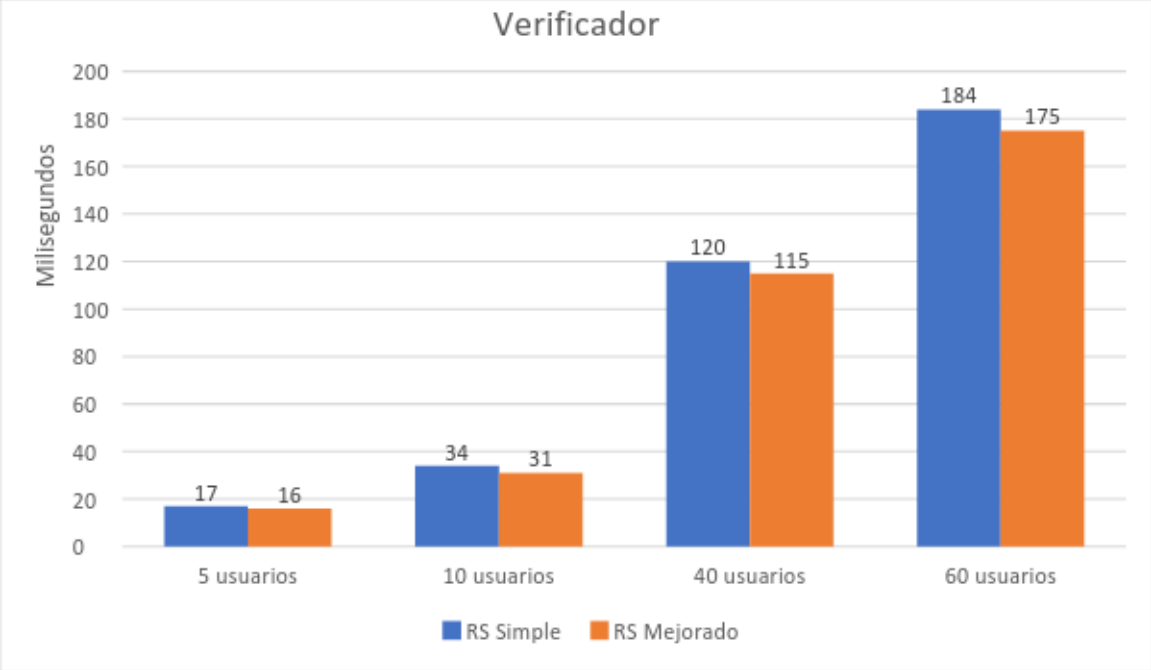


Tabla 3: comparación de verificación, alternando la cantidad de usuarios.

variando el número de usuarios, manteniendo siempre la misma posición del firmante y manteniendo el tamaño del mensaje

De un total de 5 pruebas fue obtenido el promedio de tiempos para cada uno de los casos. En cada uno de los casos se cuenta con distintas cantidades de usuarios.

En cada uno de los casos se obtuvo el resultado esperado, el cual consiste en que el tiempo de verificación de ring signature simple tomará más tiempo. Esto es debido a que el algoritmo ring signature mejorado no utiliza AES en su implementación, la cual es reemplazada por el Hash de la concatenación entre el valor semilla y el mensaje.

Las diferencias entre los tiempos de ejecución, de ambos algoritmos, son muy pequeñas, por lo tanto, el tiempo total de ejecución de los algoritmos no serán afectados por sus algoritmos verificadores.

7. Grupos ad-hoc

En la sección 5 se establece relación entre los grupos ad-hoc y la firma digital múltiple, la cual contiene una estructura booleana que hace que los usuarios firmantes puedan estar ocultos. También se hace referencia a las diferencias entre ring signatura y el algoritmo objetivo, por lo que en una tabla comparativa se especifican.

En esta sección se formalizará la definición del caso particular de ring signature llamado threshold ring signature. La estructura tradicional con umbral "t-fuera-de-n" podría ser vista como un caso especial de la estructura de acceso general en la cual puede ser especificado un criterio de mínima colaboración, de decir que existe un anillo que será firmado por múltiples usuarios de un grupo. Entonces aquí se define formalmente el grupo ad-hoc y los requerimientos de seguridad correspondientes para su firma.

7.1. ¿Qué son los grupos ad-hoc?

La firma en anillo con umbral pretende que de un grupo de usuarios exista la posibilidad que más de uno de ellos pueda firmar digitalmente un documento, es decir, implementar un anillo de múltiples usuarios firmantes. No obstante, la estructura de firma en anillo solo permite que uno de los usuarios pertenecientes al grupo cumpla dicho rol.

Entonces con esta premisa se busca crear una adaptación del esquema para que múltiples usuarios puedan ocupar el rol de firmantes. Los autores del documento académico [2] prueban que adaptar la estructura de ring signature a múltiples usuarios es posible.

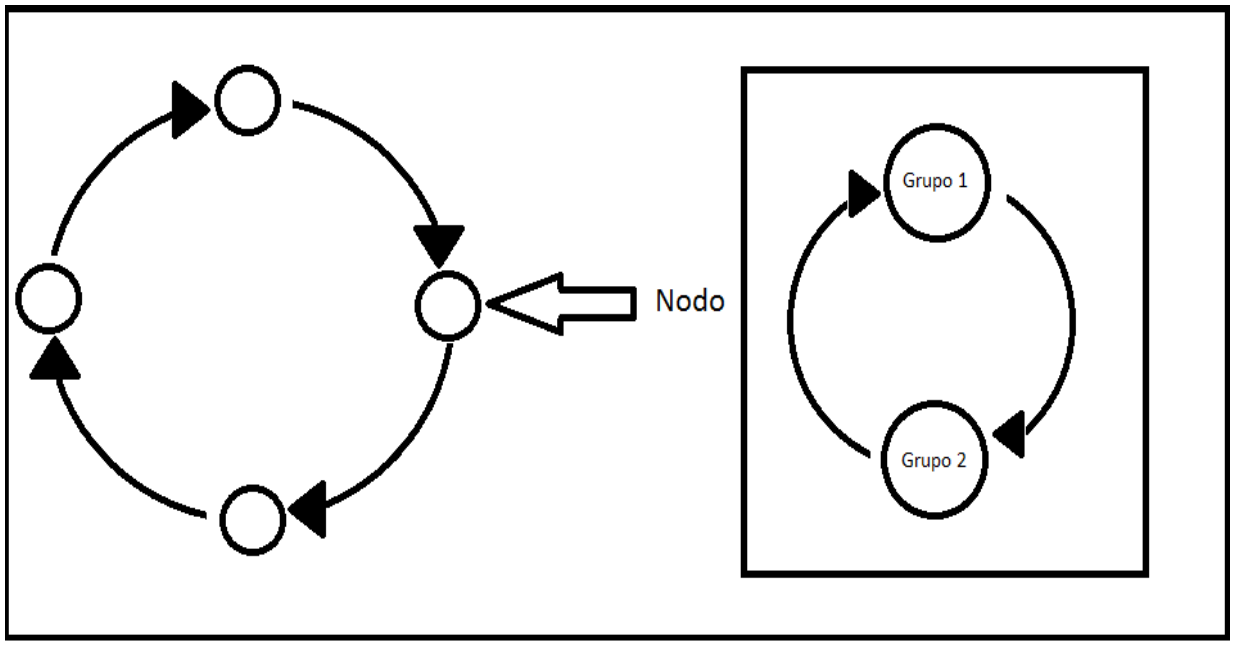


Figura 17: a la izquierda de la imagen se encuentra ubicado un anillo, el cual se compone por nodos los cuales representan a los usuarios del sistema. Por otro lado, en el lado derecho se indica que el nodo, en threshold ring signature está compuesto por grupos de usuarios.

En esta adaptación de ring signature para múltiples anillos se generaliza el concepto del nodo, como se muestra en la fig.7 en ring signature un nodo está compuesto por tan solo un usuario, pero en threshold ring signature este nodo está compuesto por grupos de usuarios, es decir, un anillo se compone de sub-anillos.

La cantidad de usuarios de los grupos que componen al nodo pueden ser tantos como sea posible, pero en la implementación propuesta en este proyecto la cantidad de usuarios es siempre la misma, es decir, los usuarios que componen el grupo son un usuario firmante y el resto que no es firmante, entonces si el grupo se compone de 4 usuarios 3 de ellos quisieran firmar los grupos contendrían 2 usuarios es decir el firmante y el resto que no firma el cual sería solo 1.

Los grupos de usuarios que componen el nodo, son llamados grupos ad-hoc. Esto es porque son hechos para poder ocultar la identidad de los firmantes. La identidad de los firmantes de un nodo queda anónima debido a que el súper anillo se compone muchos nodos entonces el nodo de los firmantes es uno entre muchos.

entonces el origen de la composición de los grupos ad-hoc es debido a que los anillos producidos con ring signature solo sirven para un usuario, por lo tanto, se crea un nodo con sub anillos que representan a distintos firmantes. Por otro lado, cuando se generan anillos para distintos firmantes es necesario comprobar que son hechos por distintos usuarios, por lo tanto, los grupos ad-hoc deben contener solo a uno de los firmantes, por

lo que quedaría como intersección de los grupos solamente los usuarios no firmantes del grupo de usuarios. No obstante, si los grupos ad-hoc quedan faltos de uno de los usuarios se evidencia cual es el firmante, por lo tanto, para tapar u ocultar al nodo de los reales usuarios firmante son creados varios nodos con la misma cantidad de grupos, compuestos por la misma cantidad de usuarios, pero con distintos usuarios como firmantes.

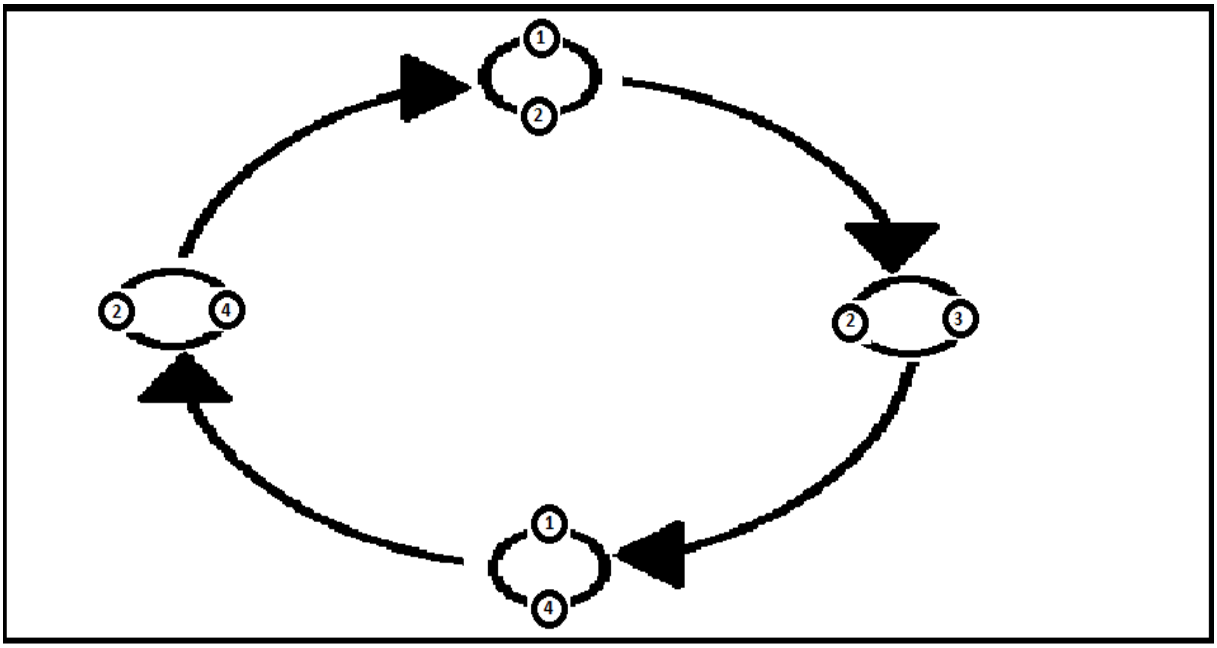


Figura 18: representación de un súper anillo de la firma en anillo con umbral. Existen cuatro nodos, en los cuales hay 2 grupos ad-hoc (sub anillos), en los cuales firman distintos usuarios, es decir, {1,2}, {2,3}, {1,4}, {2,4}.

En este trabajo, como se menciona anteriormente, no se aplican tantos grupos como es posible, si no que un método que crea anonimato con una menor cantidad de usuarios. Entonces el método para crear los grupos es una combinación según los usuarios del grupo y su correspondiente número de firmantes, de esta manera la cantidad de grupos ad-hoc se reduce drásticamente. La combinación que resuelve la cantidad de nodos del súper anillo es:

- $P = N! / T!\ (N-T)!$

Debido a que no importa el orden de los usuarios se tiene que el número de particiones (también nodos o grupos ad-hoc) es P, donde N es el número de usuarios y T el número de firmantes del grupo.

Para formalizar los grupos ad-hoc, tenemos que Σ es una lista de n usuarios, acompañado por una lista de subconjuntos S_j, también llamado los subconjuntos aceptables. Los grupos ad-hoc deben estar definidos para la firma y la verificación, por lo tanto, se consideran dos algoritmos distintos:

- Algoritmo de firma ad-hoc: al entrar un mensaje m, una especificación de un grupo ad-hoc Σ (n usuarios con el mismo número de subconjuntos aceptables), entrega un grupo ad-hoc para threshold ring signature σ del mensaje m.
- Algoritmo de verificación ad-hoc: al entrar un mensaje m y una firma σ, se entrega 'true' si σ es una firma en anillo ad-hoc válida para m, relativa a la lista de subconjuntos aceptables de Σ o 'false' en otro caso.

Observación: el algoritmo encargado de calcular los grupos ad-hoc debe ser de carácter determinístico para que no existan problemas en el posicionamiento de los usuarios dentro del sistema de firma digital.

Por último, tomando como ejemplo un grupo de 4 usuarios y dos de ellos quieren colaborar para firmar un mensaje, se pueden construir los siguientes grupos ad-hoc; los cuales deben ser transformados en anillos mediante ring signature:

- Nodo 1:
  - Sub-anillo 1: usuario 1, usuario 2, usuario 3.
  - Sub-anillo 2: usuario 1, usuario 2, usuario 4.
- Nodo 2:
  - Sub-anillo 1: usuario 1, usuario 2, usuario 3.
  - Sub-anillo 2: usuario 1, usuario 3 usuario 4.
- Nodo 3:
  - Sub-anillo 1: usuario 2, usuario 3, usuario 4.
  - Sub-anillo 2: usuario 1, usuario 2, usuario 3.
- Nodo 4:
  - Sub-anillo 1: usuario 2, usuario 3, usuario 4.
  - Sub-anillo 2: usuario 1, usuario 2, usuario 3.

Se establece que los nodos están compuestos por dos sub anillos ya que los firmantes que quieren colaborar son dos. Los grupos ad-hoc deben componer a los subanillos.

7.2. Estructura de los grupos ad-hoc

En esta sub sección se dará a conocer en detalle la estructura de threshold ring signature; los grupos ad-hoc. Estos grupos son incluidos en la implementación como un generador de particiones, el cual por motivos de tiempo de desarrollo realiza una lectura de un documento de texto, el cual contiene los datos necesarios de los grupos ad-hoc que son necesarios en la construcción de los grupos para una firma múltiple.

Threshold ring signature, claramente es un caso especial de una firma en anillo ad-hoc; fuera de los n miembros del grupo de usuarios firmantes, cada subgrupo contiene t miembros que forman parte de un grupo aceptable.

Los nodos del súper anillo son una concatenación de la firma los subanillos, de entre los cuales existe uno con el que se puede demostrar que existe una cooperación real entre diferentes usuarios firmante, los otros sub anillos conformados por los grupos ad-hoc deben ser simulados, por ende, es aquí en donde se usan las dos diferentes partes del algoritmo ring signature mejorado; expuesto en la sección 4. Utilizando un sistema de meta anillo se puede comprobar que uno de los grupos del esquema ad-hoc ha sido creado únicamente con las llaves privadas de los usuarios firmantes.

En la estructura de threshold ring signature se puede probar que, por ejemplo, el usuario P1 ha firmado m, OR P2 AND P3 AND P4, OR P1 AND P2 también. Esto es el resultado de construir los subanillos del nodo a partir de grupos ad-hoc pre establecidos.

El documento de texto, que cumple el papel de entregar los grupos ad-hoc sigue el siguiente estándar:

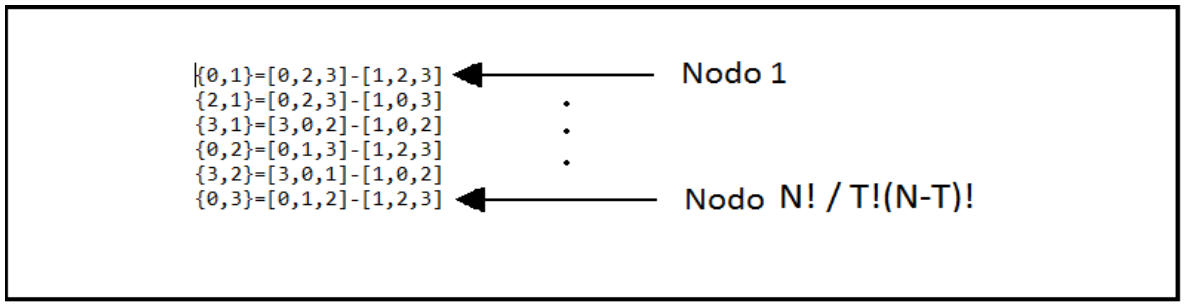


Figura 19: documento a leer para crear los grupos ad-hoc.

El grupo de usuarios son 4 y están almacenado en un arreglo por lo que se dice que los usuarios son {0, 1, 2, 3}. Cada una de las líneas del archivo representa un nodo por lo tanto para los 4 y entre ellos 2 firmante existen 6 particiones para ocultar la identidad del firmante. En el primero de los nodos se realiza una lectura que el usuario {0,1} son los firmantes, en donde 0 está en la partición [0,2,3] el usuario 1 en la partición [1,2,3]

7.3. Firma en anillo con umbral como abstracción de la firma en anillo

Parte del anillo	Firma en anillo	Firma con umbral
Anillo	-Se compone de nodos. -se encarga de entregar el valor Y del usuario firmante.	-Es un súper anillo que se compone de nodos. -Este súper anillo es utilizado para calcular el valor faltante en los grupos ad-hoc que contienen a los usuarios firmantes (concatenación de gaps).
Nodo	-Se compone de un único usuario. - Contiene asociado un único valor x e y. -Son tanto nodos como usuarios componen el grupo.	-Se compone de grupos ad-hoc de usuarios, por lo tanto, se calculan sub anillos para cada nodo del anillo. -La cantidad de sub anillos es igual al número de usuarios firmantes. -La cantidad de nodos está dada según la combinación en la que no importa el orden.
GAP	-El valor gap está predefinido para que el anillo no sea simétrico cuando el firmante cierre el anillo con su llave privada. - Es uno para todo el anillo.	-En los nodos simulados el GAP es calculado aleatoriamente. -En la partición justa el gap es pre definido para que los anillos no sean simétricos cuando el firmante del grupo ad-hoc cierre el anillo con su llave privada. -Es uno para cada uno de los nodos del súper anillo. -la concatenación del GAP de los subanillos compañeros de los grupos ad-hoc son utilizados por el súper anillo, para encontrar la concatenación de la partición justa.

Figura 20: tabla de comparación entre la estructura de un anillo de ring signature y threshold ring signature.

8. Firma en anillo con umbral

La firma en anillo con umbral, también conocida como threshold ring signature, diseñada por los autores [2] establece que existen dos tipos de nodos, los contienen grupos ad-hoc de usuarios no firmantes y los que contienen grupos ad-hoc con los usuarios firmantes. En esta sección se entregan la definición de cada uno de estos nodos. También se formalizará el esquema de threshold ring signature. Por último, se dará a conocer la implementación del algoritmo, la cual tiene diferencias y similitudes con los algoritmos expuestos con anterioridad, dado que es una generalización para grupos de usuarios componentes de nodos.

El anillo con umbral está compuesto por las siguientes partes:

- Súper anillo: anillo creado a partir de los valores del meta anillo.
- Meta anillo o nodo: son los nodos del súper anillo, y está compuesto por la concatenación de los GAP de los sub anillo.
- Sub anillo: son los usuarios del meta anillo, anillos creados mediante ring signature, el cual debe estar compuestos por los usuarios propuestos en los grupos ad-hoc.

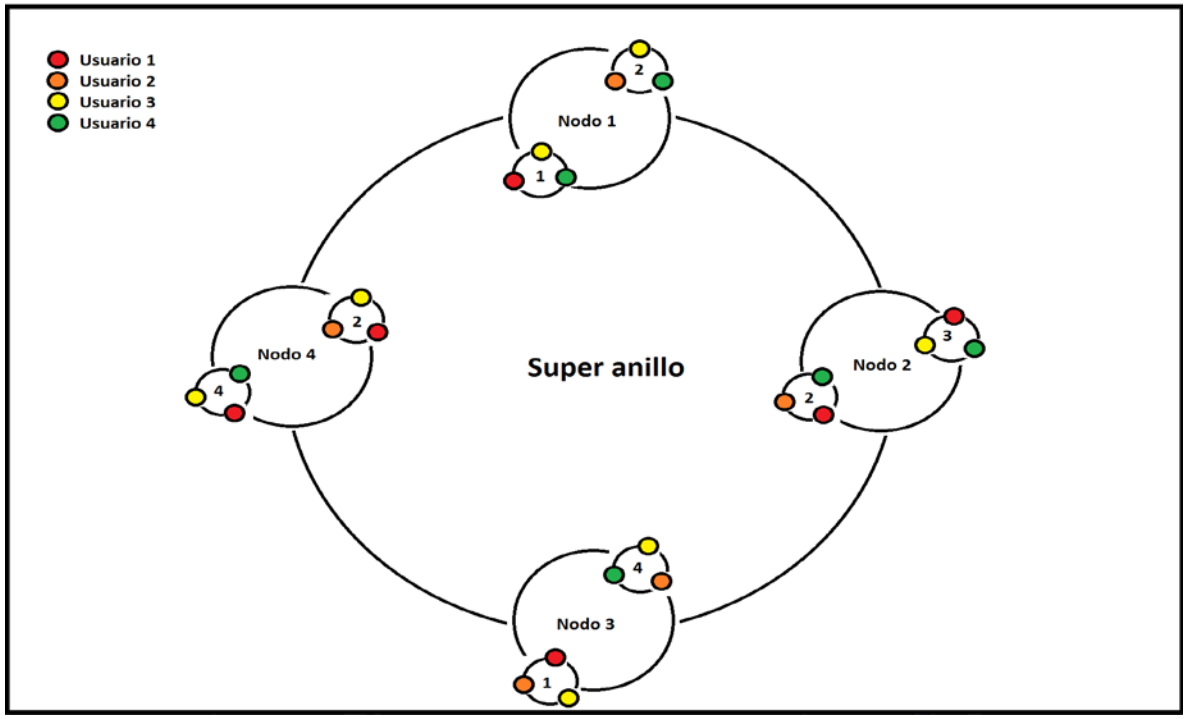


Figura 21: El súper anillo está compuesto por nodos y los nodos están compuestos por sub anillos que a la vez están compuestos por grupos ad-hoc de los usuarios de un grupo; en los subanillos se puede ver un número el cual representa al usuario firmante del grupo, el cual es el color que no se encuentra en el vecino.

Si en un grupo de r usuarios existen 2 usuarios que quieren comprobar su participación en una firma grupal, tenemos que:

- Dividir el grupo muchas veces, tal que:
  - Siempre existe una división en las cuales 2 usuarios cualquiera están en 2 diferentes sub-anillos.
- Todos estos grupos creados son utilizados como nodos de un súper-anillo.
- El súper-anillo prueba que al menos una partición ha sido resuelta, es decir, 2 sub-anillos han sido individualmente resueltos.
- Las otras particiones son simuladas como una correcta firma en anillo.

Observación: en el capítulo anterior se hace referencia en la múltiple dicción de los usuarios del grupo, por lo que de múltiples divisiones se considera más eficiente una división controlada por la combinación P = N!/T! (N-T)!.

8.1. Particiones

La partición justa es el grupo ad-hoc que contiene a los usuarios firmantes de un grupo de usuarios. Tomando como base la estructura de ring signature, los autores de [2] demuestran matemáticamente que es posible crear una firma múltiple para un grupo de usuarios. Los usuarios que quieren demostrar colaboración crean un nodo que solamente ellos mediante una llave privada serán capaces de construir. Este grupo o nodo particular es llamado en el mismo documento 'partición justa'. La partición justa presenta una serie de características que permiten que un súper anillo pueda ocultar cuál es la partición justa, es decir, el nodo en que los firmantes utilizaron su llave privada. Para llevar a cabo la construcción del súper anillo es necesario respetar las siguientes reglas:

- Entre los grupos ad-hoc siempre debe existir un grupo que contenga a los usuarios firmantes.
- Los usuarios firmantes de los grupos ad-hoc deben estar contenidos en un subanillo.
- Los sub anillo deben tener como intersección a los usuarios que no son firmantes; de manera que si son intersección los usuarios firmantes los subanillos serán iguales.

Las particiones simuladas son los grupos ad-hoc que contienen como firmante a los usuarios que no son firmantes. Estos grupos son simulados ya que sus firmantes al no contar con la llave privada la única forma de calcular un anillo es que este no cierre con un valor GAP igual a 0 es decir entre Z y V existirá una brecha que es calculada solamente con las llaves públicas de los usuarios; la partición justa como contiene llave privada de los usuarios firmantes, se calcula con un valor GAP pre definidos para que el anillo tenga una brecha distinta a 0 un anillo que solo pudo ser calculado con la llave privada.

En esta implementación los grupos ad-hoc, también llamados nodos, contienen a todos los usuarios. Por lo tanto, la utilización de los valore X e Y en cada uno de los subanillos es distinta para cada uno de los nodos, por ejemplo:

- Usuarios: 1,2,3,4
- Firmantes: 2,3
- Grupos ad-hoc: {1,2}, {1,3}, {1,4}, {2,3}, {2,4}, {3,4}.
    - Nodo {1,2}
        - Sub anillo 1: [1,3,4]
        - Sub anillo 2: [2,3,4]
    - Nodo {1,3}
        - Sub anillo 1: [1,2,4]
        - Sub anillo 2: [2,3,4]
    - Nodo {1,4}
        - Sub anillo 1: [1,2,3]
        - Sub anillo 2: [2,3,4]
    - Nodo {2,3}
        - Sub anillo 1: [1,2,4]
        - Sub anillo 2: [1,3,4]
    - Nodo {2,4}
        - Sub anillo 1: [1,2,4]
        - Sub anillo 2: [2,3,4]
    - Nodo {2,4}
        - Sub anillo 1: [1,2,3]
        - Sub anillo 2: [1,3,4]

Entonces en cada uno del nodo los usuarios 1, 2, 3 y 4 tienen asignados distintos valores X e Y, por lo tanto, estos valores ahora son una colección, es decir en la firma en anillo existe X y en la firma con umbral existe una lista de X para cada uno de los nodos, es te caso 6 distintos X y 6 distintos Y.

Observación: es imposible utilizar los mismos valores X e Y en cada uno de los nodos ya que existirá como incógnito los valores de los usuarios firmantes. Este es otro de los motivos del porqué los sub anillos deben tener un GAP que indique asimetría en la partición justa y en las particiones simuladas.

En el documento [2] se explica lo siguiente. Sea t un número entero y Π = { π^1, …, π^n} un conjunto de particiones [1,..,n] en t subconjuntos; $\underline{\pi}$ define una partición del anillo R = {P_1, …,P_2} en t sub anillos R^1 hasta R^t. Entonces lo anterior dice que $\underline{\Pi}$ son el conjunto de nodos del súper anillo, los cuales son realizados para los todos los grupos ad-hoc que contienen dos o más firmantes.

En el caso de contar con dos usuarios (t = 2), sea π = {π^1, π^2 }particiones de [1, n] usuarios yP_1 y P_2 dos usuarios que quieren probar colaboración 2 fuera de n para un mensaje. Si los usuarios P_1 y P_2 pertenecen a distintos sub anillos, es decir P_1 está en R^1 y P_2 está en R^2, es posible producir dos correctas firmas en anillo para cada una de los usuarios. En este caso se dice que existe una partición justa para los índices {1, 2}.

Para probar el anonimato, es necesario proporcionar un conjunto Π de particiones tal que para los índices {1,2} en [1,..n], existe una partición justa π que está en Π para {1,2}. Los subanillos calculados son un meta anillo sobre el conjunto Π, esto prueba que al menos en una partición π ambos sub anillos subyacente pueden ser resueltos al mismo tiempo.

Por otro lado el caso general, sea π = (π^1,…, π^t)una partición de [1, n] en t subconjuntos y sea I={} un conjunto de t índices para [1, n]. si todos los enteros I pertenecen a t diferentes subconjuntos por ejemplo i_j está en π^j, se dice que π es una partición justa para i.

Para asegurar el anonimato de la partición justa, es necesario proporcionar Π tal que exista una partición justa para cada uno de los conjuntos de cardinalidad t, entonces:

- Sea t< n dos enteros, se dice que el conjunto Π de [1,n] es un sistema (n,t)-completo si para cada uno de los valores I de carnalidad t, existe una partición justa en Π para I:

o $$\forall I \subset [1,n],\ \#(I) = t, \quad \exists \pi = (\pi^1, \ldots, \pi^t) \in \Pi, \quad \forall j \in [1,t],\ \#(I \cap \pi^j) = 1$$

Ecuación 3: Partición justa.

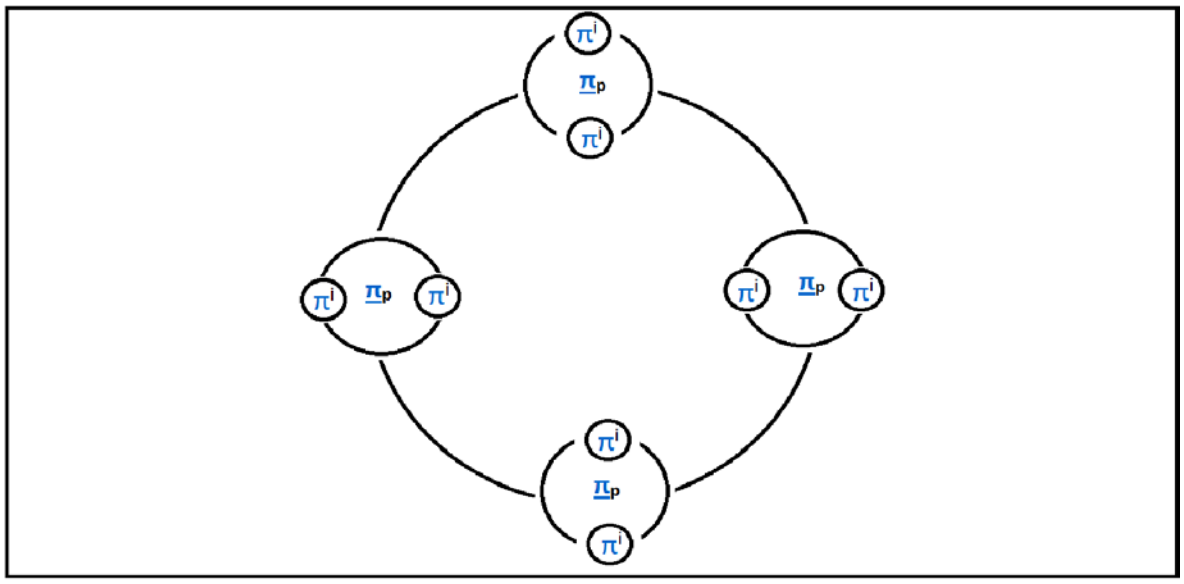


Figura 22: este anillo es la representación de la teoría de conjuntos expuesta con los autores de [2]. Π = {π1,…, πp}, donde p es el total de las particiones y πp = {π^1, …,π^i} don de i = {i_1, …, i^t} tal que para cada valor i existe una partición justa.

Entonces teniendo esta demostración presentada por los autores tanto para la presentación de un caso particular y un caso general, se puede probar que existe una posible partición justa que será asignada para cado uno de los firmantes que podrían o no tener el rol. También, no menos importante, los autores mencionan que la construcción de las particiones justas debe ser de tal manera que no existe intersección de los usuarios firmantes en cada uno de los grupos, por otro lado, cuando los usuarios intersectan en cada uno de las particiones esto quiere decir que no pertenecen al grupo de los firmantes de la partición.

8.2. Esquema

La firma de anillo con umbral, como se mencionó anteriormente, debe contar con entidades como un súper anillo, los nodos del súper anillo, y las particiones de cada uno de los nados. Estas particiones que deben indicar cuál es la posición de los usuarios dentro de un grupo, los autores asumen que está dada por defecto, no obstante, en este proyecto no es necesaria su utilización por lo que se implementa un prototipo “simulador de particiones”, el cual no será expuesto en mayor profundidad ya que solamente realiza la lectura de un archivo que sigue un estándar de comunicación; anteriormente expuesto en la sección 5, específicamente en Fig.18.

Dado lo anterior, se tiene que la firma en anillo con umbral, al igual que los anteriores

algoritmos cuenta con dos sistemas separados:

- Algoritmo de firma con umbral
- Algoritmo de verificación con umbral

en esta implementación la cantidad de particiones es el óptimo necesario, es decir son tantas como sea posible según la combinación N!/ t! (N-t)!. Esta ecuación, en el documento [2], es distinta, ya que se toma en cuenta que el orden de los usuarios en un anillo siempre entregará un resultado diferente.

En los siguientes algoritmos presentados se muestra una función de hash G la cual debe devolver un arreglo de largo (t x l)-bits. también no menos importantes es mencionar que las particiones deben estar públicamente disponibles, ya que de esta manera se puede comprobar que verdaderamente los usuarios utilizados en la firma son reales, ya que para partición justa podría caber la posibilidad que un mismo usuario firmó los subanillos de la partición, por ende, si la estructura de las particiones es pública siempre se sabrá quién fue el firmante de las particiones en el método de verificación.

8.2.1. Algoritmo de firma

Los autores de [2] denotan para P_{i1}, …, P_{it} un subgrupo de usuarios que quieren firmar un mensaje mientras se prueba que hay a lo menos t firmantes entre n usuarios del anillo. la idea es resolver una colección de sub anillos correspondientes a los grupos firmantes y los no firmantes, entonces al resolver los anillos y utilizando un mecanismo de resolución de un súper anillo se podría verificar que a lo menos unos de los grupos de sub anillos fue resuelto individualmente con llaves privadas. Para esto se entiende los siguiente: Denotar para π_s una partición justa ara I = {i_1, …, i_t}, entonces se asume que para cada j que está entre [1, t] existe i_j que está en π^j_s.

1- Escoger un valor seed para cada uno de los subanillos de cada partición.

$$\textbf{For } i = 1, \ldots, p, \textbf{ Do}$$
$$\quad \textbf{For } k = 1, \ldots, t, \textbf{ Do } v_i^k \xleftarrow{R} \{0,1\}^\ell.$$

2- Simular anillos para todas las particiones de menos para la partición de los usuarios firmantes.

$$\textbf{For } i = 1, \ldots, p, \; i \neq s, \textbf{ Do}$$
$$\quad \textbf{For } j = 1, \ldots, n, \textbf{ Do } x_i^j \xleftarrow{R} \{0,1\}^\ell, \text{ and } y_i^j \leftarrow g_j(x_i^j).$$
$$\quad \textbf{For } k = 1, \ldots, t, \textbf{ Do}$$
$$\quad\quad z_i^k \leftarrow C_{v_i^k,1,m}(0, y_i^j, j \in \pi_i^k(R)) \text{ and } \gamma_i^k \leftarrow v_i^k \oplus z_i^k.$$

3- Calcular el súper anillo con los valores GAP obtenidos.

$\sigma_s \xleftarrow{R} \{0,1\}^{t\ell}$, and $u_{s+1} \leftarrow G(\sigma_s)$
For $i = s+2, \ldots, p, 1, \ldots, s$, **Do**
$u_i \leftarrow G\big(u_{i-1} \oplus (\gamma_{i-1}^1 \| \ldots \| \gamma_{i-1}^t)\big)$

4- Calcular el valor GAP para los subanillos de la partición de los firmantes, la cual debe cerrar el súper anillo.

$(\gamma_s^1 \| \ldots \| \gamma_s^t) \leftarrow u_s \oplus \sigma_s$.

5- Resolver los subanillos para la partición justa del súper anillo.

For $j \in [1,n] \backslash I$, **Do** $x_s^j \xleftarrow{R} \{0,1\}^{\ell}$, and $y_s^j \leftarrow g_j(x_s^j)$.
For $j \in I$, **Do**
$\sigma_k \xleftarrow{R} \{0,1\}^{\ell}$ for k such that $j \in \pi_s^k$
$y_s^j \leftarrow C^{-1}_{j,\gamma_s^k,m}(\sigma_k, y_s^j, j \in \pi_s^k(R))$ and $x_s^j \leftarrow {g_j}^{-1}(y_s^j)$.

6- Generar la salida de los valores de la firma.

$\nu \xleftarrow{R} [1,p]$ and output $\left(\nu, u_\nu, \bigcup_{1 \le i \le p} \left(x_i^1, \ldots, x_i^n, v_i^1, \ldots, v_i^t\right)\right)$.

8.2.2. Algoritmo de verificación

un anillo de t firmantes con n usuarios (t-out-of-n) es verificado de la siguiente forma:

1- Calcular todos los sub anillo comenzando desde el índice 1.

For $i = 1, \ldots, p$, **Do**
For $j = 1, \ldots, n$, **Do** $y_i^j \leftarrow g_j(x_i^j)$.
For $k = 1, \ldots, t$, **Do**
$z_i^k \leftarrow C_{v_i^k,1,m}(0, y_i^j, j \in \pi_i^k(R))$, and $\gamma_i^k \leftarrow v_i^k \oplus z_i^k$.

2- Verificar el súper anillo desde el índice v y obteniendo los valores GAP.

$u_\nu \stackrel{?}{=} G(\gamma_{\nu-1}^1 \| \ldots \| \gamma_{\nu-1}^t \oplus G(\ldots G(\gamma_\nu^1 \| \ldots \| \gamma_\nu^t \oplus u_\nu) \ldots))$.

8.3. Implementación

Para una mayor apreciación del diagrama de clases del algoritmo de firma en anillo con umbral las clases serán expuestas en primer luego solo con sus dependencias, para luego mostrar cada uno de ellas por separado junto con sus métodos y atributos.

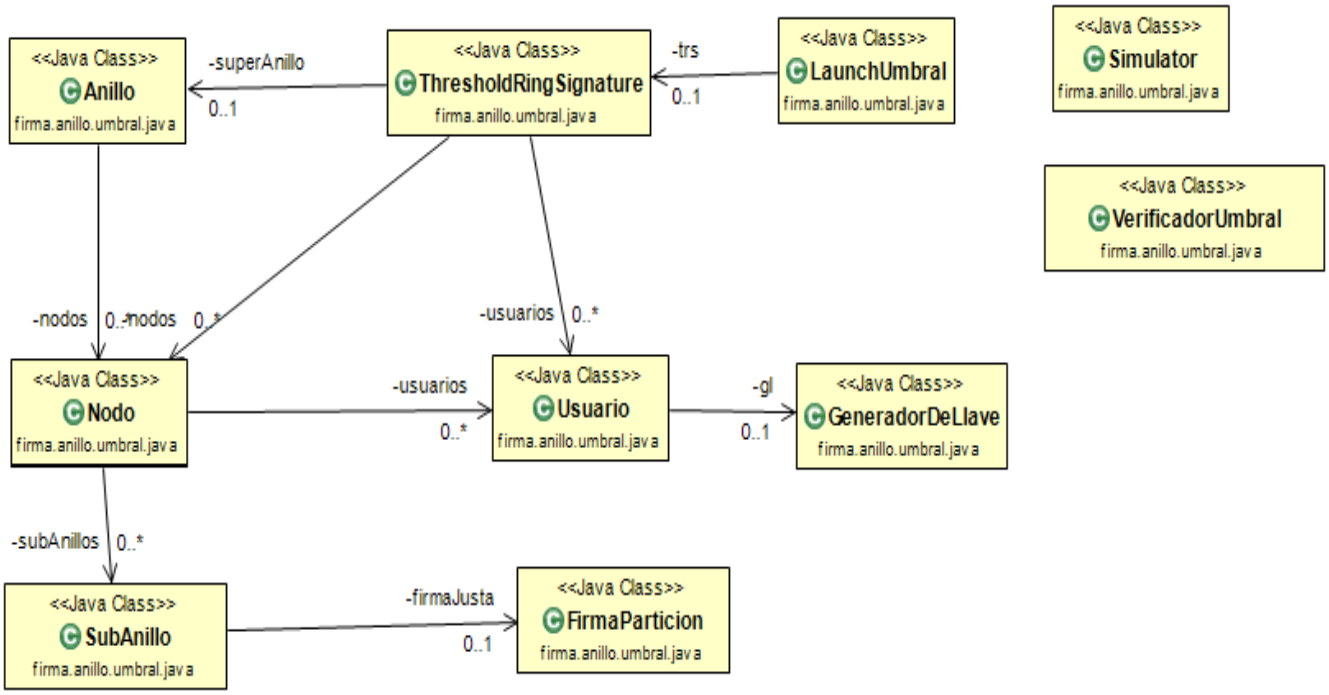


Figura 23: diagrama de casos de uso de la firma con umbral; basada en [5].

Esta clase es la encargada de simular el super anillo del algoritmo, por lo tanto, debe contener los nodos o particiones predefinidas por la clase simulador de particiones.

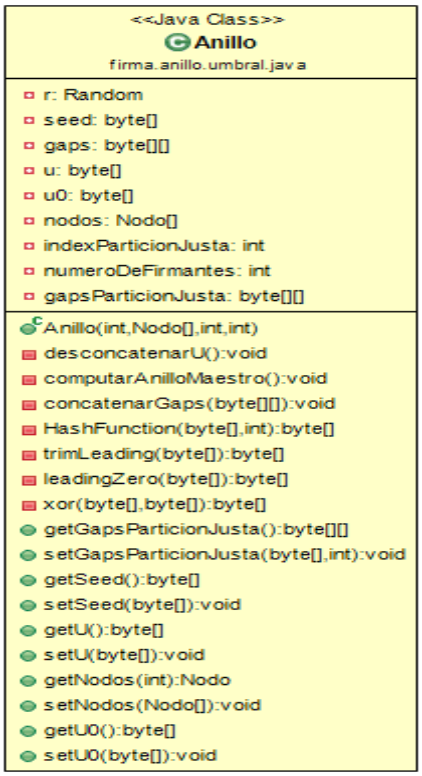


Figura 24: Clase anillo de la firma con umbral.

La clase" FirmaParticon" es la encargada de, mediante la utilización de las llaves privadas de los usuarios, crear los subanillos de la partición justa, esos sub anillo demuestran que solo una partición presenta colaboración.

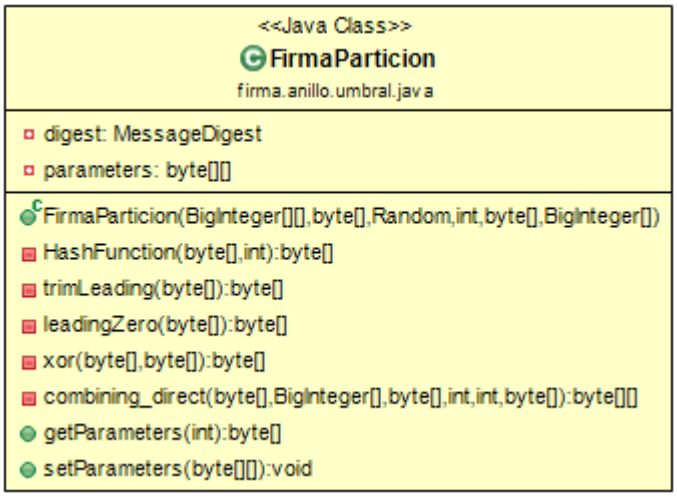


Figura 25: clase FirmaParticion de la firma con umbral.

la clase "GeneradorDeLlave" se encarga de entregar la llave privada y pública de los usuarios que firmaran y la pública de los usuarios que no firmaran.

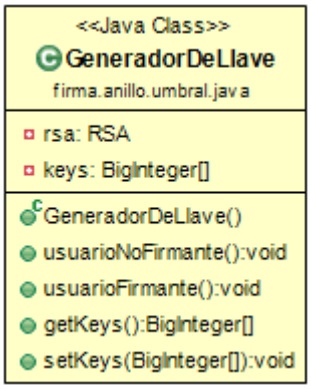


Figura 26: clase GeneradorDeLlave de la firma con umbral.

La clase “simulator” se encarga de encarga de simular las particiones que no contienen a los usuarios firmantes, por lo tanto, esta implementa la capacidad de generar un anillo solamente con usuarios con llaves públicas.

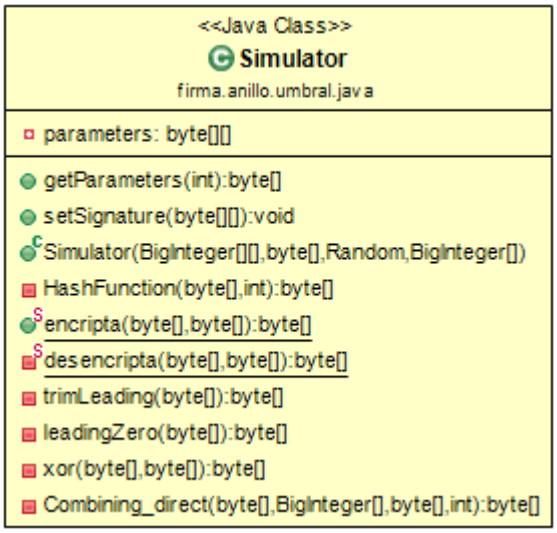


Figura 27: clase simulador de la firma con umbral.

La clase “Usuario” se encarga de almacenar las llaves de todos los usuarios del grupo.

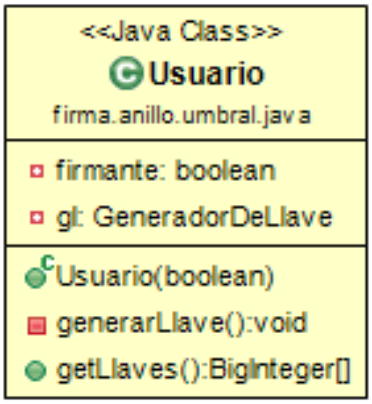


Figura 28: clase usuario de la firma con umbral.

La clase “Nodo” se encarga de mantener y de diferenciar cuales son los nodos o particiones que deben ser simuladas y cuales son la partición justa.

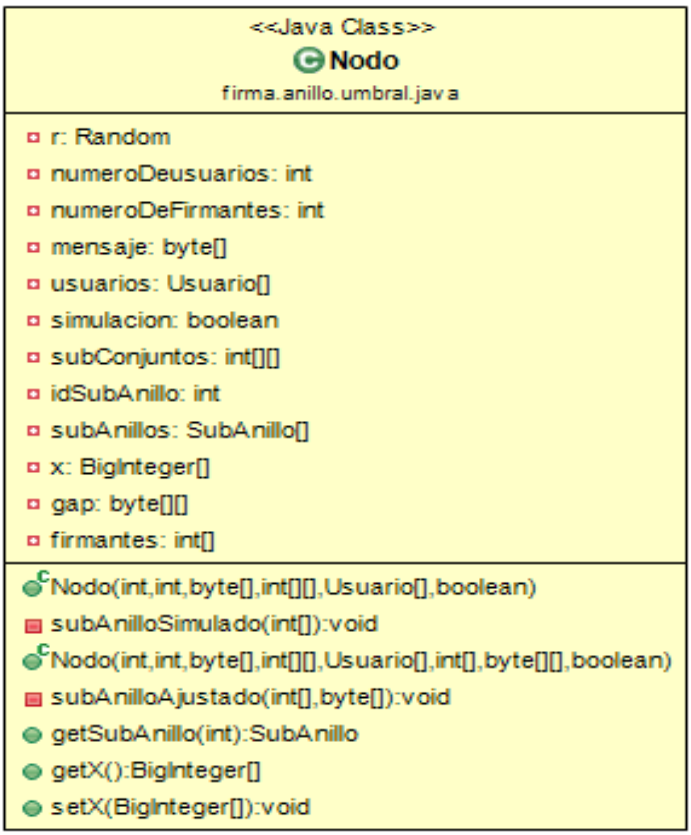


Figura 29: clase Nodo de la firma con umbral.

La clase “Sub Anillo” tiene la capacidad de crear ya sea un anillo simulado o un anillo perteneciente a un firmante gracias a la sobreescritura de métodos.

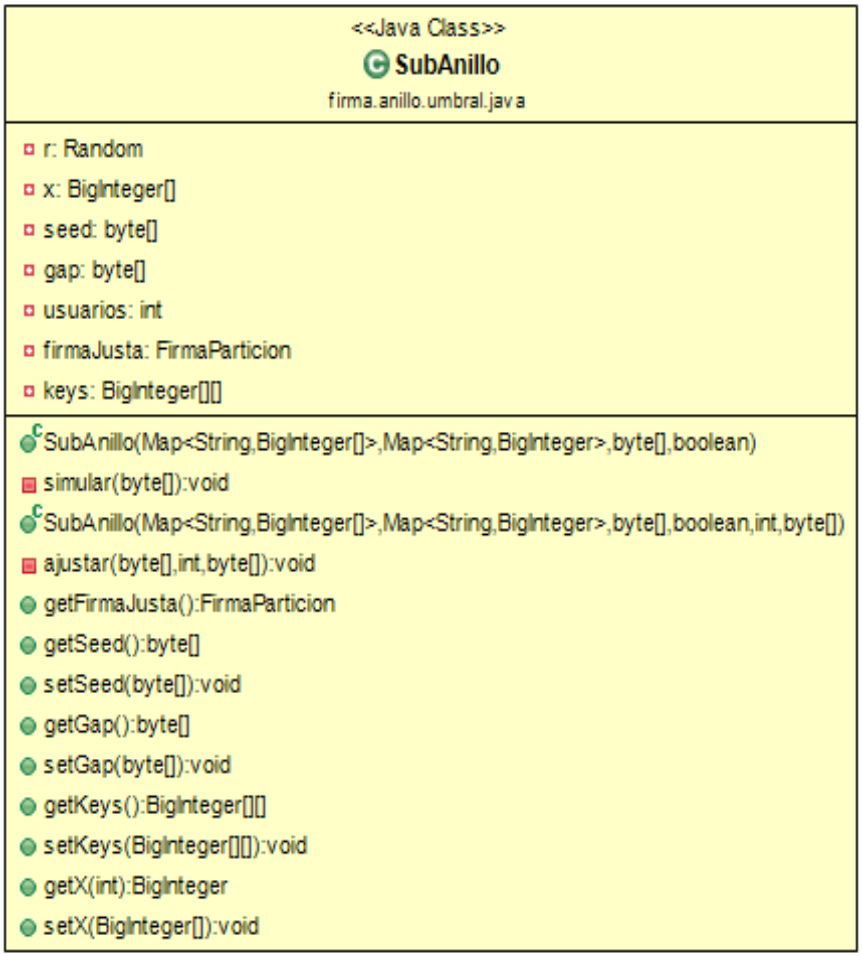


Figura 30: clase SubAnillo de la firma con umbral.

La clase “ThresholdRingSignature” es la encargada de generar la firma a partir de todos los datos que se pueden reunir de los subanillos, los nodos y el súper anillo.

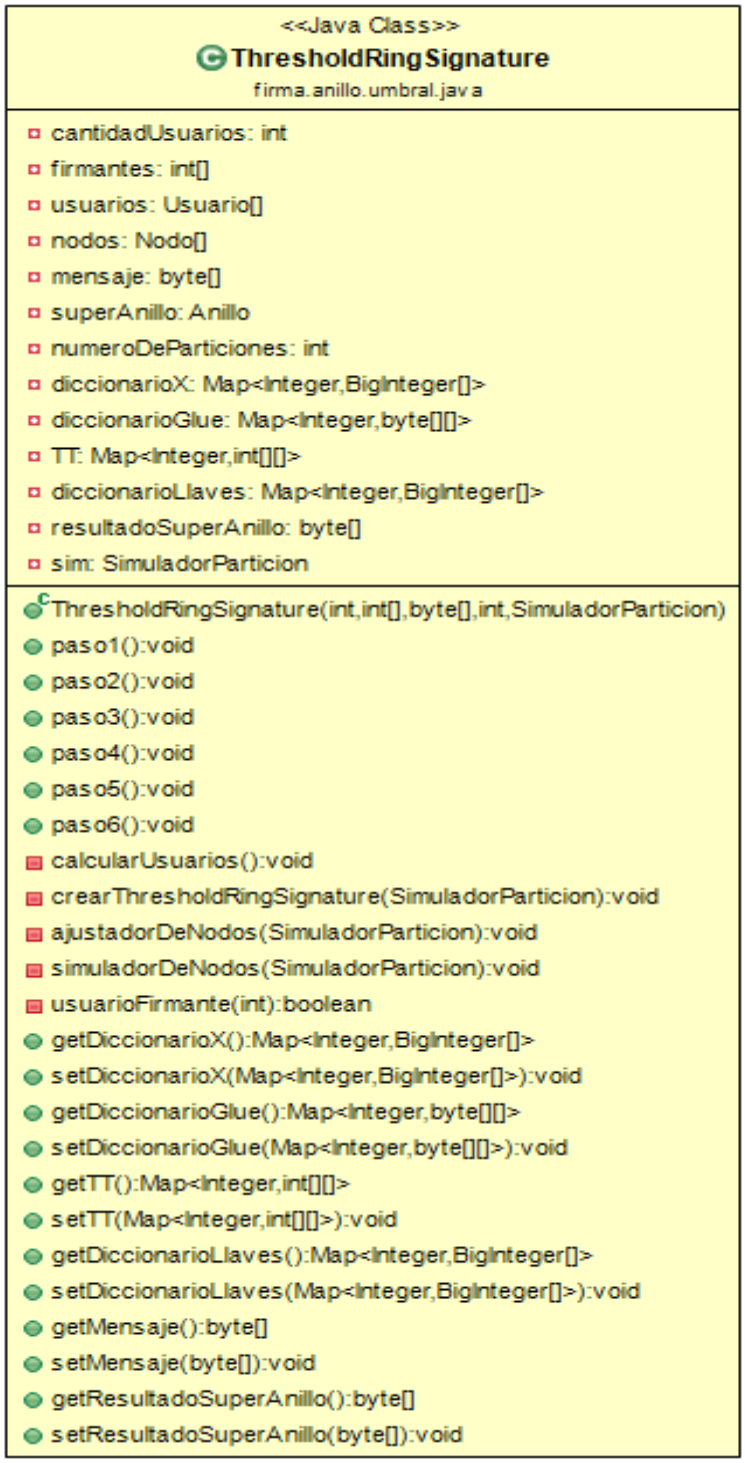


Figura 31: Clase ThresholdRingSignature de la firma con umbral.

Por último, como bien dice su nombre la clase “varificadorUmbral” es la encargada de verificar la firma con umbral.

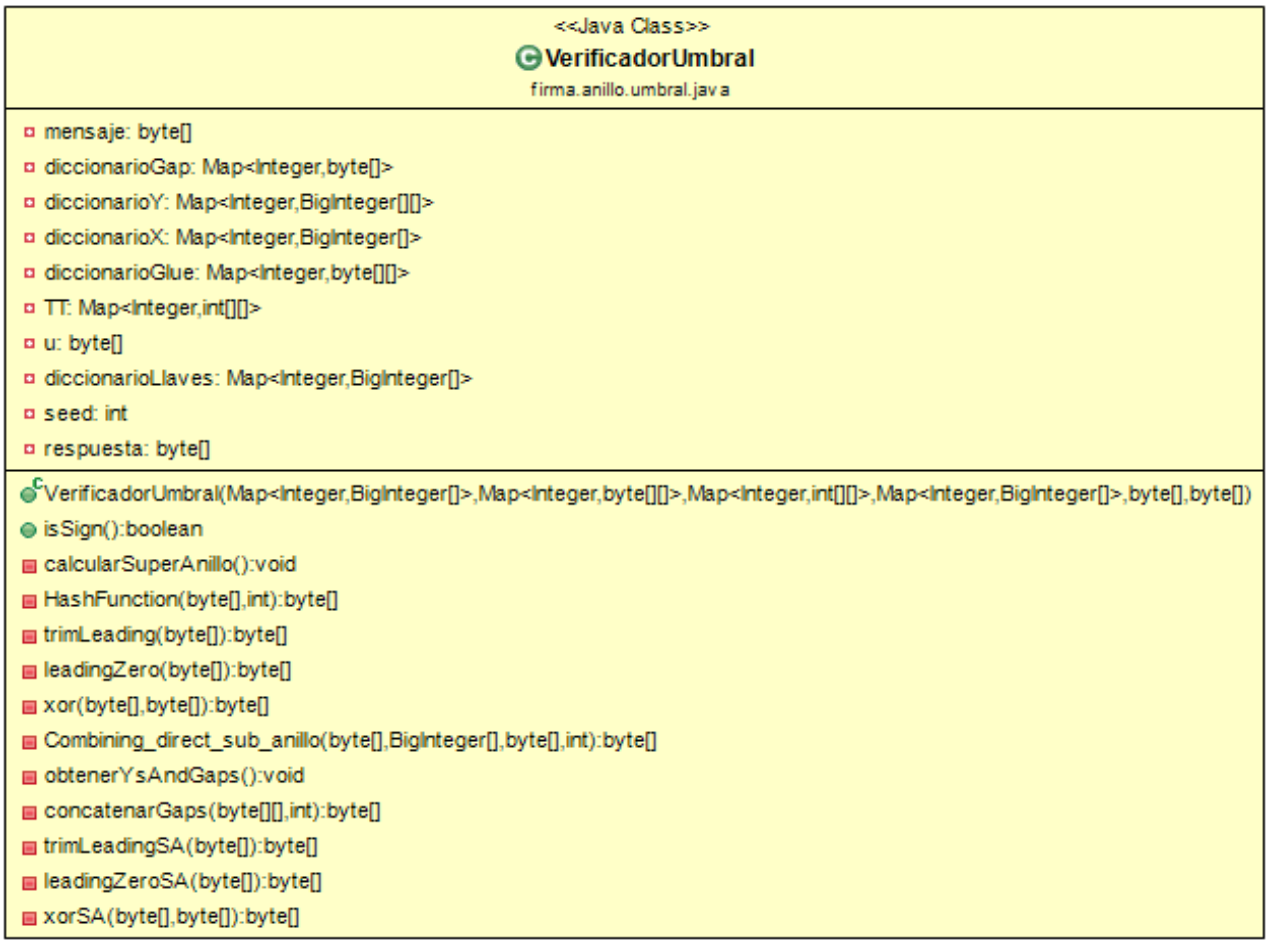


Figura 32: clase VerificadorUmbral de la firma con umbral.

En cada una de la clase los métodos XOR, leading Zero, TrimLeading y las funciones de hash presentan modificaciones, por lo que la opción más fácil, pero no la más óptima de aplicarlas, es crear para cada una de las clases su correspondiente variación de los métodos mencionados con anterioridad.

8.3.1. Clases destacadas

Entre todas las clases presentadas del algoritmo de firma en anillo con umbral, existen clases que deberían ser analizadas con más detalle, por lo tanto, aquí se entregara código fuente de las clases encargadas de simular y ajustar los subanillos.

La clase "**FirmaParticion**" como se dijo anteriormente es la encargada de calcular los anillos de la partición justa, por lo tanto, en el constructor de la clase se realizan las operaciones necesarias para generar el anillo, por ende, en primer lugar, se consiguen todos los valores Y de los usuarios del sub anillo:

```
BigInteger[] y = new BigInteger[keys.length];

for(int i = 0; i<keys.length;i++){//seleccion de los Y que son de los usuarios no firmantes...

    if(i!=s){
        y[i] = rsa.encrypt(x[i],keys[i][0],keys[i][1]); //(ei,Ni,Null) RSA
        if (y[i].toByteArray().length == 129) {//limitar tamaño de y no firmanate...
            System.out.println("X["+i+"]: "+ x[i].hashCode());
            do{
                x[i] = new BigInteger(1024,r);
                if(x[i].toByteArray()[0]<0){//asegurar que el x sea positivo
                    x[i].toByteArray()[0]=(byte) ((byte) x[i].toByteArray()[0]*-1);
                }

                y[i] = rsa.encrypt(x[i],keys[i][0],keys[i][1]);
                System.out.println("-");
            }while(y[i].toByteArray().length>128);


        }
    }

}
```

Figura 33: extracción de parte del algoritmo de firma del sub anillo del usuario firmante.

El algoritmo está encargado de que los valores que deben ser escogidos al azar no presentan una saturación de buffer, en un arreglo de 128 bytes.

Luego de generar los valores, comienza el proceso de generación del valor Y correspondiente al usuario firmante, a través de:

```
byte[][] Vs=null;
byte[] seed=new byte[128];
do{
    do{
        do{//garantias de que el valor sera siempre positivo
            r.nextBytes(seed);
        }while(new BigInteger(seed).compareTo(new BigInteger("0")) == -1);

        Vs = combining_direct(seed,y,message,s, keys.length, gap);
        y[s]= new BigInteger(this.xor(seed, Vs[s]));
        x[s] = rsa.decrypt(y[s],keys[s][2],keys[s][1]);
        BigInteger t = rsa.encrypt(x[s],keys[s][0],keys[s][1]);
        //System.out.println("y:"+y[s]+" \n t: "+t);
        //System.out.println("ks:"+keys[s][2]+" \n kp: "+keys[s][0]);
        //System.out.println("n:"+keys[s][1]);
        //System.exit(0);

    }while(!y[s].equals(rsa.encrypt(x[s],keys[s][0],keys[s][1])));


}while(y[s].compareTo(new BigInteger("0")) == -1 );
//asumimos que envia todos los Z (s+1,...,s-1)
```

Figura 34: extracción del detalle de la saturación de buffer, en la clase FirmaParticion

Entonces con este algoritmo nos aseguramos, a través de los operadores “do while” que el valor encontrado para los usuarios sea siempre un valor que no provoque una saturación de buffer, si esto sucede el valor seed o semilla se cambia hasta que la combinación convocada entregue un valor sin error.

Por último, el método “combining_direct” convocado en la anterior operación es el encargado de ajustar el sub anillo con el gap que es salvado del súper anillo con la llave privada del firmante.

```
private byte[][] combining_direct(byte[] seed,BigInteger[] y, byte[] mensaje, int s, int users, byte[] gap)
                                                    throws NoSuchAlgorithmException, NoSuchPaddingException,
                                                    InvalidKeyException, IllegalBlockSizeException,
                                                    BadPaddingException{

            //Vs[i] = combining_direct(v,y,message,s+1);
            byte[] glue = seed;
            byte[][] Vs = new byte [users][131];

            //indice para identificar la primera vez que entramos al anillo
            int value =0;
            byte[] c ;//concatenacion
```

Figura 35: parte 1 del método combining_direc..

```
//concatenacion de la raiz y el mensaje
for (int i = s+1; true ; i++){
	c = new byte[glue.length + mensaje.length];

	//para el caso inial:
	if (value==0){
		//verificamos si sobrepaza el valor del anillo para comenzar desde 0:
		if (i>=users){
			i=0;
			//si se inicia desde 0 realizamos un xor entre la seed y e gap
			//glue= this.xor(glue, gap);

			//concatenacion entre glue y mensaje:
			System.arraycopy(glue, 0, c, 0, glue.length);
			System.arraycopy(mensaje, 0, c, glue.length, mensaje.length);
			//sacamos el hash de la concatenacion(tamaño 1024, para que el resultado final sea 128)
			Vs[i]=this.xor(gap, this.HashFunction(c,1024));

			//cambiamos glue con el valor de Vs, para ser usado en el siguiente:
			glue=Vs[i];
			value ++;
		}else{
			//si no comenzamos desde 0 no realizar xor entre gap y seed

			//concatenacion entre glue y mensaje:
			System.arraycopy(glue, 0, c, 0, glue.length);
			System.arraycopy(mensaje, 0, c, glue.length, mensaje.length);
			//sacamos el hash de la concatenacion(tamaño 1024, para que el resultado final sea 128)
			Vs[i]=this.HashFunction(c,1024);

			//cambiamos glue con el valor de Vs, para ser usado en el siguiente:
			glue=Vs[i];
			value ++;
		}



	}else{
```

Figura 36: parte 2 del método combining_direct.

```
                if (i>=users){
                        //si se inicia desde 0 realizamos un xor entre la seed y e gap
                        i=0;
                        //si sobrepasamos el indice entonces comenzamos en 0 y utilizamos el y[n]
                        glue= this.xor(glue, y[users-1].toByteArray()) ;

                        //concatenacion entre glue y mensaje:
                        System.arraycopy(glue, 0, c, 0, glue.length);
                        System.arraycopy(mensaje, 0, c, glue.length, mensaje.length);
                        //sacamos el hash de la concatenacion(tamaño 1024, para que el resultado final sea 128)
                        Vs[i]=this.xor(gap, this.HashFunction(c,1024));

                        //cambiamos glue con el valor de Vs, para ser usado en el siguiente:
                        glue=Vs[i];

                }else{

                        //si recorremos el anillo con normalidad solo aplicar xor al glue y el indice y[i-1]
                        glue= this.xor(glue, y[i-1].toByteArray() );

                        //concatenacion entre glue y mensaje:
                        System.arraycopy(glue, 0, c, 0, glue.length);
                        System.arraycopy(mensaje, 0, c, glue.length, mensaje.length);
                        //sacamos el hash de la concatenacion(tamaño 1024, para que el resultado final sea 128)
                        Vs[i]=this.HashFunction(c,1024);

                        //cambiamos glue con el valor de Vs, para ser usado en el siguiente:
                        glue=Vs[i];
                }


                }
                if (i==s)break;

        }//end for

        return Vs;
}
```

Figura 37: parte 3 del método combining_direct.

Este método resulta trivial, ya que solamente debe existir la idea de recorrer el sub anillo en un sentido y cuando se llegue a estar en la posición última se debe agregar el gap pre definido para que el subanillo sea asimétrico.

Por otro parte la clase "**Simulator**" se vuelve más trivial ya que no existe ningún ajuste, es decir para todos los usuarios se calculan valores X e Y:

```
BigInteger[] y = new BigInteger[keys.length];
//seleccion random de valores X's e Y's
for(int i = 0; i<keys.length;i++){//Conversion de todos los x a y
    //System.out.println("Posicion:"+i);
    y[i] = rsa.encrypt(x[i],keys[i][0],keys[i][1]); //(ei,Ni,Null)
    System.out.print("X["+i+"]:"+x[i].hashCode());
    System.out.println(" -> Y["+i+"]: "+y[i].hashCode());

}
```

Figura 38: extracto del algoritmo de la clase Simulator

Para terminar el algoritmo con una combinación directa que realiza la operación de XOR entre todos estos valores, para así encontrar un valor GAP que indica que el anillo es asimétrico:

```
private  byte[] Combining_direct(byte[] v,BigInteger[] y, byte[] mensaje, int users) throws NoSuchAlgorithmException,
                                                                                    NoSuchPaddingException, InvalidKeyException,
                                                                                    IllegalBlockSizeException, BadPaddingException{

   byte[] glue = v;
   byte[] z = new byte [128];



   //concatenacion de la raiz y el mensaje
   byte[] c = new byte[glue.length + mensaje.length];
   for (int i = 0; i<users ; i++){
       //iniciamos glue con el valor de Vs e Ys anterior:
       glue=xor(glue, y[i].toByteArray());//valores de largo 128

       //concatenacion entre glue y mensaje:
       System.arraycopy(glue, 0, c, 0, glue.length);
       System.arraycopy(mensaje, 0, c, glue.length, mensaje.length);

       //sacamos el hash de la concatenacion(tamaño 1024, para que el resultado final sea 128)
       z=this.HashFunction(c,1024);
       glue=z;

   }
   return z;
}
```

Figura 39: extracto del algoritmo de la clase Simulator.

9. Pruebas de firma en anillo con umbral

Como se menciona anteriormente el algoritmo de firma en anillo con umbral presenta un error de salida, ya que cuando un usuario firmante realiza la operación de ajuste de su sub anillo asignado, el valor Y incognito es devuelto como un arreglo de 128 bytes, no obstante, cuando este valor Y es descifrado con la respectiva llave privada el valor X resulta de 129 bytes.

Después de realizar diversas pruebas con el código involucrado, es decir la clase "**FirmaParticion**", se derivó que entre los valores involucrados el valor que al cambiarlo evitaría el error es el valor semilla (seed).

En esta sección estará descrito específicamente el problema y la solución a la saturación de buffer, por lo demás se incluirán pruebas de ejecución para detallar los algunos valores que son importantes dentro del algoritmo y así verificar el buen funcionamiento y la correcta ejecución de los pasos del algoritmo de firma en anillo con umbral.

9.1. Saturación de buffer

la clase encargada de ejecutar la combinación directa para el sub anillo que contiene a uno de los usuarios firmantes del grupo es "**FirmaParticion**" por ende en esta sub sección se analizara en detalle la saturación de buffer que ocurre antes de haber presentado cualquier solución, es decir aquí se llevara una cuenta de los pasos seguidos para solucionar el error.

Primero, una vez terminado todo el algoritmo en una primera ejecución tenemos como resultado que:

- el valor Y encontrado para el subanillo es:
  - Y:145337878866969436077210701825247701826530363504216791333061023056098738965788931115449797664737612417942663326835846577783827989134782561615911077236271967518261833251258468985677554742620694807599858245997492265151329497522889852016976848548148757883217174547959568156178620953899482970199949325863420574344
  - Largo Y: 129 bytes

- El valor X de G(Y) para el subanillo es:
    - X:955272400523610742860657813578000357377031989861278108184467070322318645238254657113488663701803447226622200165462067178566915101297399345034758323977195803987069662641778348689758297029285976622260436338758718899982318709494590956971585565682422164791306286660423425288999095976997736957418245947774070700979
    - Largo X: 128 bytes

Entonces al conocer estos valores tenemos que si el valor X lo volvemos a cifrar nos entrega:

- F(X):57367173131721017778103376273839347857960297062601828601065498127862871808231189247645976544847270405201168507804097216439046250532709610289670982811113153856134810849254589383924952606518748115121321382243761661687344729656953817744229078243427932931508923126289490440476708671230366834365477476130940139461
- Largo de Y: 128

Como podemos notar el valor Y que tenemos ahora es un valor de 128 bytes distinto al valor Y inicialmente calculado. Por lo tanto, para solucionar este problema se realiza un ajuste de todos los valores a utilizar en para el subanillo, ya que en este caso el valor si lo pasamos a bits en el valor más significativo es 1 y este valor el contado como negativo en el arreglo de bytes, es decir algo como:

- Valor Y calculado:
    - Bytes: 14533787886…
    - Bits: 11110110011110100000010101…

Cuando este valor es pasado a un Byte de 128, porque ese es el largo estándar escogido para todos los valores X e Y del algoritmo, el valor X cuenta el Bit de la posición 0 si es 1 como negativo y si es 0 como positivo. En este caso como el valor es 1 el de la posición 0, por lo tanto, el valor más significativo, su representación queda como un arreglo negativo de 128 bytes, por lo tanto, al tratar de convertir el valor X a Y queda un valor distinto, es decir algo como:

- Valor X:
    - Bytes: 955272400...
    - Bits: 000111111111000…
- Valor de X a Y:
    - Bytes: 57367173...
    - Bits: 0010101010101010…

Una vez que se realizaron los ajuste para que todos los valores no sean superiores a 128 bytes, el error seguía sucediendo, por lo que se concluye que el error esta en otro calor distinto a X e Y, por lo tanto, el único valor distinto involucrado era el seed o semilla, que al igual que los otros, este valor es controlados para que siempre fuera un valor de 128 bytes, ya que el XOR es entre valores de un mismo largo y si los bytes no son iguales, en cuanto a largo, quedará un valor sin su operación XOR; por lo demás se utiliza el operador XOR de java por lo que es estrictamente que los valores de entrada tengan el mismo largo.

Entonces debido a que el valor semilla es el valor que generaría menos costo computacional al cambiarlo, se realiza un cambio en el algoritmo de tal forma que este valor se cambie hasta que el valor X e Y coinciden al momento de cifrar o descifrar, según el algoritmo estime necesario; sería Y=F(x), o también X=G(y).

```
byte[] seed=new byte[128];
do{
    do{
        do{//garantias de que el valor sera siempre positivo
            r.nextBytes(seed);
        }while(new BigInteger(seed).compareTo(new BigInteger("0")) == -1);

        Vs = combining_direct(seed,y,message,s, keys.length, gap);
        y[s]= new BigInteger(this.xor(seed, Vs[s]));
        x[s] = rsa.decrypt(y[s],keys[s][2],keys[s][1]);
        BigInteger t = rsa.encrypt(x[s],keys[s][0],keys[s][1]);

    }while(!y[s].equals(rsa.encrypt(x[s],keys[s][0],keys[s][1])));

}while(y[s].compareTo(new BigInteger("0")) == -1 );
//asumimos que envia todos los Z (s+1,...,s-1)
```

Figura 40: extracto del algoritmo de firma, don del se agregan los Do while de comprobación para la saturación de buffer.

Cuando en la clase "**FirmaParticion**" se llega al paso en que se debe realizar la combinación para encontrar los valores incógnitos, se establecen sentencias de condición que indican que si el valor Y es distinto al valor cifrado de X volvemos a realizar toda la operación, además si el valor Y es negativo también se debería volver a realizar toda la operación.

Para comprobar un promedio de cuantas veces es ajustado el valor semilla antes de que todo esté funcionando correctamente se realizaron 20 pruebas con 4 usuarios y dos firmantes, con distintos usuarios en cada prueba.

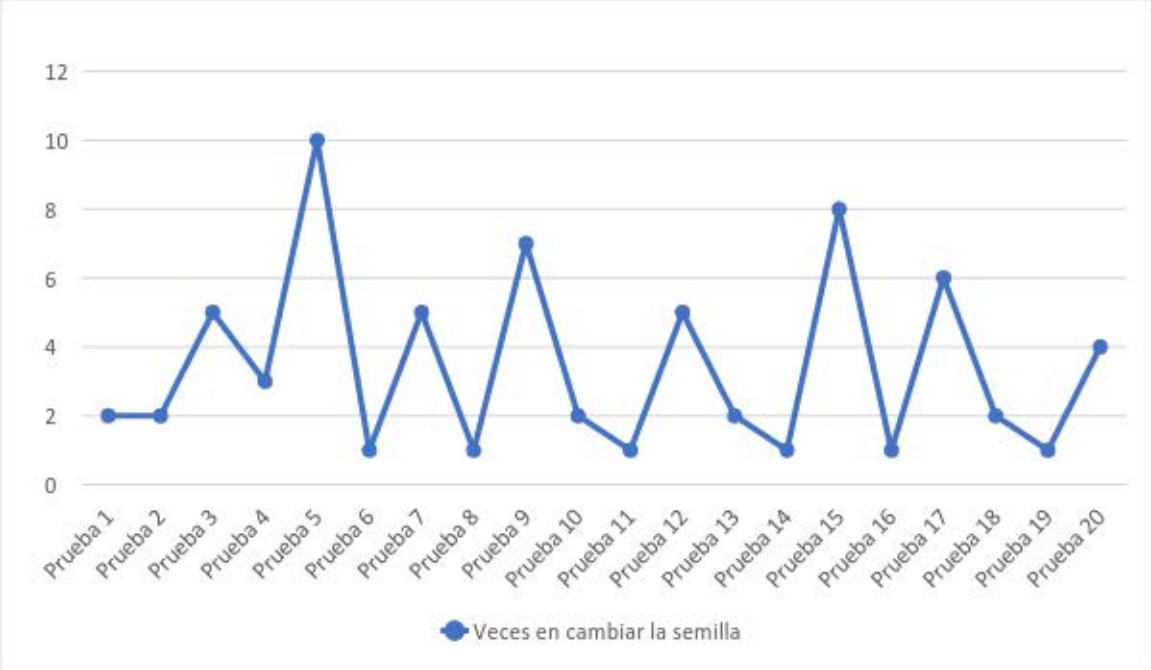


Tabla 4: pruebas de algoritmo basado en hash.

Esta tabla demuestra la variación en la cantidad de veces que se cambia el valor semilla, en total estos valores entregan un promedio de 3.45 veces por prueba lo que según la rapidez de ejecución de la instrucción puede resultar en insignificante para el rendimiento del algoritmo.

9.2. Ejecución del algoritmo de firma en anillo con umbral

El algoritmo de firma en anillo con umbral tiene ubicado estratégicamente diversas variables de salida que pueden ser observadas por consola java. Estos valores son los necesarios para entender la eta del proceso en que se encuentra el algoritmo.

En los siguientes datos de salida se construye un anillo de cuatro usuarios y dos firmantes, en donde si pensamos que los usuarios están almacenados en un arreglo, las posiciones serían de la 0 hasta la 3. Entre esas posiciones los usuarios 0, 1 y 3 son firmantes en el anillo.

Algoritmo **de firma**

En el **paso 1** del algoritmo de firma, se realiza una lectura de los usuarios y se da a conocer mediante una variable booleana si el usuario es o no firmante:

usuario[0], firmante: true
usuario[1], firmante: true
usuario[2], firmante: false
usuario[3], firmante: true

siguiendo, el algoritmo en el **paso 2** indica, sin mostrar datos, que se está realizando una lectura de los grupos ad-hoc que deben estar pre definidos para la construcción del súper anillo.

El **paso 3** del algoritmo con umbral tiene como salida los datos de las simulaciones de los subanillos de los nodos que deben ser simulados. Además, se hace referencia a las llaves de cada uno de los usuarios participantes; el valor e corresponde a la llave pública del usuario n es el módulo de RSA. Por último, también se puede observar el valor Glee y el valor GAP de cada uno de los subanillos. Como también se puede observar como este anillo debe contar con la participación de 3 firmantes, cada uno de los nodos cuanta con 3 sub anillos de 2 usuarios.

Nodo: 0

sub Anillo: 0
usuario: 0 -> key ->e: -332403942 ->n: 253725969
usuario: 3 -> key ->e: -100341684 ->n: -2067016577
X[0]:-1275908362 -> Y[0]: -2071424224
X[1]:627786470 -> Y[1]: -1262644663
GLUE: -273909511
GAP: -1343772113

sub Anillo: 1
usuario: 1 -> key ->e: 1830565037 ->n: -1784099854
usuario: 3 -> key ->e: -100341684 ->n: -2067016577
X[0]:1055337745 -> Y[0]: -653125055
X[1]:627786470 -> Y[1]: -1262644663
GLUE: 1909991706
GAP: -1518009275

sub Anillo: 2
usuario: 2 -> key ->e: -1818362495 ->n: 1072992224
usuario: 3 -> key ->e: -100341684 ->n: -2067016577
X[0]:2124386466 -> Y[0]: 1972510258
X[1]:627786470 -> Y[1]: -1262644663
GLUE: -144460363
GAP: 1923398841

Nodo: 2

sub Anillo: 0
usuario: 0 -> key ->e: -332403942 ->n: 253725969
usuario: 1 -> key ->e: 1830565037 ->n: -1784099854
X[0]:1486584398 -> Y[0]: 611561914
X[1]:-2070226782 -> Y[1]: 1537049842
GLUE: -1476520556
GAP: -754521243

sub Anillo: 1
usuario: 1 -> key ->e: 1830565037 ->n: -1784099854
usuario: 2 -> key ->e: -1818362495 ->n: 1072992224
X[0]:-2070226782 -> Y[0]: 1537049842
X[1]:-359523305 -> Y[1]: 5922224
GLUE: 2023823657
GAP: 795395501

sub Anillo: 2
usuario: 1 -> key ->e: 1830565037 ->n: -1784099854
usuario: 3 -> key ->e: -100341684 ->n: -2067016577
X[0]:-2070226782 -> Y[0]: 1537049842
X[1]:2063729636 -> Y[1]: 1084501432
GLUE: -548068248
GAP: -1988938399

Nodo: 3

sub Anillo: 0
usuario: 0 -> key ->e: -332403942 ->n: 253725969
usuario: 1 -> key ->e: 1830565037 ->n: -1784099854
X[0]:463393559 -> Y[0]: 1518506839
X[1]:-1531241553 -> Y[1]: -441058508
GLUE: -1416656825
GAP: -1025051774

sub Anillo: 1
usuario: 0 -> key ->e: -332403942 ->n: 253725969
usuario: 2 -> key ->e: -1818362495 ->n: 1072992224
X[0]:463393559 -> Y[0]: 1518506839
X[1]:-1129278208 -> Y[1]: -183184047
GLUE: 1834007316
GAP: -1587591546

sub Anillo: 2
usuario: 0 -> key ->e: -332403942 ->n: 253725969
usuario: 3 -> key ->e: -100341684 ->n: -2067016577
X[0]:463393559 -> Y[0]: 1518506839
X[1]:-341108964 -> Y[1]: -310361424
GLUE: 1805208079
GAP: -1312178648

Para cuando se llega al **paso 4**, con los subanillos simulados ya calculados, se debe construir el súper anillo. En este paso se indica la posición (dentro de un arreglo) del nodo que contiene a los usuarios firmantes. Se muestra el valor GAP, es decir el valor a encontrar en el súper anillo, se muestra también el valor GLUE del súper anillo, el valor z del súper anillo y la des-concatenación del valor GAP para los tres sub anillos de la partición justa.

Paso 4: Super anillo calculado
Super-anillo calculado correctamente
Nodo particion justa: 1
Gaps particion justa: 445778320
U indice 0 glue: 1044183383
Z-Us: -854337320
gaps de sub-anillo: [0]615641232
gaps de sub-anillo: [1]-358925973
gaps de sub-anillo: [2]1628031893

en el **paso 5** del algoritmo, entonces, se realiza el cálculo de los subanillos de la partición justa. En un principio se muestra el nodo correspondiente que se está ajustando, para luego proceder con la demostración de los subanillos. En estos se observa a los usuarios participantes, luego se muestra la poción que utiliza el firmante dentro del sub anillo, y por último se muestran los valores obtenidos de la combinación de ajuste con el valor GAP obtenido en el cálculo de súper anillo, es decir el valor X y el valor Y=G(X) del usuario firmante del sub anillo. También no menos importante, se muestra el valor GLUE y el valor GAP del sub anillo.

nodo: 1

sub Anillo: 0
usuario: 0 -> key ->d: 64675619 ->n: 253725969
usuario: 2 -> key ->e: -1818362495 ->n: 1072992224
POSICION DEL FIRMANTE: 0
Generando X firmante...
X [0]:
69697895458657561169893804789803227170741946923831040648891979797253255713682962593106832267740547870177972171932072699958120697473666460235995081553137850410319507193189756838197370175095886936170356574984788996056297309133686388199350814775679006402765364028372779941251708545682800922819998941517125957402 -> Y [0]:
62401371471080646539064817582388259979565859043382985070958957976967304239073238216763644025485363593242394702947961163942131191842654229246839959836858052766007706489536132458145297264537666293392657669844052307665738052914963466470557482371102346862281443485637783531497088320373560293146396908350020501196 -> Usuario firmante -> G(X):
62401371471080646539064817582388259979565859043382985070958957976967304239073238216763644025485363593242394702947961163942131191842654229246839959836858052766007706489536132458145297264537666293392657669844052307665738052914963466470557482371102346862281443485637783531497088320373560293146396908350020501196
X [1]: 356198998 -> Y [1]: 414587389 -> key: [Ljava.math.BigInteger;@4e25154f
Glue: -1911010392
GAP: 615641232

sub Anillo: 1
usuario: 1 -> key ->d: 1657307254 ->n: -1784099854
usuario: 2 -> key ->e: -1818362495 ->n: 1072992224
POSICION DEL FIRMANTE: 0
Generando X firmante...
X [0]:
32762952089261726446758739595360483414644709437021998693803854850576206932083400971553981450418622769983047888171456129053165210927389909991721274465264016487131695829685105773008744017911537737367961172915609291698395222559330637027599655294060645764848286941294044388452723859851092232095682712596260424597 -> Y [0]:
58629983584385074674996516228740072033851981356130235604740834002810633739615005648204256819838701040030825778221827998290444336437444613111883994056539440175441900388445569297160475042370191734576711033169592297043001182362646515871014871922116292244342203885030780563848607656034797037529450163686616913570 -> Usuario firmante -> G(X):
58629983584385074674996516228740072033851981356130235604740834002810633739615005648204256819838701040030825778221827998290444336437444613111883994056539440175441900388445569297160475042370191734576711033169592297043001182362646515871014871922116292244342203885030780563848607656034797037529450163686616913570
X [1]: 356198998 -> Y [1]: 414587389 -> key: [Ljava.math.BigInteger;@4e25154f
Glue: 1767068515
GAP: -358925973

sub Anillo: 2
usuario: 2 -> key ->e: -1818362495 ->n: 1072992224
usuario: 3 -> key ->d: 1963969425 ->n: -2067016577
POSICION DEL FIRMANTE: 1
Generando X firmante...
X [0]: 356198998 -> Y [0]: 414587389 -> key: [Ljava.math.BigInteger;@4e25154f
X [1]:
1114809726950042669114393015841512581353722270613916086069972271805707668422178930184106586944509580830527097761182987541939362689576955689605552836202077848549772767273844492972837305993914051871455071444782645618760690003123549104691721819574590159128152319821050472403122793099095280730419614286618059872164 -> Y [1]:
6540408175040150223693733306959994038294040263076502922209432914916875948945920727026911449108376524678018592159890184570707438444853065995308491595843788754105860860076374371997045664395717635827323586075339311085310331373533516882587380314866672858075249410569411528834997621512650740983413127932817396443 0 -> Usuario firmante -> G(X):
6540408175040150223693733306959994038294040263076502922209432914916875948945920727026911449108376524678018592159890184570707438444853065995308491595843788754105860860076374371997045664395717635827323586075339311085310331373533516882587380314866672858075249410569411528834997621512650740983413127932817396443 0
Glue: -671837157
GAP: 1628031893

Por ultimo para finalizar con el algoritmo de firma, en el **paso 6** se muestran los datos que se almacenan de la firma en anillo con umbral. Se comienza almacenando todos los valores x que se utiliza en cada nodo, luego los Glue de cada sub anillo correspondiente al nodo especificado, también los sub grupos con los que debe ser construido cada sub anillo por ultimo las llaves públicas de los usuarios.

DICCIONARIO DE X
Nodo:0
X[0]: -1275908362
X[1]: 1055337745
X[2]: 2124386466
X[3]: 627786470
Nodo:1
X[0]: 427395337
X[1]: -1536628783
X[2]: 356198998
X[3]: 1034600784
Nodo:2
X[0]: 1486584398
X[1]: -2070226782
X[2]: -359523305
X[3]: 2063729636
Nodo:3
X[0]: 463393559
X[1]: -1531241553
X[2]: -1129278208
X[3]: -341108964

DICCINARIO DE GLUE
Nodo:0
Glue[0]: -273909511
Glue[1]: 1909991706
Glue[2]: -144460363
Nodo:1
Glue[0]: -1911010392
Glue[1]: 1767068515
Glue[2]: -671837157
Nodo:2
Glue[0]: -1476520556
Glue[1]: 2023823657
Glue[2]: -548068248
Nodo:3
Glue[0]: -1416656825
Glue[1]: 1834007316
Glue[2]: 1805208079

DICCIONARIO DE TT(SUB-CONJUNTOS)
Nodo: 0
TT[0]: 03
TT[1]: 13
TT[2]: 23
Nodo: 1
TT[0]: 02
TT[1]: 12
TT[2]: 23
Nodo: 2
TT[0]: 01
TT[1]: 12
TT[2]: 13
Nodo: 3
TT[0]: 01
TT[1]: 02
TT[2]: 03

DICCIONARIO DE LLAVES PUBLICAS DE USUARIOS
Usuario: 0 -> Para X: 0 -> clave publica: [Ljava.math.BigInteger;@70dea4e
Usuario: 1 -> Para X: 1 -> clave publica: [Ljava.math.BigInteger;@5c647e05
Usuario: 2 -> Para X: 2 -> clave publica: [Ljava.math.BigInteger;@33909752
Usuario: 3 -> Para X: 3 -> clave publica: [Ljava.math.BigInteger;@55f96302

Algoritmo de verificación

Los datos de salida del proceso de verificación hacen referencia a la construcción de los nodos del súper anillo, mostrando ahí los datos de sus sub anillos y la concatenación de los GAPS de cada sub anillo.
Nodo:0
Sub-anillo[0]:
X[0]: -1275908362 -> Y[0]: -2071424224
X[3]: 627786470 -> Y[1]: -1262644663
Z: 73482160
Glue: -273909511
Gap: -1343772113

Sub-anillo[1]:
X[1]: 1055337745 -> Y[0]: -653125055
X[3]: 627786470 -> Y[1]: -1262644663
Z: 1748287103
Glue: 1909991706
Gap: -1518009275

Sub-anillo[2]:
X[2]: 2124386466 -> Y[0]: 1972510258
X[3]: 627786470 -> Y[1]: -1262644663
Z: -1333978248
Glue: -144460363
Gap: 1923398841

Concatenacion de gaps: -960128722

Nodo:1
Sub-anillo[0]:
X[0]: 427395337 -> Y[0]: 1801670733
X[2]: 356198998 -> Y[1]: 414587389
Z: -1125210288
Glue: -1911010392
Gap: 615641232

Sub-anillo[1]:
X[1]: -1536628783 -> Y[0]: 1638352337
X[2]: 356198998 -> Y[1]: 414587389
Z: -1262387820
Glue: 1767068515
Gap: -358925973

Sub-anillo[2]:
X[2]: 356198998 -> Y[0]: 414587389
X[3]: 1034600784 -> Y[1]: 1748728358
Z: 846552916
Glue: -671837157
Gap: 1628031893

Concatenacion de gaps: 445778320

Nodo:2
Sub-anillo[0]:
X[0]: 1486584398 -> Y[0]: 611561914
X[1]: -2070226782 -> Y[1]: 1537049842
Z: 1940462651
Glue: -1476520556
Gap: -754521243

Sub-anillo[1]:
X[1]: -2070226782 -> Y[0]: 1537049842
X[2]: -359523305 -> Y[1]: 5922224
Z: -1355550164
Glue: 2023823657
Gap: 795395501

Sub-anillo[2]:
X[1]: -2070226782 -> Y[0]: 1537049842
X[3]: 2063729636 -> Y[1]: 1084501432
Z: -1873014473
Glue: -548068248
Gap: -1988938399

Concatenacion de gaps: -1327823756

Nodo:3
Sub-anillo[0]:
X[0]: 463393559 -> Y[0]: 1518506839
X[1]: -1531241553 -> Y[1]: -441058508
Z: -455556681
Glue: -1416656825
Gap: -1025051774

Sub-anillo[1]:
X[0]: 463393559 -> Y[0]: 1518506839
X[2]: -1129278208 -> Y[1]: -183184047
Z: 1811131352
Glue: 1834007316
Gap: -1587591546

Sub-anillo[2]:
X[0]: 463393559 -> Y[0]: 1518506839
X[3]: -341108964 -> Y[1]: -310361424
Z: -641788437
Glue: 1805208079
Gap: -1312178648

Concatenacion de gaps: 928451121

Para finalizar el proceso de verificación muestra unos datos muy fáciles de comprobar a la vista, los cuales indican si el súper anillo es verdadero o no, estos datos son el valor U que es el valor GLUE o valor inicializador y el valor Z del súper anillo, los cuales si son iguales indican la validez de los datos recibidos del algoritmo de firma, los cuales en este caso indican que el anillo es correcto.

Comprobacion
U-0: 1044183383
Calculo Súper anillo: 1044183383

Observación: *en pasajes de los datos de salida se puede observar que los valores X e Y tienen un largo muy grande, esto es para poder comprobar que los valores no son negativos y que no ocurre una saturación de buffer. También no menos importante, se puede comprobar que ambos algoritmos calculan y llegan a los mismos resultados en cada una de las partes mostradas.*

10. Representación gráfica

La interfaz de usuario que se dispondrá para utilizar los algoritmos es realizada para dar a entender de forma más didácticas los algoritmos, por lo que debe ser amigable con el usuario y fácil de utilizar.

Una vez terminado los algoritmos de firma digital, en este proyecto se realiza una representación gráfica del uso de estos. Entonces teniendo en cuanta esto se comienza con un diseño, de interfaz gráfica, según los procesos que fueron explicados en las anteriores secciones.

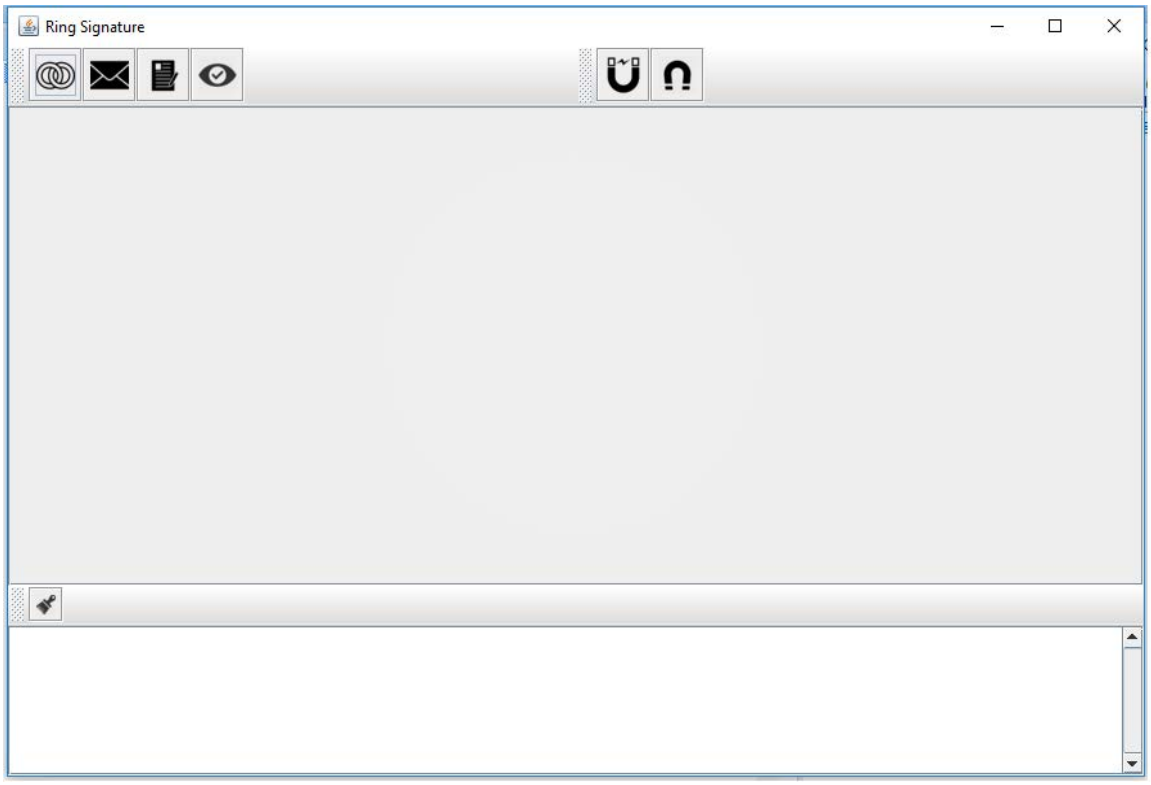
En esta sección, entonces, se muestran las ventas de la interfaz gráfica, así como una pequeña explicación de su navegación y por último una explicación más al detalle de cada una de las funcionalidades más importantes del sistema, a través de los casos de uso y su documentación

10.1. Modelos de la interfaz gráfica

Ventana principal de la interfaz gráfica de usuario, cuenta con en el norte con un panel que contiene 6 botones, de izquierda a derecha se encargan de:

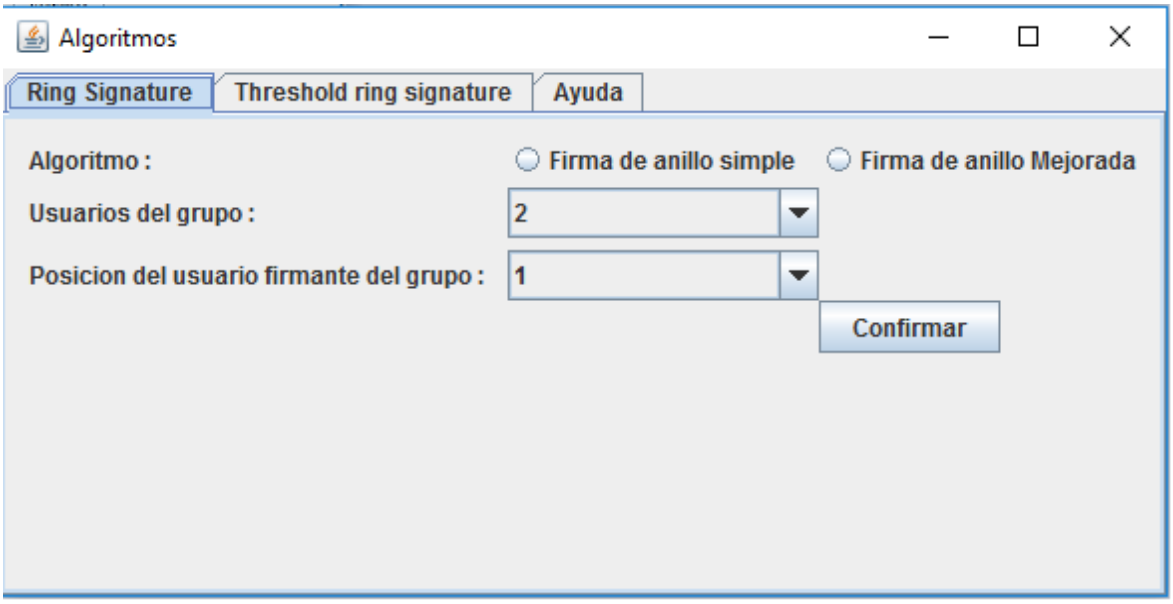
- Botón 1: elección del algoritmo a utilizar, junto con sus datos.
- Botón 2: escribir el mensaje a firmar.
- Botón 3: firmar el mensaje.
- Botón 4: verificar la firma del mensaje.
- Botón 5: encargado de centrar los nodos en un anillo.
- Botón 6: deja de mantener centrado los nodos en el anillo.

En el centro se observa un panel en donde se dibujan los nodos. Por último, en el panel del sur se encuentra ubicado una consola encargada de comunicar información de los nodos y los pasos de la firma y verificación. Y un panel que contiene un botón 7 encargado de limpiar los datos de la consola.



Frame 1: pantalla inicial.

Esta ventana aparece cuando al oprimir el botón 1, anteriormente mencionado. Ventana de algoritmos a seleccionar para firmar un mensaje. Aquí el algoritmo seleccionado el ring signature, se puede ver que se presenta una selección del algoritmo a utilizar, ya sea el simple o el mejorado, la cantidad de grupo de usuarios que componen al grupo y la posición del firmante dentro de un arreglo. Por último, el botón confirmar es para almacenar los datos de la información y esperar que el usuario firme el mensaje.

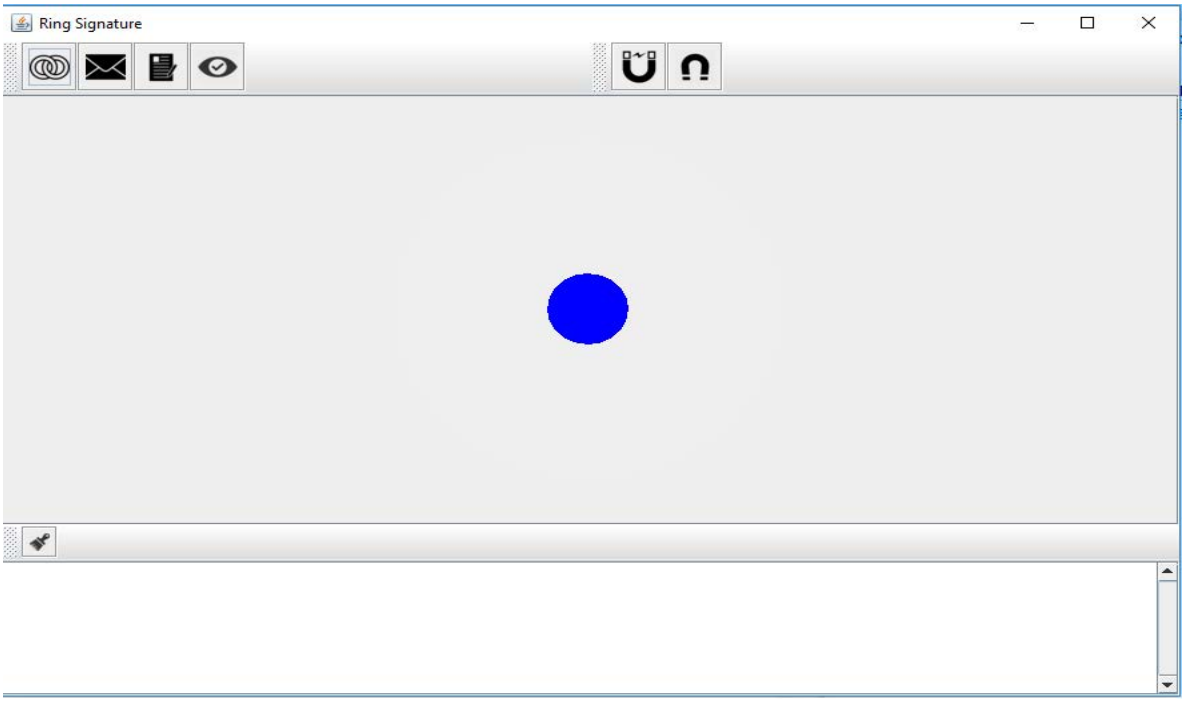


Frame 2: pantalla selección firma en anillo.

Esta ventana aparece cuando al oprimir el botón 1, anteriormente mencionado. La sección de firma de anillo con umbral presenta los valores del grupo y los usuarios. En la selección de usuarios del grupo se puede escoger entre 4 o 6 usuarios, entonces según la elección primera los siguientes valores se van ajustando ya que existen limitados archivos de particiones. Por último, el botón confirmar es para almacenar estos valores y esperar que el usuario firme el mensaje.

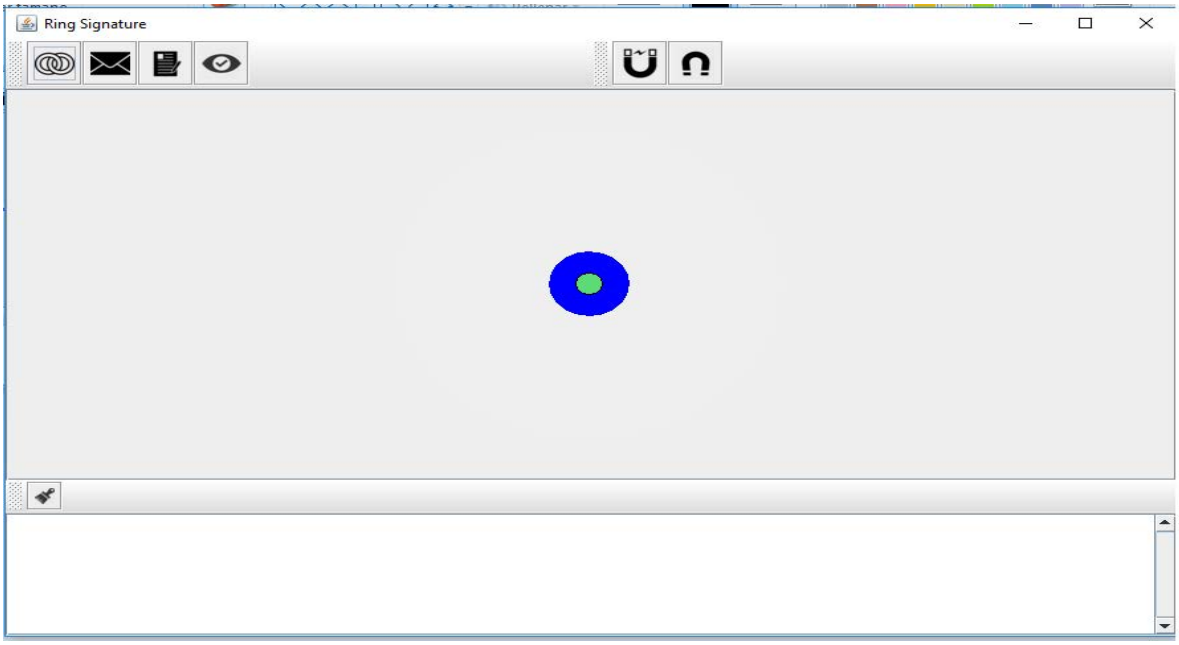
Frame 3: pantalla selección firma con umbral.

Ventana principal una vez que los valores para utilizar ring signature fueron confirmados.



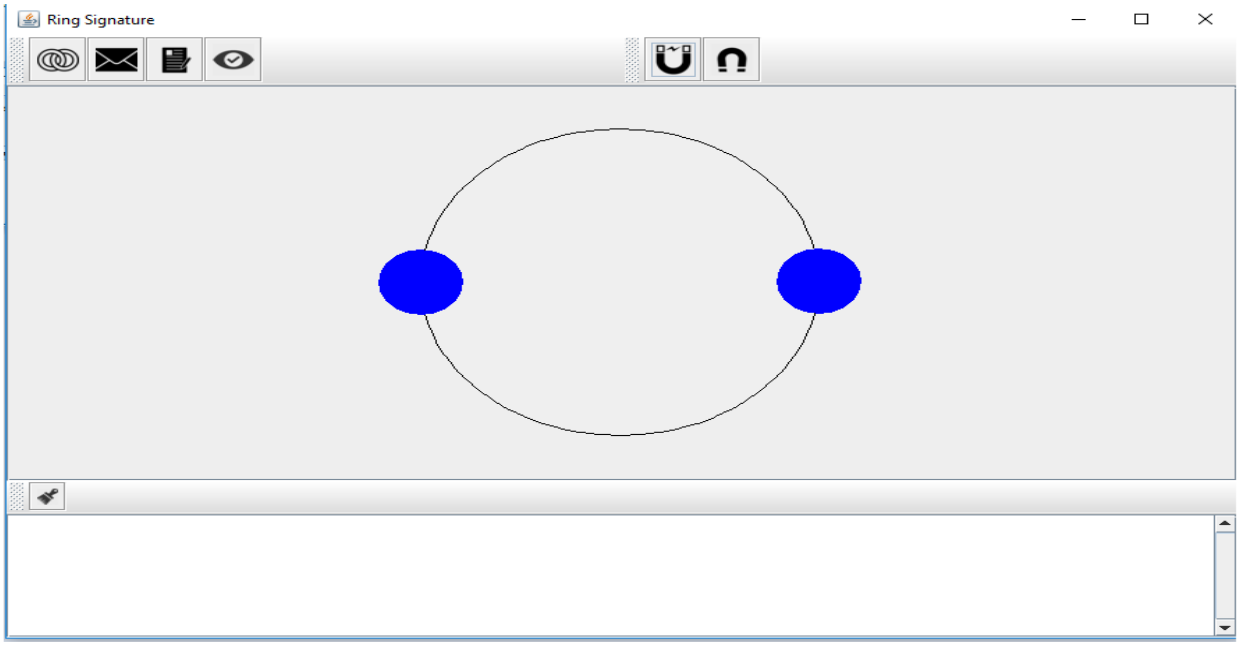
Frame 4: pantalla de principal con nodos de la firma en anillo.

La ventana principal una vez que los valores de la firma con umbral fueron confirmados.



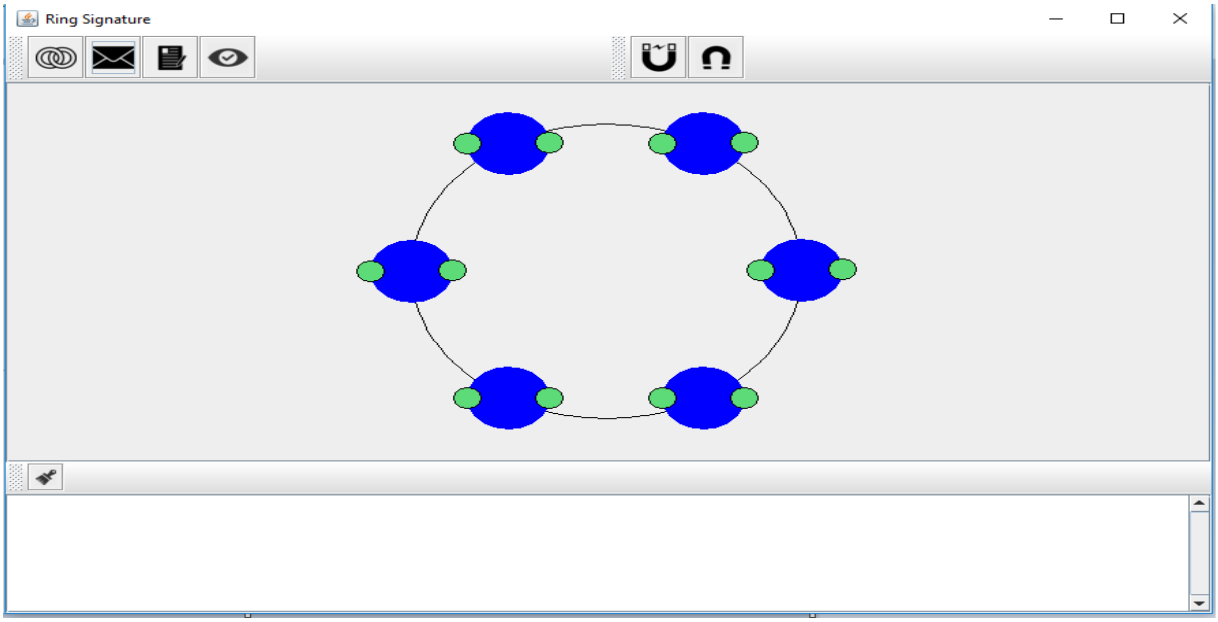
Frame 5: pantalla con los nodos de la firma en umbral.

Ventana principal, eligiendo como algoritmo la firma de anillo; 2 usuarios y el usuario 1 firmante.



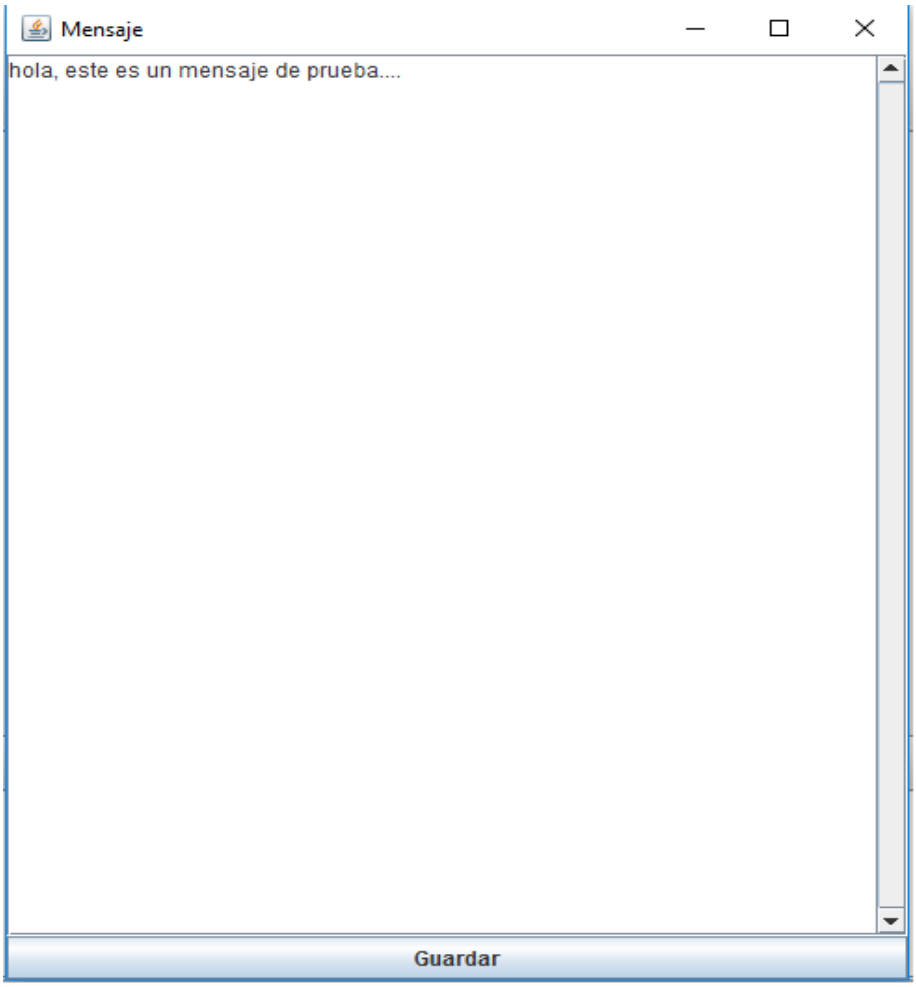
Frame 6: pantalla con la selección de la atracción de los nodos de la firma en anillo.

Ventana principal, eligiendo como algoritmo a la firma con umbral; 4 usuarios 2 firmantes.



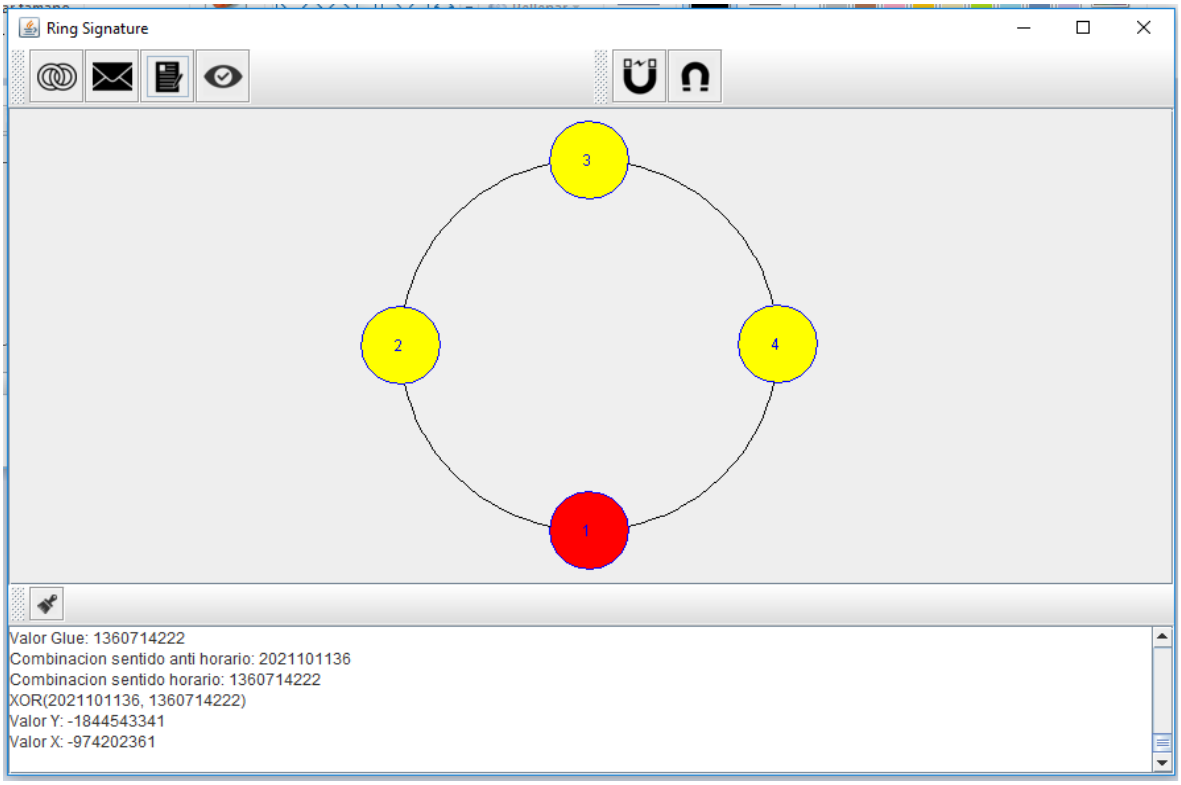
Frame 7: pantalla con la selección de la atracción de los nodos de la firma con umbral.

Ventana para escribir el mensaje a firmar. Una vez escrito con el botón guardar se almacena el mensaje a la espera que el usuario lo firme.



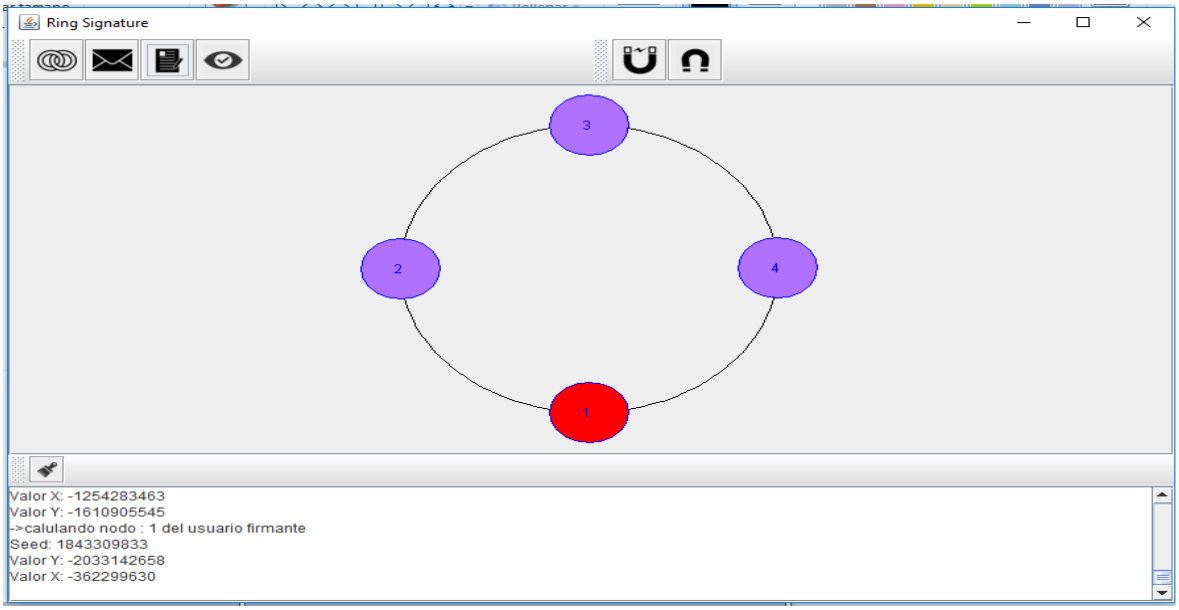
Frame 8: pantalla para escribir el mensaje.

Ventana principal ejecutando la acción de botón de firma, aplicando su función es decir cambiar de color los nodos en los que está operando y comunicando el paso en la consola. Algoritmo de firma en anillo simple.



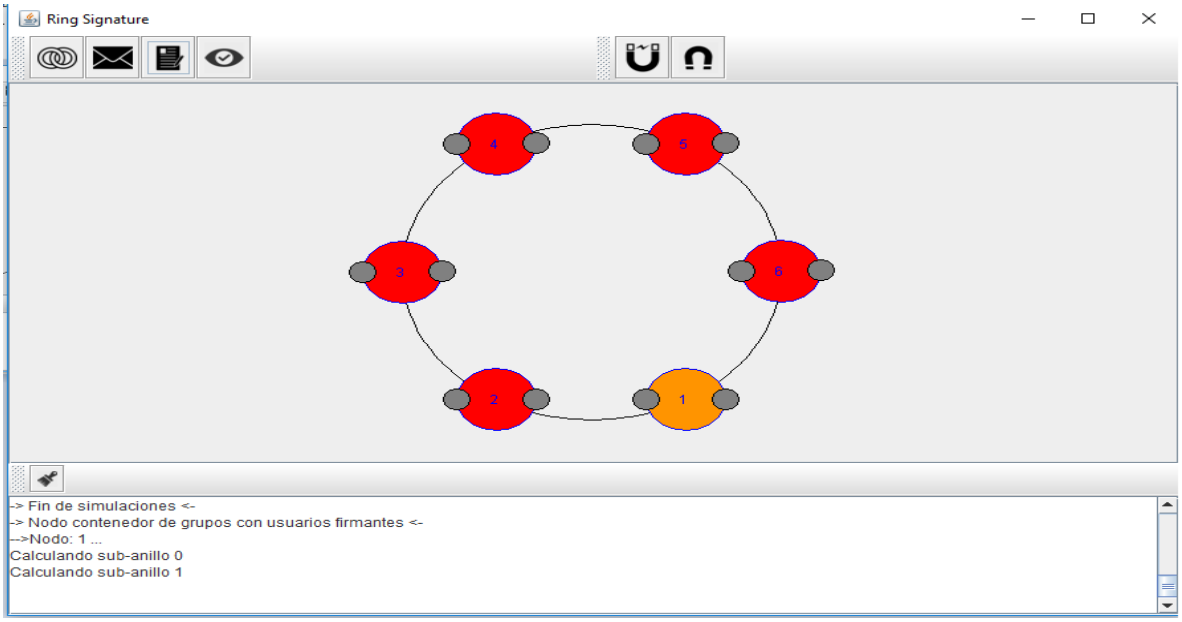
Frame 9: pantalla con la simulación de la firma en anillo simple.

Ventana principal ejecutando la acción de botón de firma, aplicando su función es decir cambiar de color los nodos en los que está operando y comunicando el paso en la consola; algoritmo de firma en anillo mejorado.



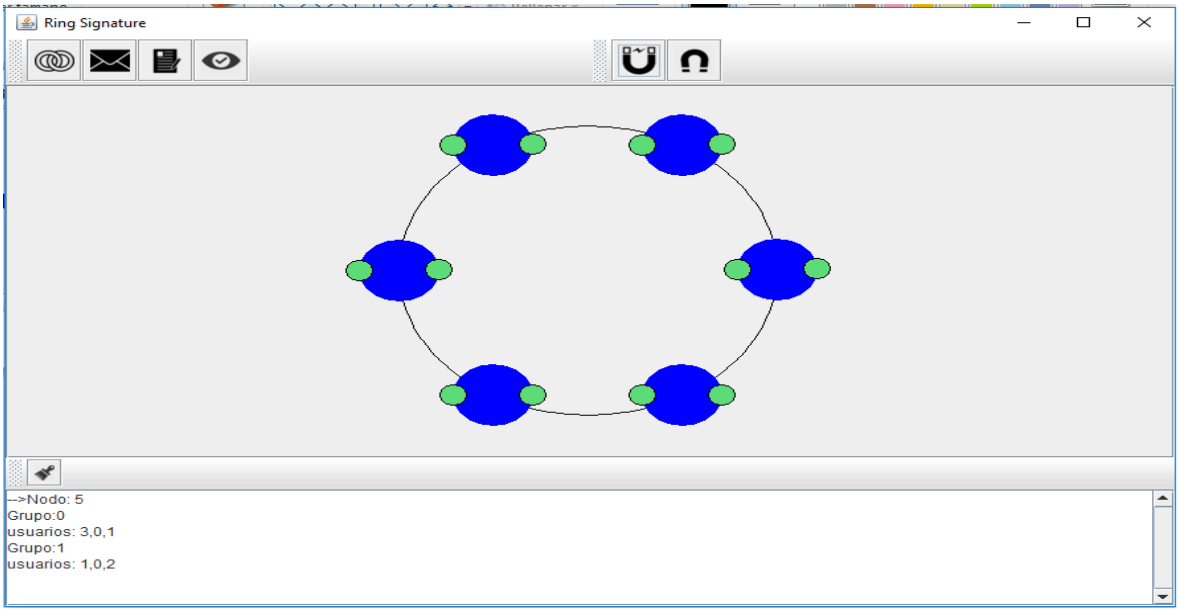
Frame 10: pantalla con la simulación de la firma en anillo mejorada

Selección del botón de firma aplicando su acción en los nodos y explicando en consola el paso del algoritmo.



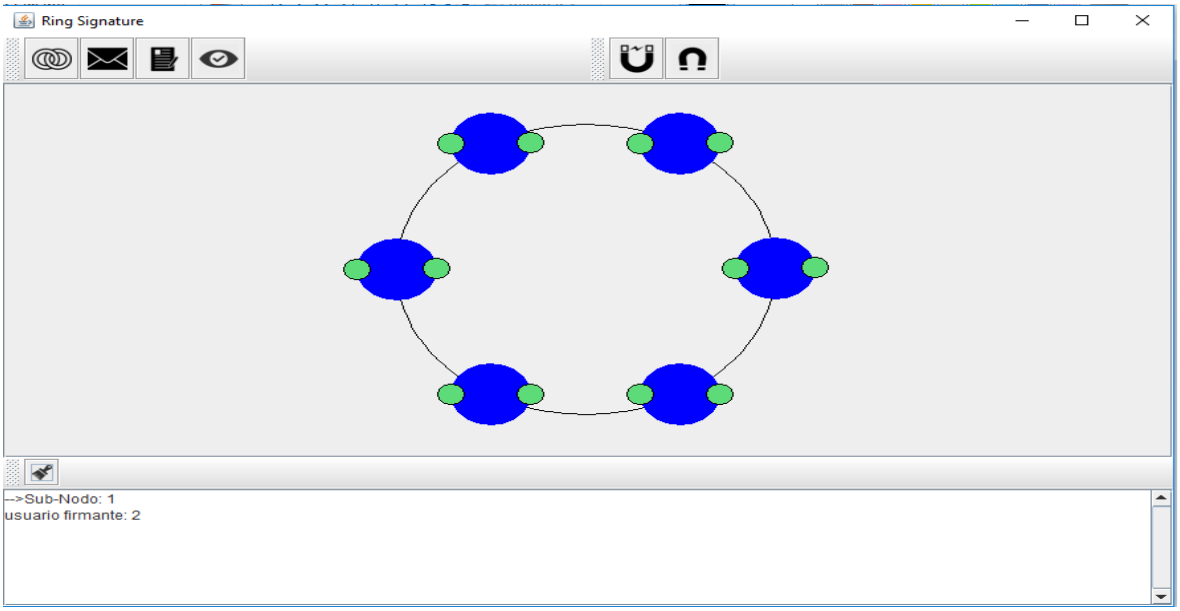
Frame 11: Pantalla con la simulación de la firma en anillo con umbral.

Ventana con la selección del algoritmo de firma en anillo con umbral, seleccionando también el botón para sentar los nodos. También se puede ver que se hizo clic en uno de los nodos, por lo tanto, en consola se muestra información.



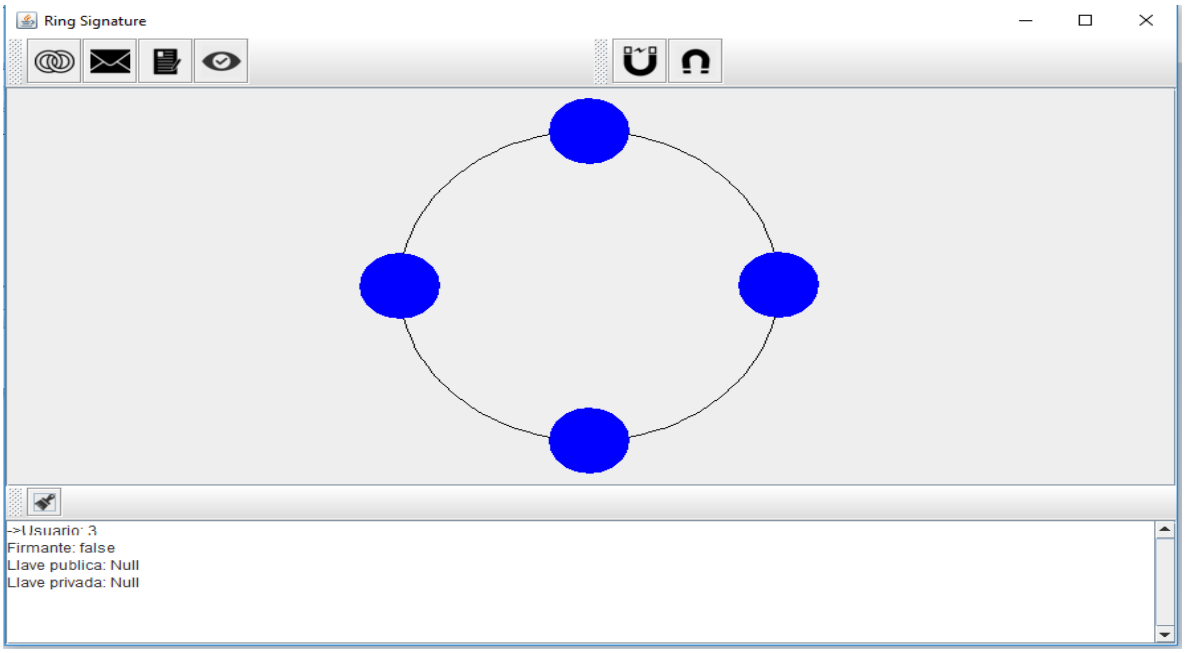
Frame 12: pantalla haciendo clic en uno de los nodos.

Ventana con la selección del algoritmo de firma en anillo con umbral, seleccionando también el botón para sentar los nodos. También se puede ver que se hizo clic en uno de los sub nodos, por lo tanto, en consola se muestra información.



Frame 13: pantalla haciendo clic en uno de los subanillos.

Selección y confirmación del botón 1 explicado anteriormente. Se puede observar que los nodos fueron centrados con el botón respectivo e hizo clic en uno de los nados para conocer un poco más de información.



Frame 14: pantalla haciendo clic en uno de los nodos.

10.2. Navegación

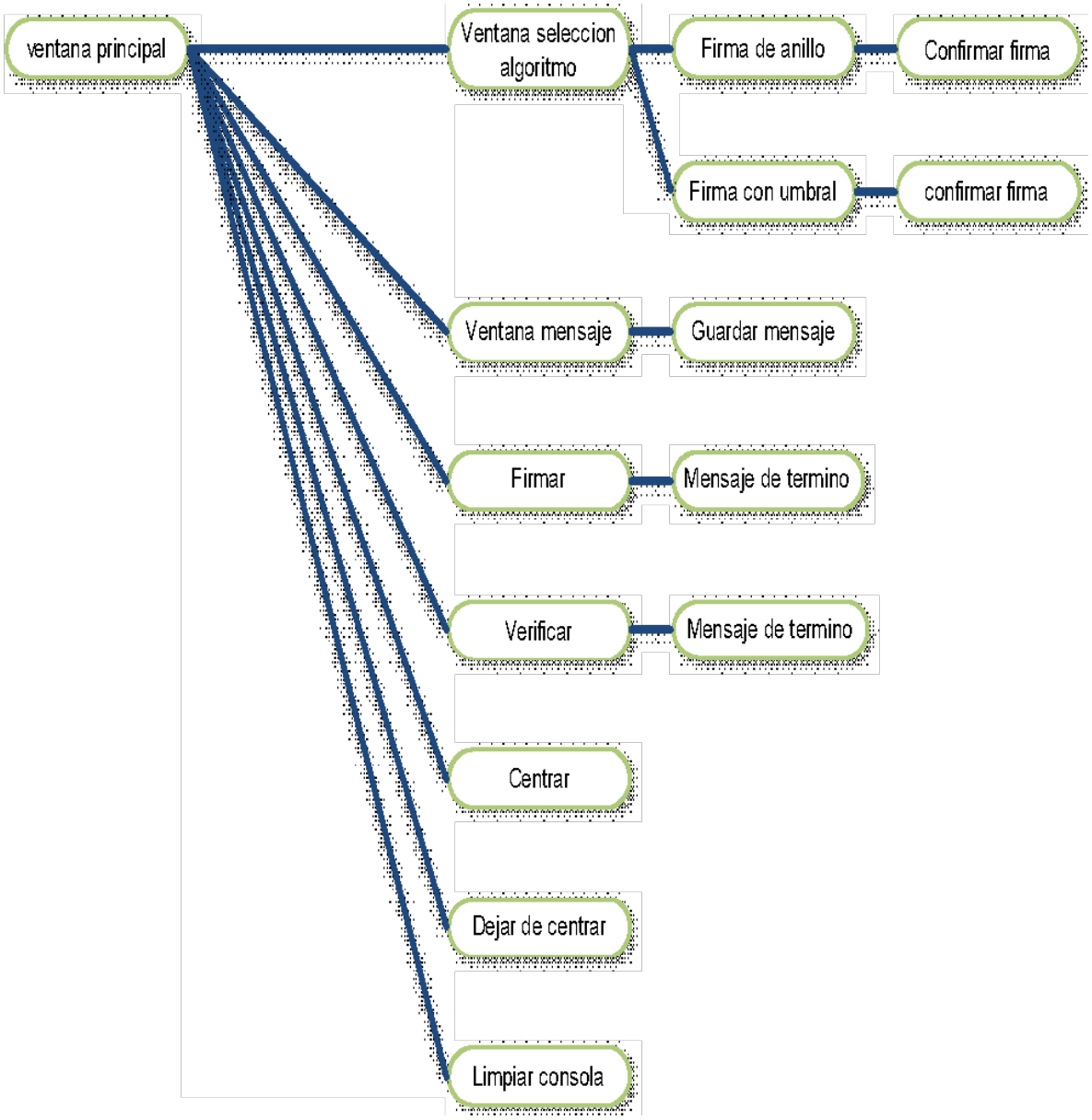


Figura 41: mapa de navegación en el sistema.

10.3. Casos de uso

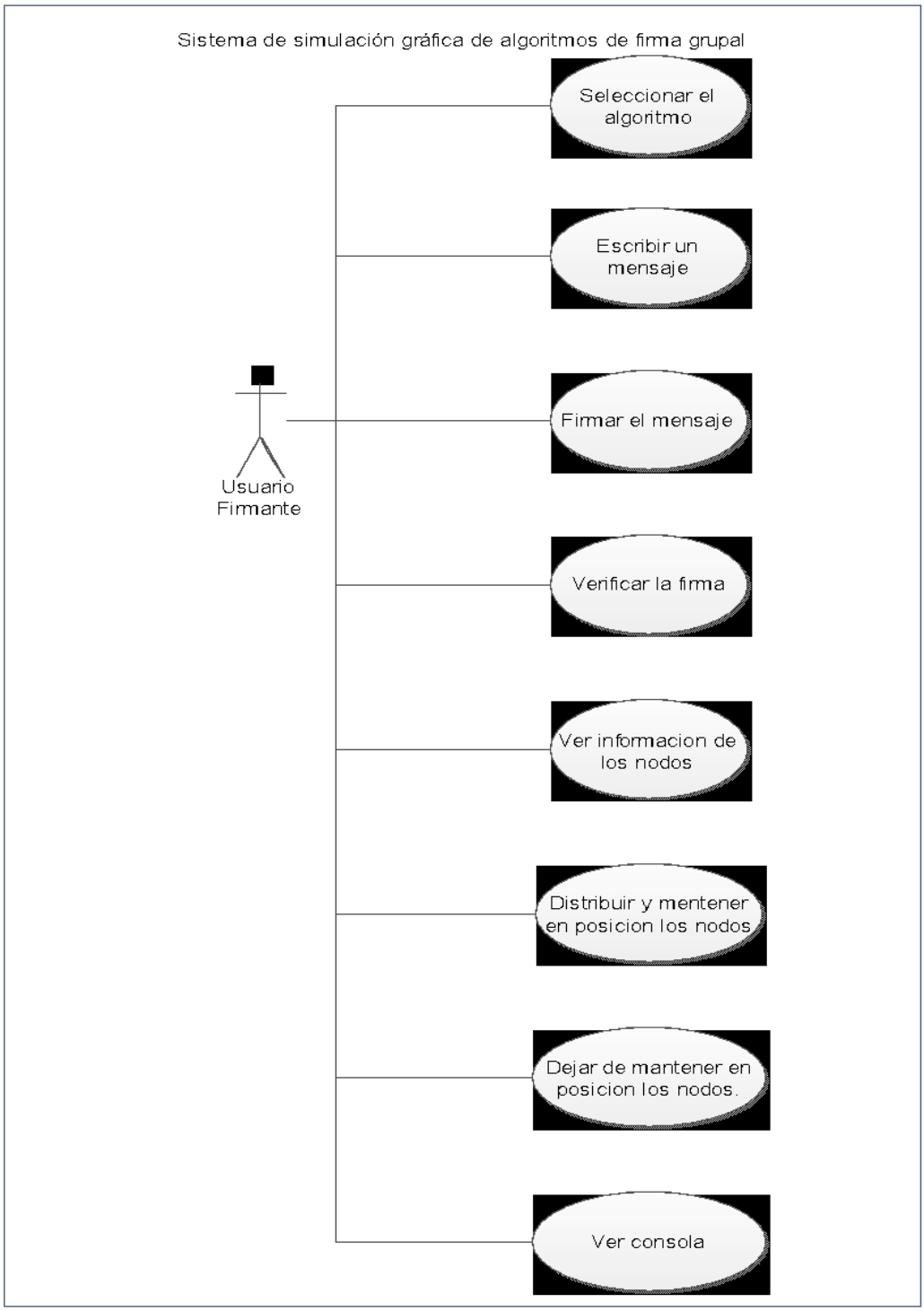


Figura 42: diagrama de casos de uso del sistema.

<table>
<tr><td>Nombre</td><td>Seleccionar el algoritmo</td></tr>
<tr><td>Descripción</td><td>El sistema debe permitir que el usuario seleccione el algoritmo que desea utilizar para firmar un mensaje. Además, el usuario deberá ingresar las características de la firma:
Usuarios que conformar el grupo.
Usuario que actuará como firmante.
En el caso de la firma con umbral, Grupo de usuarios firmantes.</td></tr>
<tr><td>Flujo principal</td><td>
<table>
<tr><td>Actor</td><td>sistema</td></tr>
<tr><td>El usuario oprime un botón para seleccionar el algoritmo a utilizar; firma en anillo o firma con umbral.</td><td>El sistema abre una ventana con los algoritmos que se pueden seleccionar.</td></tr>
<tr><td>El usuario selecciona la firma en anillo como algoritmo de firma grupal.</td><td>El sistema entrega una plana de la ventana para llenar el formulario correspondiente:
Seleccionar el tipo de firma en anillo; simple o mejorada.
Seleccionar la cantidad de usuarios del grupo.
Seleccionar el usuario firmante del grupo.</td></tr>
<tr><td>El usuario selecciona la firma de anillo con umbra como algoritmo de firma grupal.</td><td>El sistema entrega una plana de la ventana para llenar el formulario correspondiente:
Seleccionar la cantidad de usuarios del grupo.
Seleccionar la cantidad de firmantes del grupo.
Seleccionar los usuarios firmantes del grupo.</td></tr>
<tr><td>El usuario confirma, mediante un botón, la configuración del algoritmo para utilizar en la firma grupal.</td><td>El sistema cierra la ventana abierta desde un principio. Luego almacena los datos configurados esperando a que el usuario convoque la firma del mensaje.</td></tr>
</table>
</td></tr>
<tr><td>Flujo alternativo</td><td>
<table>
<tr><td>Actor</td><td>sistema</td></tr>
<tr><td>El usuario confirmar, mediante un botón, la configuración del algoritmo a utilizar, mientras no fue llenado el formulario previo.</td><td>El sistema abre una ventana alternativa de comunicación indicando que parte del formulario debía ser resuelto antes de presionar el botón de confirmación de los datos.</td></tr>
</table>
</td></tr>
<tr><td>Pre condición</td><td>Ninguno.</td></tr>
<tr><td>Post condición</td><td>Después de esto se debe poder visualizar en pantalla la configuración del anillo creado.</td></tr>
</table>

Caso de uso 1

<table>
<tr><td>Nombre</td><td>Ver información de los nodos</td></tr>
<tr><td>Descripción</td><td>El sistema debe permitir mostrar de manera gráfica los nodos creados en la etapa de selección de los algoritmos y su posterior formulario.</td></tr>
<tr><td>Flujo principal</td><td>
<table>
<tr><td>Actor</td><td>sistema</td></tr>
<tr><td>El usuario confirma los datos del formulario del proceso de selección del algoritmo.</td><td>El sistema dibuja los nodos, representados con un círculo, del color correspondiente a su grado:
Nodo: color azul.
Sub anillo: color verde.
Súper anillo: borde color negro, color de fondo igual al panel en que se encuentra.</td></tr>
<tr><td></td><td>El sistema ubica los círculos dibujados en el centro del panel en que son dibujados.</td></tr>
<tr><td>El usuario mantiene oprimido el botón derecho sobre un circulo.</td><td>El sistema cambia de color al nodo a celeste.</td></tr>
<tr><td>El usuario suelta el botón derecho sobre un circulo.</td><td>El sistema cambia el nodo a su color original.</td></tr>
<tr><td>El usuario oprime el botón izquierdo sobre un nodo.</td><td>El sistema entrega información del nodo oprimido.</td></tr>
</table>
</td></tr>
<tr><td>Flujo alternativo</td><td>
<table>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td>El usuario mantiene oprimido clic derecho sobre un nodo y arrastra el mouse.</td><td>El sistema mueve el nodo hasta donde es movido el nodo.</td></tr>
</table>
</td></tr>
<tr><td>Pre condición</td><td>El usuario debe seleccionar y completar el formulario de la firma digital.</td></tr>
<tr><td>Post condición</td><td>El usuario debe poder ver dibujados en el centro de la pantalla todos los nodos creados en el proceso de selección del algoritmo.</td></tr>
</table>

Caso de uso 2

<table>
<tr><td>Nombre</td><td>Distribuir y mantener en posición los nodos.</td></tr>
<tr><td>Descripción</td><td>El sistema debe permitir centrar en una ubicación específica los nodos, esa ubicación debe ser en la orilla de un súper anillo, provocando una idea de atracción de un imán, es decir cuando el nodo es arrastrado este vuelve a su posición inicial.</td></tr>
<tr><td>Flujo principal</td><td>
<table>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td>El usuario oprime el botón de atracción de los nodos</td><td>El sistema crea un súper anillo en el centro y los nodos son ubicados en distintas posiciones dentro del supera anillo. Los sub anillos son ubicados en distintas posiciones dentro del nodo correspondiente.</td></tr>
<tr><td>El usuario arrastra un nodo o sub anillo.</td><td>El sistema reubica el nodo en su posición correspondiente.</td></tr>
</table>
</td></tr>
<tr><td>Pre condición</td><td>El usuario debe seleccionar el algoritmo de firma grupal y tiene que haber llenado los datos del formulario correspondiente.</td></tr>
<tr><td>Post condición</td><td>Los nodos siempre serán atraídos a su posición correspondiente.</td></tr>
</table>

Caso de uso 3

<table>
<tr><td>Nombre</td><td>Dejar de mantener en posición los nodos.</td></tr>
<tr><td>Descripción</td><td>El sistema debe permitir que la atracción predispuesta al súper anillo termine, es decir que cuando el nodo se quiera sacar de su posición este se mantenga en donde queda.</td></tr>
<tr><td>Flujo principal</td><td>
<table>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td>El usuario oprime el botón para acabar la atracción al súper anillo.</td><td>El sistema detiene el proceso de distribución y mantenimiento de la posición del nodo.</td></tr>
</table>
</td></tr>
<tr><td>Pre condición</td><td>El usuario debe activar el proceso de distribución y mantenimiento de la posición del nodo.</td></tr>
<tr><td>Post condición</td><td>El usuario puede mover libremente el nodo que seleccione.</td></tr>
</table>

Caso de uso 4

<table>
<tr><td>Nombre</td><td>Ver consola</td></tr>
<tr><td>Descripción</td><td>El sistema debe permitir al usuario ver información en una consola. Esta información es enviada por el sistema y además el usuario puede limpiar la consola de información.</td></tr>
<tr><td>Flujo principal</td><td>
<table>
<tr><td>Actor</td><td>sistema</td></tr>
<tr><td>El usuario hace lectura de la información en la consola.</td><td>El sistema recibe información de los otros procesos.</td></tr>
<tr><td>El usuario oprime el botón para limpiar la información de la consola.</td><td>El sistema borra la información de la consola que es enviada por los procesos del sistema.</td></tr>
</table>
</td></tr>
<tr><td>Pre condición</td><td>El sistema quiere comunicar alguna información al usuario.</td></tr>
<tr><td>Post condición</td><td>Ninguna.</td></tr>
</table>

Caso de uso 5

<table>
<tr><td>Nombre</td><td>Escribir un mensaje</td></tr>
<tr><td>Descripción</td><td>El sistema debe permitir al usuario escribir un mensaje a firmar, en el cual no exista un límite de caracteres ni limitaciones de símbolos. El sistema debe permitir almacenar el mensaje mientras el usuario firma el mensaje.</td></tr>
<tr><td>Flujo principal</td><td>
<table>
<tr><td>Actor</td><td>sistema</td></tr>
<tr><td>El usuario oprime un botón de mensaje.</td><td>El sistema abre una ventana alternativa que permite al usuario escribir un mensaje de forma libre.</td></tr>
<tr><td>El usuario termina el mensaje y oprime un botón para guardar el mensaje.</td><td>El sistema guarda el mensaje escrito por el usuario, esperando que este sea usado en el proceso de firma.</td></tr>
</table>
</td></tr>
<tr><td>Pre condición</td><td>Ninguno.</td></tr>
<tr><td>Post condición</td><td>El mensaje configurado continuará siendo al mismo, dentro del sistema, hasta que el usuario vuelva a cambiarlo todo, es decir, si se vuelve a configurar un nuevo anillo el mensaje seguirá siendo el mismo hasta que el usuario lo edite.</td></tr>
</table>

Caso de uso 6

<table>
<tr><td>Nombre</td><td>Firmar el mensaje</td></tr>
<tr><td>Descripción</td><td>El sistema debe permitir al usuario generar una firma digital para el mensaje y el algoritmo anteriormente configurado por el mismo. También es necesario que el sistema almacene los datos de la firma esperando por el proceso de verificación.</td></tr>
<tr><td>Flujo principal</td><td>
<table>
<tr><td>Actor</td><td>sistema</td></tr>
<tr><td>El usuario oprime el botón para firmar el mensaje.</td><td>El sistema concentra los datos del algoritmo configurado con posterioridad para llamar al algoritmo correspondiente. Luego se comienza con el proceso de firma.</td></tr>
<tr><td></td><td>El sistema comienza a realizar una simulación visual del proceso de la firma, sobre el anillo creado en la ventana principal.</td></tr>
<tr><td></td><td>El sistema comienza a mostrar datos del proceso de firma, de manera escrita.</td></tr>
<tr><td></td><td>El sistema avisa del término de la firma. También muestra si la firma fue o no correcta.</td></tr>
<tr><td>El usuario acepta la firma generada.</td><td>El sistema almacena la firma producida, en espera de que el usuario active el método de verificación.</td></tr>
</table>
</td></tr>
<tr><td>Pre condición</td><td>Se debe contar a lo menos con la configuración del algoritmo a utilizar, el mensaje no es necesario ya que puede existir la posibilidad de que se quiera realizar una firma para un mensaje en blanco.</td></tr>
<tr><td>Post condición</td><td>Se debe contar con una firma almacenada, la cual es utilizada por el proceso de verificación de la firma.</td></tr>
</table>

Caso de uso 7

<table>
<tr><td>Nombre</td><td>Verificar la firma</td></tr>
<tr><td>Descripción</td><td>El sistema debe permitir verificar la firma del mensaje que fue escrito. A la vez de comenzar con el proceso de verificación según el algoritmo configurado con anterioridad, también mantiene una comunicación escrita con el usuario mostrando datos relevantes.</td></tr>
<tr><td>Flujo principal</td><td>
<table>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td>El usuario oprime el botón de verificación.</td><td>El sistema obtiene la configuración del algoritmo seleccionado. Luego obtiene la información de la firma y el mensaje almacenado por el usuario.</td></tr>
<tr><td></td><td>El sistema comienza a mostrar de forma escrita el proceso de firma.</td></tr>
</table>
</td></tr>
<tr><td>Pre condición</td><td>El usuario debe seleccionar un algoritmo de firma grupal y además firmar el mensaje.</td></tr>
<tr><td>Post condición</td><td>Ninguna.</td></tr>
</table>

Caso de uso 8

11. Conclusión

Este proyecto de título alcanzo los objetivos específicos pre dispuestos en las secciones anteriores, los cuales corresponden a:

- Realizar un recorrido empírico e implementar los algoritmos de firma digital grupal propuestos en el documento [3].
- Diseñar e implementar mediante métodos de prototipo una interfaz gráfica para la simulación del algoritmo de firma digital grupal con anonimato.

Para lograr el primer objetivo general del proyecto, fue necesario revisar la literatura, es decir, entender los conceptos previos de las firmas digitales, en donde su característica fundamental se centra en el uso de una llave privada y una pública. Finalmente, mediante la utilización del conocimiento acumulado, se comenzó el diseño e implementación de los algoritmos de firma grupal basado en funciones de HASH y RSA, también el algoritmo de firma grupal con umbral.

El recorrido empírico que se realizó, tuvo como consecuencia un importante avance en el conocimiento de las firmas digitales, es por eso que, a partir de este hecho, se abre un camino intelectual que permitirá seguir investigaciones de la misma línea.

Hallazgo más importante

En el presente trabajo, en el que implementaron diversos algoritmos de firma digital con garantías de anonimato, se realizó uno hallazgo importante, que está relacionado con el funcionamiento del algoritmo. El algoritmo de firma digital grupal con garantías de anonimato depende de la composición de los grupos que lo componen, por lo que fue necesario crear una clase que creara grupos de distintos usuarios a partir de un conjunto de ellos, lo cual es implementado en una pequeña parte, ya que para ello fue es necesario construir un algoritmo de carácter determinístico. En el proyecto se implementa un protocolo en el que los grupos son representados por determinados datos almacenados en un archivo (txt).

12. Referencias

[1] M. Chamizo, “Evaluación e implementación en Java de un algoritmo de firma grupal digital para preservar el anonimato en redes ad-hoc inalámbricas”, Proyecto de Titulación de alumno de Intercambio, Universidad Del Bío-Bío, 2016.

[2] Emmanuel Bresson, Jacques Stern, Michael Szydlo, “Threshold Ring Signature and Applications to Ad-hoc Groups”, Crypto’02, Santa Barbara, California.

[3] Ron Rivest, Adi Shamir, Yair Tauman, “How to leak a secret”, Asiacrypt’01

[4] P. P. Tsang, M. Au, J. K. Liu, W. Susilo, D. S. Wong, “A Suite of Non-Pairing ID-Based Threshold Ring Signature Schemes with Different Levels of Anonymity”, Cryptology ePrint Archive, PDF 326, 2005.

[5] Object Aid UML Explorer - Link: http://www.objectaid.com/download, Object Aid LLC is based in Atlanta, GA, USA.