Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

Dpto. Matemática Aplicada 1

Trabajo Fin de Grado

Descripción, análisis e implementación de un protocolo autenticado de transferencia de claves en grupo basado en la compartición de secretos.

Grado en Ingeniería Informática - Tecnologías Informáticas

Realizado por:
Marco Herrero Serna

Dirigido por:
José Andrés Armario Sampalo

Sevilla, 1 de septiembre de 2014

Resumen / Abstract

En este TFG se ha realizado un estudio y una posterior implementación de un protocolo de acuerdo de clave en grupo. Las claves son generadas y distribuidas por un KGC (centro generador de claves) usando técnicas de compartición de secretos. Se ha implementado también una interfaz gráfica que permite visualizar los mensajes intercambiados durante el protocolo.

In this TFG has been conducted a study and subsequent implementation of a group key agreement protocol. These keys are generated and distributed by a KGC (key generation center) using sharing secrets techniques. A graphic interface has been implemented in order to visualize the messages exchanged in this protocol.

Índice general

Índice de figuras

Índice de código

CAPÍTULO 1

Introducción

En la era de las tecnologías de la información es necesario disponer de métodos para garantizar la seguridad de las comunicaciones que se producen. A lo largo de la historia la criptografía ha servido para cumplir este cometido, permitiendo garantizar la confidencialidad de la información en las comunicaciones.

La confidencialidad representa uno de los objetivos principales de la comunicaciones seguras. Se asegura de que la información solo sea accesible para las partes autorizadas, logrando este objetivo a través del cifrado de la misma. En el caso de la criptografía simétrica, dicha información es cifrada usando una clave secreta que comparten tanto el emisor como el receptor. Suponiendo que un sistema es seguro, una entidad que no posea dicha clave será incapaz de descifrar la información, quedando esta oculta para cualquier participante no autorizado.

Si los sistemas criptográficos actuales basan su seguridad en la fortaleza de la clave, la generación y distribuición de la misma es un paso clave para dicha seguridad. En este TFG se analizará este problema, y se propondrá una solución al mismo.

Conceptos básicos como criptografía simétrica serán definidos en los siguientes capítulos, sentando las bases para la descripción de un protocolo de acuerdo de clave en grupo.

CAPÍTULO 2

Análisis temporal y costes de desarrollo

Para poder gestionar la planificación de un proyecto, lo primero a desarrollar será el *Project Charter*. Es un documento donde se delimitará el alcance del proyecto, se definirán los objetivos, se establecerán los entregables y sus plazos, así como los roles y responsabilidades de los participantes. Será el documento de referencia durante la duración del mismo, consultándolo cuando sea necesario.

2.1– Project Charter

Trabajo de fin de grado, curso 2013/2014.

Las personas implicadas en estre proyecto son:

- Marco Herrero Serna - Autor del proyecto.
- José Andrés Armario Sampalo - Tutor

Está previso que se dediquen aproximadamente 360 horas de trabajo a este proyecto por parte del alumno, siendo el equivalente a 12 créditos ECTS (teniendo 30 horas de trabajo cada crédito). Durante la realización del proyecto se efectuarán unas sesiones de control en las cuales se analizará el trabajo realizado y la dirección del mismo.

2.1.1. Descripción y motivación del proyecto

El trabajo de fin de grado TFG es necesario para completar los estudios en el Grado de Ingenería Informática en Tecnologías Informáticas. Consiste en el desarrollo de un proyecto informático de un tema elegido por el alumno con la colaboración de un tutor, donde se pongan en uso

los conocimientos adquiridos durante el grado.

Las necesidades del proyecto son la elección del tema, investigación del mismo para encontrar una línea de trabajo, implementación de un software, creación de una memoria de desarrollo y defensa del trabajo ante un tribunal.

Después de la fase de investigación del tema se ha llegado al acuerdo de que el TFG esté orientado al estudio e implementación de un acuerdo de intercambio de claves en grupo basado en la compartición de secretos tal y como se explica en el artículo [Sun12]. Por lo tanto se realizará un estudio teórico sobre este tema y una implementación de dicho acuerdo de intercambio de claves.

2.1.2. Productos finales a generar

Al final del proyecto serán generados cuatro productos que se entregarán al correspondiente tribunal para su evaluación. Dichos productos son:

1. **Memoria del TFG:** Documento que relata el desarrollo del TFG, incluyendo el estudio del tema, desarrollo de los productos a generar, fundamentos teóricos y experimentación.

2. **Implementación del protocolo:** Biblioteca que implementa el intercambio de clave de manera modular, permitiendo su uso en diferentes estructuras y sistemas sin que sea necesario modificar el código. También debe permitir su posterior ampliación.

3. **Aplicación de escritorio didáctica:** Software de escritorio que mostrará el proceso de intercambio de clave y permitira a un usuario examinar los diferentes mensajes enviados. Esta aplicación hará uso de los recursos de la bibloteca de intercambio de clave anteriormente citada.

4. **Presentación:** Diapositivas tipo *Powerpoint* que se usarán el día de la defensa del trabajo.

2.1.3. Puntos de control y cronograma

El proyecto será realizado desde el 10 de noviembre de 2014 hasta el 15 de septiembre de 2014. y constará de cinco fases, cada una correspondiente a un objetivo. El fin de cada uno de los puntos de control coincidirá con una evaluación del trabajo realizado hasta ese momento.

Puntos de control

- **Punto de control 1:** Elección de un tema sobre el que desarrollar el trabajo.

  Investigación sobre las diferentes opciones que ofrece el tutor del proyecto y elección de una de ellas. Hay que tener en cuenta que el artículo debe proporcionar la posibilidad de desarrollar un TFG completo.

  Evaluación del punto de control: 20 de enero de 2014

- **Punto de control 2:** Investigación en base a un artículo proporcionado por el tutor.

  Compresión y búsqueda de información en base a dicho artículo.

  Evaluación del punto de control: 24 de marzo de 2014

- **Punto de control 3**: Diseño e implementación de un programa que ejecute el protocolo citado en ese artículo.

  Implementar un conjunto de clases que modelicen el comportamiento de dicho protocolo. Implementar la comunicación de mensajes de dicho protocolo. Crear una interfaz para la comprensión didáctica del mismo.

  Evaluación del punto de control: 30 de julio de 2014

- **Punto de control 4:** Creación de una memoria

  Redacción de una memoria que incluya una introducción al TFG, planificación, conceptos teóricos y documentación.

  Evaluación del punto de control: 30 de agosto de 2014

- **Punto de control 5:** Exposición del trabajo

  Creación de una presentación en powerpoint y preparación de la defensa, que posteriormente se expondrá ante un tribunal

  Evaluación del punto de control: 10 de septiembre de 2014

2.2– Análisis temporal

En esta sección se analizará el tiempo planificado para la resolución de cada punto de control, así como el tiempo finalmente dedicado. Como se calculó al inicio de esta sección, el TFG tiene 360 horas de trabajo, que serán distribuidas entre los diferentes puntos de control. Las horas asignadas a un punto de control serán las horas trabajadas desde el anterior punto de control (o el principio del proyecto) hasta la fecha límite.

Tarea	Estimación inicial	Tiempo final
Elección del tema	36 horas	36 horas
Investigación del artículo	36 horas	50 horas
Implementación	144 horas	150 horas
Creación de una memoria	108 horas	140 horas
Exposición del trabajo	36 horas	No finalizado

Cuadro 2.1: Planificación del proyecto

La planificación temporal consistirá en una estimación del tiempo dedicado a cada una de las fases del proyecto, como se muestra en el cuadro 2.1 Para ello asignaremos a cada una de esas fases las dos siguientes estimaciones:

1. **Estimación inicial**, la cual se usará al principio del proyecto para estimar la cantidad de trabajo necesaria para la realización de los objetivos de un punto de control.

2. **Tiempo final**, la cual indica la cantidad de horas reales trabajadas antes de cada punto de control.

Nota: El tiempo dedicado al último punto de control relativo a la exposición del trabajo no ha sido contado, ya que la fecha de realización de esta memoria es anterior a la exposición del trabajo.

2.3– Análisis de recursos

Para el desarrollo de este proyecto, se tendrán en cuenta los costes relacionados con software, hardware, personal y costes indirectos.

2.3.1. Costes de software

A la hora de desarrollar el proyecto, se ha optado por usar mayoritariamente software libre, o en caso de no encontrar alternativas adecuadas, software con coste cero, para reducir al mínimo el presupuesto relacionado con el software de este proyecto. A continuación se indica el software usado durante el proyecto:

- El sistema operativo usado es Mac OS X - Mavericks, que no es software libre, pero es gratuito, así como muchas de sus utilidades.
- TeXShop como editor LaTeX de la memoria
- La clase `pclass` de LaTeX, desarrollada en esta escuela.
- MacVim como editor de textos y entorno de desarrollo

- wxPython como librerías de desarrollo de entornos gráficos
- PyInstaller como empaquetador de aplicaciones de escritorio
- VirtualBox para la virtualización de diferentes sistemas operativos.

2.3.2. Costes de hardware

Todo el proyecto ha sido desarrollado en un ordenador portátil Macbook Pro, de 1265 euros. Se ha contado también con un disco duro Seagate de 500GB para las copias de seguridad, a un precio de 85 euros.

2.3.3. Costes indirectos

Dentro de los costes indirectos se incluirán todos los gastos producidos por material consumible, luz, limpieza, etc. Se ha considerado representativo un aumento del 5 % del presupuesto total para cubrir los costes indirectos.

2.3.4. Costes de personal

El coste de personal se ha calculado usando el BOE, del 30 de diciembre de 2014, en el cual se publicó el Real Decreto 1046/2013, de 27 de diciembre, en el que se establecía que el salario mínimo 21,51 euros/día, o lo que es lo mismo, 2,63 euros/hora, teniendo en cuenta una jornada laboral completa de 8 horas.

Dado que la realización de este proyecto requiere conocimientos especializados, se ha decidido que el coste por trabajador sea 2 veces el salario mínimo, es decir 5,38 euros/hora.

Teniendo el cuenta que el proyecto tiene planificadas 360 horas de trabajo para un solo desarrollador, el coste de personal será de 1936 euros.

2.3.5. Presupuesto preliminar del proyecto

En el cuadro 2.2 se han resumido los costes del proyecto, resultando en un coste económico de 3450,3 euros. Este es un coste ficticio, puesto que al ser un proyecto académico, los objetivos no se corresponden con una actividad profesional, si no con una simulación de la misma.

Costes Software	0 euros
Costes Hardware	1350 euros
Costes Personal	1936 euros
Costes Indirectos	164,3 euros
TOTAL	3450,3 euros

Cuadro 2.2: Presupuesto del proyecto

CAPÍTULO 3

Conceptos básicos

En este capítulo se describirán algunos conceptos básicos que serán usados más tarde para la descripción del problema y la solución presentada.

3.1– Criptografía de clave pública vs simétrica

Los algoritmos de cifrado existentes en la actualidad se pueden dividir en dos grupos, los de clave simétrica y los de clave asimétrica, también llamados de clave pública. Todos los sistemas clásicos de cifrado son simétricos, no apareciendo sistemas asimétricos hasta finales del siglo XX [Sin00]

3.1.1. Criptografía simétrica

La criptografía simétrica cifra la información usando una clave secreta que comparten tanto el emisor como el receptor. Suponiendo que un sistema es seguro, una entidad que no posea dicha clave será incapaz de descifrar la información, quedando esta oculta para cualquier participante no autorizado.

La fortaleza de un sistema debe depender en exclusiva de la fortaleza de la clave usada. Confiar en que nadie descubra el algoritmo de cifrado es un error, conociéndose esta técnica como *seguridad por oscuridad*. El principio de Kerckhoff's dice que un sistema criptográfico debe ser igual de seguro si toda la información sobre el sistema, menos la clave, es hecha pública. Si dependemos únicamente de la fortaleza de la clave, esta se convierte en un punto crítico para cualquier sistema.

Una de las grandes desventajas de este sistema es que es complicado establecer un sistema de distribución y gestión de claves eficiente entre emisor y receptor. Este será uno de los problemas que se tratará en este trabajo, la distribución de claves entre participantes.

3.1.2. Criptografía asimétrica

La criptografía de clave pública, por otro lado, cifra la información con un par de claves, que se denominarán pública y privada. Ambas claves son generadas por el receptor del mensaje, que distribuirá la pública y guardará la otra en privado. El emisor cifrará el mensaje con la clave pública, el cual sólo podrá ser descifrado con la clave privada que guarda el receptor.

Esto soluciona el problema de distribución de claves, pues el receptor no tiene que distribuir la clave de descifrado en ningún momento, mientras que la clave pública no ofrece la posibilidad de comprometer la confidencialidad del mensaje.

El uso de un par de claves, pública y privada, evita la necesidad de distribuir una clave por un canal seguro. Sin embargo, los algoritmos criptográficos asimétricos requieren más cálculos que los simétricos, haciendolos más lentos cuando se desea cifrar una gran cantidad de información.

Este trabajo se centrará solo en el uso de criptografía simétrica, pues la clave que se generará y distribuirá será compartida por todos los participantes del protocolo.

3.2– Secret sharing: Compartición de secretos

El *secret sharing* o compartición de secretos es una clase de métodos creados para distribuir un secreto, entendido generalmente como un número o trozo de información, entre un grupo de participantes. Cada uno de estos participantes obtendrá una división de este secreto. El secreto en su totalidad sólo podrá ser averiguado si varios de los participantes ponen sus divisiones en común. Una división es inservible por si sola. [Sha79]

Por ejemplo, si como secreto queremos acordar un punto en el espacio, con sus tres coordenadas, podemos asginar a cada participante una división de este secreto en forma de plano. Cada plano que asignemos a un participante pasará por este punto, pero un plano por si sólo es incapaz de mostrar cual es el punto acordado. Si dos usuarios combinan sus planos,

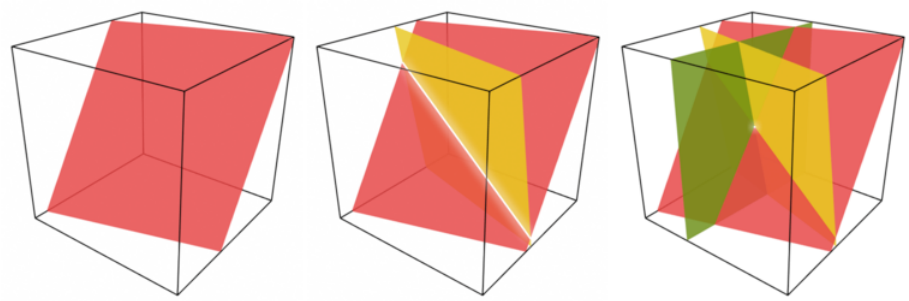

Figura 3.1: Secret sharing en el espacio

conseguirán una recta, información insuficiente para recuperar el secreto. Al combinar los planos de tres participantes estos pueden recuperar el punto original, como se muestra en la figura 3.1

Como se ha comentado, una división por si sola, en este caso un plano, es incapaz de aportar información suficiente para revelar el secreto elegido. Hasta que no se une a una serie de participantes con sus planos, no se puede desvelar el secreto.

3.3– Generación de claves

Para la generación de claves usamos el siguiente esquema:

$$g^x \ mod \ p = k \tag{3.1}$$

dónde a g lo llamamos generador del grupo G. Un generador g, trabajando en módulo p, debe ser capaz de alcanzar cualquier elemento del grupo G al ser elevado a cualquier número x. La función que genera la clave, por tanto, será una función $f(x) = g^x \ mod \ p$, y dependerá de los valores g y p que se elijan.

La elección de unos valores g y p es un paso crucial a la hora de generar una clave, pues el espacio de claves a generar depende de la relación entre estos dos valores. Por ejemplo, si trabajamos *mod* 9, diremos que un elemento g es un mejor generador si genera todos los primos relativos a 9. De esta manera, 7 no será un generador, ya que

$$\{7^x \ mod \ 9 \mid i \in \mathbb{N}\} \ = \ \{7, 4, 1\} \tag{3.2}$$

mientras que 2 sí es un generador, ya que genera todos los coprimos con 9.

$$\{2^x \ mod\ 9 \mid i \in \mathbb{N}\} \ = \ \{2, 4, 8, 7, 5, 1\} \tag{3.3}$$

3.3.1. El problema del logaritmo discreto

Sea G un grupo con un generador g. El problema del logaritmo discreto para el grupo G se puede definir de la siguiente manera: Dado $g \in G$ y $a \in G$, encontrar un entero x tal que $g^x = a$ es un problema computacionalmente intratable [McC90]

Esto se conoce como *función de un sentido*. Este tipo de funciones son aquellas que se resuelven de una manera simple, pero la función inversa es computacionalmente intratable. Esta propiedad se usará para garantizar la seguridad de las operaciones usadas por los participantes en un intercambio de clave.

3.4– Acuerdo de clave

Los conceptos anteriormente definidos van a ser usados para la definición del problema del acuerdo de clave, así como para la solución presentada. Para un conocimiento más profundo de estos conceptos, se recomienda una lectura de los artículos que se han listado en la bibliografía.

CAPÍTULO 4

Definición del problema: Acuerdo de clave

En la comunicación en grupo, es necesario que todos los participantes acuerden una clave antes de comenzar la misma. Esto permitirá proteger la confidencialidad de la información a través de algoritmos de cifrado. En este caso, nos centramos en los algoritmos de cifrado simétricos, ya que la clave que se va a distribuir será la misma para todos los participantes.

4.1– Acuerdo de clave en grupo

Dado una serie de participantes en una comunicación, se trata de generar y distribuir una clave entre todos ellos, de tal manera que esta esté disponible para cualquier participante autorizado. Del mismo modo, cualquier participante no autorizado no tendrá acceso a la misma.

Existen dos tipos de protocolos de acuerdo de clave en grupo, los acuerdos de clave distribuidos y los acuerdos de clave centralizados, dependiendo de los elementos que se usen para este acuerdo.

4.1.1. Acuerdos de clave distribuidos

En los acuerdos de clave distribuidos, todos los miembros del grupo deben contribuir a la generación y la distribución de la clave de sesión. No hace falta confiar en otros participantes, como un centro de claves, para el acuerdo de la misma.

Este esquema reduce la dependencia en otros actores, evitando tener que confiar en terceras partes para la generación o distribución de la clave. Por otra parte, la cantidad de operaciones que debe realizar cada participante se verá incrementada, pues no solo debe generar su parte, si no verificar la de los demás.

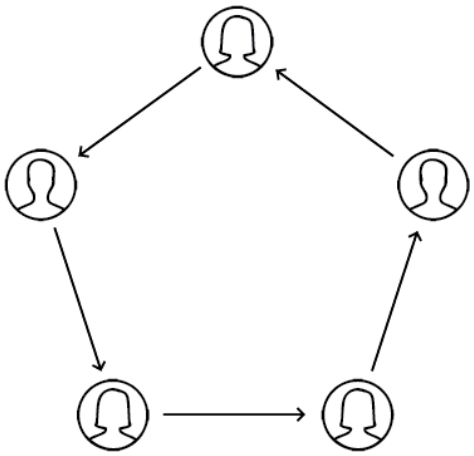

Figura 4.1: Diagrama de comunicación en un acuerdo de clave con una distribución en anillo

4.1.2. Soluciones existentes distribuidas

Aquí se enumeran algunas soluciones existentes para el acuerdo de clave en grupo de forma distribuida. Para más información sobre estos protocolos, así como otros del mismo tipo consultar [CS05]

1. Distribución en anillo. En esta solución, la cooperación de los miembros del grupo forma un anillo. Si existen n participantes, cada miembro M_i se comunica con el M_{i+1}, excepto el M_n, que se comunica con el M_1, cerrando el anillo, como se muestra en la figura 4.1

   Los participantes generan la clave en (n-1) rondas. En la ronda inicial, cada miembro M_i genera un valor aleatorio N_i, calcula g^{N_i} y se lo envía al miembro siguiente. En cada ronda consecutiva, cada miembro eleva a N_i el valor intermedio que ha recibido. Al final, cada miembro tendrá la clave de grupo $K_g = g^{N_1,N_2,...,N_n}$

2. El protocolo OCTOPUS está basado en Diffie-Hellman (DH). El grupo se divide en 4 subgrupos, cada cual acuerda internamente un valor DH: $I_{subgrupo} = \alpha^{u_1,u_2,..,u_n/4}$, donde u_i es la contribución del usuario i. Después, los cuatro subgrupos intercambian los valores DH, pudiendo calcular la clave final K_g

3. Fiat y Naor especificaron un protocolo distribuido que sí requería un centro generador T. Sigue siendo un protocolo distribuido porque en la generación de la clave solo intervienen los participantes. Al iniciarse el protocolo, T elige dos primos q_1 y q_2, y envía a todos

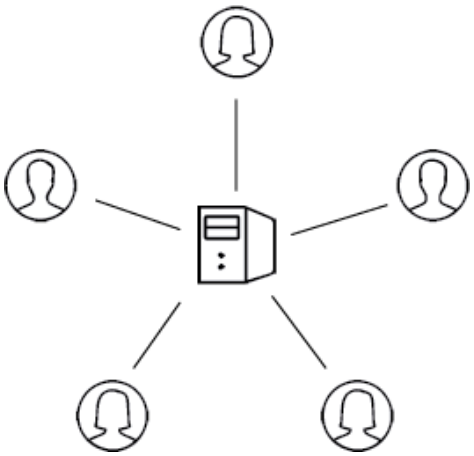

Figura 4.2: Intercambio de mensajes en un acuerdos de clave centralizado

los particiantes el valor $p = q_1 q_2$. Despues, T genera un número aleatorio g y lo guarda como secreto.

Cuando un nuevo miembro M_i se une al grupo, T le envía dos valores: un número aleatorio x_i y una clave $\alpha_i = g^{x_i}$ mod p. Para acordar una clave, cada miembro envía a los demás su valor x_i, y calcula $K = g^{x_1 x_2 \ldots x_n}$ mod p.

Este método no es seguro contra ataques de colisiones y, como ya se ha comentado, depende de una entidad T en la que hay que confiar.

Una de las desventajas de los protocolos distribuidos es que en el momento en el que uno de los participantes no se encuentra disponible, el resto de participantes debe empezar el protocolo de nuevo, ya que la clave no puede ser generada sin dicho participante. Esto se debe a que todos los participantes deben realizar operaciones para generar la clave, distribuirla y verificarla.

4.1.3. Acuerdos de clave centralizados

En los acuerdos de clave centralizados, se confía la generación y distribución de esta clave a un centro de generación de clave (KGC). Con la ayuda del KGC, se reduce la cantidad de mensajes que deben enviarse,y se elimina la necesidad de que sean los participantes los que generen la clave.

En este caso, los participantes tienen que confíar en el KGC como elemento generador, teniendo solo que comprobar que la comunicación ha sido correcta. Esto constituye una mejora respecto a la cantidad de

mensajes enviados y procesados por cada usuario pero con la condición de la confianza en el KGC, lo cual incorpora un elemento vulnerable a la comunicación. Es decir, si el KGC está comprometido, la comunicación entera también estará comprometida. Un acuerdo de clave distribuido no depende de un KGC, por lo tanto constituye un riesgo menor para la comunicación, a un coste mayor.

Actualmente, la mayoría de protocolos de acuerdo de clave usados en entornos reales son centralizados, dependiendo de un KGC, ya que prima la eficiencia sobre la seguridad. [Sun12]

4.1.4. Soluciones existentes centralizadas

Los protocolos definidos usan un centro de claves al que llamaremos KS (*Key Server*). Se distinguen dos tipos de claves, la primera, para establecer una comunicación segura entre el KS y cada participante, llamadas KEK (Key Encryption Keys). Y las segundas, para cifrar el tráfico entre los participantes, llamadas TEK (Traffic Encryption Keys). Cuando hablamos de acuerdo de clave, la clave que se está tratando de acordar es la TEK. Para más información sobre estos protocolos, así como otros del mismo tipo consultar [CS05]

1. Harney y Muckenhirn propusieron el protocolo de administración de claves de grupo, GKMP, que está basado en el uso de pares de claves. El KS comparte con cada miembro una clave KEK, estableciendo una comunicación segura. Entonces comparte una TEK, que será común para todos los miembros. Cuando es necesaria una actualización, ya sea porque un participante nuevo ha entrado, genera una nueva clave TEK, que envía al participante nuevo y al resto mediante la comuniación segura existente.

2. En el ejemplo anterior, se creaban canales seguros entre cada miembro y el KS, lo cual aumenta el coste en caso de que la clave quede comprometida, ya que es necesario reenviar una nueva clave a todos los participantes. Wong et al. idearon un protocolo en el que se distribuian las claves con estructura de árbol. Cada miembro mantiene su clave y las de sus subgrupos. Si alguna clave queda comprometida, solo hay que actualizar las claves de ese subgrupo.

3. Hasta ahora siempre se requería una comunicación privada entre todos los participantes y el KS mientras el acuerdo de clave se llevaba a cabo. Chico y Chen proponen un protocolo basado en el teorema chino del resto en el que el KS comparte un valor secreto con cada participante. Después genera y cifra la clave TEK con un valor aleatorio, que los participantes autorizados serán capaces de calcular

gracias a ese valor secreto. Esto elimina la necesidad de mantener un canal seguro entre los participantes durante el acuerdo de clave.

El protocolo de acuerdo de clave definido en este trabajo se basa en la misma idea, compartir un secreto y emitir en abierto el resto de la comunicación, como se detallará más adelante. Primero hay que entender con más profundidad la compartición de secretos.

4.2– Uso de secret sharing en el problema de acuerdo de clave

El uso del *secret sharing* en el problema de acuerdo de clave es sencillo. A continuación se detalla un protocolo para el intercambio de un secreto S entre un grupo de n participantes.

4.2.1. Procololo basado en la compartición de secretos

En este caso, los participantes deben cooperar con los otros emitiendo sus partes del secreto para poder reconstruirlo.

- Fase 1: Compartición de secretos
  1. El repartidor divide el secreto S en n partes: $S = s_1 + s_2 + ...s_n$
  2. El repartidor envía al participante P_i la parte del secreto s_i, $i = 1, 2, ..., n$, respectivamente en un canal seguro.
- Fase 2: Reconstrucción del secreto
  1. Todos los participantes P_i, $i = 1, 2, ..., n$ retransmiten sus partes al resto de participantes cuando quieren recuperar el secreto.
  2. El participante P_i recupera el secreto S calculando $S = s_1 + s_2 + ...s_n$

En este esquema, los participantes dependen los unos de los otros para recuperar el secreto, por lo tanto es un esquema válido para un intercambio de clave distribuido, pero no tanto para uno centralizado. Para arreglar esto, vamos a definir una modificación de este protocolo.

4.2.2. Protocolo basado en la compartición derivativa de secretos

- Fase 1: Compartición de secretos
  1. El repartidor divide el secreto S en 2 partes n veces: $S = s_1 + s'_1 = s_2 + s'_2 ... = s_n + s'_n$

2. El repartidor envía al participante P_i la parte del secreto s_i, $i = 1, 2, ..., n$, respectivamente en un canal seguro.

- Fase 2: Reconstrucción del secreto

1. El repartidor hace públicos las partes s'_i, para todo $i = 1, 2, ..., n$
2. El participante P_i recupera el secreto S calculando $S = s_i + s'_i$

Este esquema elimina la dependencia mutua en otros participantes. Al tener en cuenta el problema de acuerdo de clave y la compartición de secretos, se ve que la baja dependencia en otros participantes es lo esperado en los protocolos centralizados de acuerdo de clave. Por lo tanto, podemos usar este esquema para la creación de un protocolo de acuerdo de clave centralizado.

4.3– Casos prácticos

El crecimiento de internet en los últimos tiempos y el incremento del ancho de banda dedicado a las comunicaciones en red ha motivado el desarrollo de nuevas aplicaciones y usos de la red. Aunque las comuniaciones entre dos particiantes, también llamadas *unicast*, son predominantes, la demanda de aplicaciones *multicast*, es decir, entre varios participantes, está aumentando la demanda desde los ISP (Proveedores de Servicios de Internet).

Actualmente hay muchas aplicaciones que hacen uso del multicasting, sobre todo las aplicaciones orientadas a grupos de usuarios, como videoconferencias, juegos multijugador, TV online, servicios e-learning, entrega de actualizaciones de software, etc. La amplia variedad de aplicaciones que hacen uso de las comunicaciones en grupo, como el multicasting, hace necesaria una implementación de mecanismos de seguridad que garanticen la confidencialidad, integridad y autenticidad de la información.

Es vital, por tanto, que una solución a este problema sea lo más eficiente posible, evitando comunicaciones innecesarias y que a la vez se pueda garantizar su seguridad.

CAPÍTULO 5

Solución presentada

La solución al problema de un acuerdo de clave centralizada está inspirada en el articulo [Sun12]. Ya se ha definido el esquema a seguir en la resolucion del problema en el apartado 4.2.2. Se trata de la creación de un protocolo de acuerdo de clave centralizado basado en la compartición de secretos derivativa.

Esto permitirá, usando técnicas de compartición de secretos, eliminar la dependencia del resto de participantes.

Como se indicó anteriormente, existirá un KGC (Centro generador de claves) que se encargará tanto de la generación como de la distribución de la clave. Es decir, actuará a la vez como KGC (Centro generador de claves) y como KS (Servidor de claves).

5.1– El protocolo

- **Registro de usuarios:** Cada usuario debe registrarse en el KGC si desea participar en el acuerdo de clave. En el momento del registro, el KGC comparte un secreto s_i' con cada usuario U_i.

  La comunicación entre el participante y el KGC durante el registro debe ser una comunicación segura, pues si la información que se intercambia aquí quedara comprometida, un participante no autorizado podría hacerse pasar por este participante sí autorizado.

  Para el registro y la comunicación segura no se establece ningúna directriz, permitiendo que se use el sistema deseado.

Una vez todos los participantes se han registrado en el KGC y han obtenido su secreto s'_i, puede empezar la generación y distribución de la clave.

- **Generación de la clave de grupo y distribución:**

  1. El participante líder envía al KGC un mensaje con la lista de usuarios que van a participar en el acuerdo. Esta lista $\{u_1, u_2, ...u_n\}$ contendrá las identidades de los usuarios en la sesión.
  2. El KGC emite a todos los usuarios la lista de participantes que acaba de recibir, para que todos comprueben los usuarios entre los que se va a llevar a cabo el acuerdo.
  3. Cada miembro envía un valor aleatorio r_i al KGC. Este valor es público.
  4. El KGC genera un secreto S y calcula la clave de sesión $K = g^S$. Entonces, usa el protocolo de compartición de secretos derivativa para dividir S en dos partes n veces

  $$S = s_1 + s'_1 = s_2 + s'_2 = ... = s_n + s'_n \tag{5.1}$$

  Calcula M_i, para $i = 1, 2, .., n$ y $Auth$ de la siguiente manera:

  $$M_i = \{g^{s_i+r_i}, u_i, H(u_i, g^{s_i+r_i}, s'_i, R_i)\} \tag{5.2}$$

  $$Auth = H(K, g^{s_1+r_1}, ..., g^{s_n+r_n}, u_1, ..., u_n, r_1, ..., r_n) \tag{5.3}$$

  Una vez los ha calculado, distribuye $\{M_1, .., M_n, Auth\}$ entre todos los participantes
  5. Cuando el usuario U_i recibe $\{M_1, .., M_n, Auth\}$, calcula:

  $$h = H(u_i, g^{s_i+r_i}, s'_i, R_i) \tag{5.4}$$

  dónde $g^{s_i+r_i}$ y u_i están sacados de M_i, s'_i es la parte secreta almacenada por U_i y r_i es un valor público. Una vez calculado, el usuario comprueba si h es igual a la parte correspondiente que venía en M_i. En caso contrario, para. Si lo es, continúa calculando:

  $$K' = g^{s'_i} * g^{s_i+r_i}/g^{r_i} = g^{s_i+s'_i} = g^S \tag{5.5}$$

  $$Auth' = H(K', g^{s_1+r_1}, ..., g^{s_n+r_n}, u_1, ..., u_n, r_1, ..., r_n) \tag{5.6}$$

  Y comprueba si K' puede satisfacer la condición de que Auth sea idéntico a Auth'. En eso caso, K' es igual a la clave de sesión K generada por el KGC.

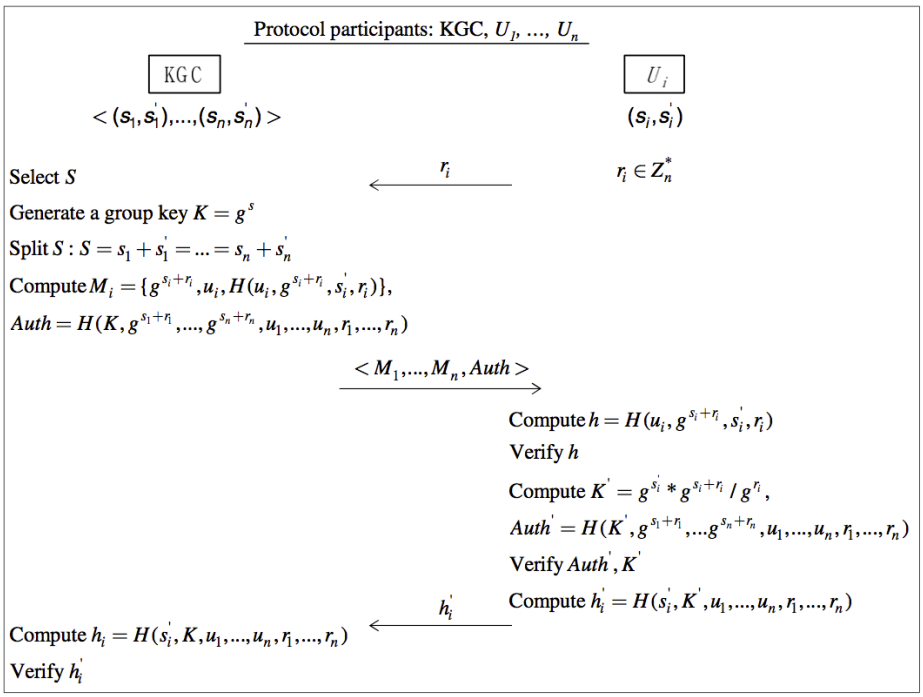

Figura 5.1: Esquema de ejecución del protocolo

6. Por último, cada usuario envía un valor h al KGC:

$$h = H(s_i', K', u_1, ..., u_n, r_1, ..., r_n) \tag{5.7}$$

   El cual también lo calcula y comprueba que sean idénticos. Esta última comprobación se hace para estar seguros de que cada usuario del acuerdo ha obtenido la clave correcta.

En la figura 5.1 se muestra un esquema de la ejecución del protocolo una vez se ha producido la fase de registro.

5.2– Análisis del protocolo

En este esquema un usuario autorizado solo necesita almacenar un mismo secreto s_i que se le proporciona durante la fase de registro para todas las sesiones en las que se vea involucrado. Se asigna una vez y se sigue usando en los sucesivos acuerdos.

Cuando es necesario incorporar un usuario o eliminarlo, lo único que tiene que hacer el KGC es generar un s_i nuevo para ese usuario o no usar su s_i en la comunicación, respectivamente. Esto reduce la cantidad

de mensajes a transmitir y almacenar en cada ronda, en comparación con los ya vistos en el apartado 4.1.4, de ejemplos centralizados de acuerdo de clave.

5.2.1. Análisis de requisitos de seguridad en la clave

Los requisitos de seguridad requeridos para el uso de la clave en el protocolo son los siguientes:

- **Frescura de la clave:** Es necesario que la clave de sesión distribuida por el KGC nunca se haya usado antes en la comunicación. Esto elimina la amenaza del uso de una clave que haya sido comprometida en una ronda anterior. Esto está garantizado ya que el KGC genera aleatoriamente el secreto S en cada sesión y comprueba que dicho secreto no se haya generado en una sesión anterior.

- **Confidencialidad de la clave:** Garantizar que la clave solo puede ser recuperada por los usuarios autorizados. Debido a la controversia de este apartado, se tratará con más profundidad en el siguiente.

- **Autenticidad de la clave:** Se requiere una garantía a los usuarios autorizados de que la clave que han obtenido es la generada por el KGC. Para esto, se ha usado una función resumen. En el paso 4, el KGC genera $Auth = H(K, g^{s_1+r_1}, ..., g^{s_n+r_n}, u_1, ..., u_n, r_1, ..., r_n)$ y lo retransmite. Dado que sólo los participantes autorizados y el KGC conocen la clave de grupo K, sólo estos puede generar el mensaje Auth, siendo imposible para un atacante externo.

5.2.2. Análisis de ataques externos e internos

Aquí se analizarán los diferentes ataques según su origen.

Atacantes externos

Según el artículo [Sun12], un atacante externo intentaría dos formas de comprometer la clave. La primera es intentar hacer uso de toda la información que se retransmite por el canal de la comunicación entre el KGC y los usuarios autorizados, e intentar hacer un ataque de *replay*.

Un ataque de replay consiste en interceptar mensajes de la comunicación y volverlos a mandar para obtener la misma respuesta. Así se conseguiría información útil y potencialmente comprometida. La ventaja de este protocolo es la inutilización de la información desfasada.

Una información desfasada es inútil, ya que en el paso 3 se introduce un elemento aleatorio que se genera en cada sesión, impidiendo la reutilización de la información entre sesiones.

La otra manera es hacerse pasar por un miembro del grupo y participar en la ejecución del protocolo. No habría ninguún impedimento en iniciar el acuerdo con un miembro no autorizado y obtener las respuestas apropiadas del KGC, pero al no poseer el secreto compartido por el KGC en la fase de registro es imposible recuperar la clave.

En el artículo [MKW13], se citan algunas razones porque la que no se puede garantizar la confidencialidad de la clave en caso de un atacante externo, que se detallarán a continuación.

Un atacante podría interceptar el mensaje $\{M_1, .., M_n, Auth\}$ del canal de transmisión entre el KGC y los participantes. Ya que conoce $M_i = \{g^{s_i+r_i}, u_i, H(u_i, g^{s_i+r_i}, s'_i, R_i)\}$, puede obtener $H(u_i, g^{s_i+r_i}, s'_i, R_i)$. Por lo tanto conoce $g^{s_i+r_i}$ y u_i. También conoce r_i, ya que este es un valor público.

El atacante podría averiguar s'_i con un ataque de fuerza bruta sobre la función resumen $H(u_i, g^{s_i+r_i}, s'_i, R_i)$, pero la dureza de esta función reside en la dureza de s'_i, por lo tanto sólo es una amenaza en caso de que la debilidad del s'_i lo sea.

Atacantes internos

Para un atacante interno, el protocolo debe ser capaz de asegurar que todos los participantes reciben la clave. Es decir, que ningún participante puede impedir que otro reciba la clave y dar por entendido que sí la ha recibido.

Es este protocolo, un usuario P_i puede generar un valor aleatorio r'_j para hacerse pasar por otro usuario P_j y recibir los mensajes correspondientes del KGC. Pero en este caso no puede generar la respuesta $h = H(s'_j, K', u_1, ..., u_n, r_1, ..., r_n)$, ya que no posee s'_j, por lo tanto el KGC detectaría que uno de los participantes no ha sido capaz de recibir la clave y daría por erróneo el acuerdo, empezando de nuevo.

CAPÍTULO 6

Implementación

6.1– Definición de requisitos

Los requisitos relativos a la implementación de este TFG se pueden resumir en cinco puntos.

1. Creación de una estructura de clases para trabajar con el centro generador de claves KGC.
2. Generación de una clase participantes, capaz de llevar a cabo una comunicación con otras entidades
3. Desarrollo de una estructura que permita la comunicación entre los participantes y el KGC
4. Creación de una interfaz que permita el uso y la visualización del protocolo usado
5. Desarrollo de un programa final con soporte multiplataforma, debido al enfoque didáctico de este trabajo.

6.2– Selección del entorno de trabajo

Como lenguaje de implementación se ha elegido Python en su versión 2.7. Gracias a su amplio uso en el Grado de Ingeniería Informática de la ETSII, el tiempo total dedicado al desarrollo se verá reducido por los recursos y herramientas proporcionadas a lo largo de la carrera.

También es crucial la facilidad de trabajo con números grandes y el soporte multi-plataforma del lenguaje y sus bibliotecas.

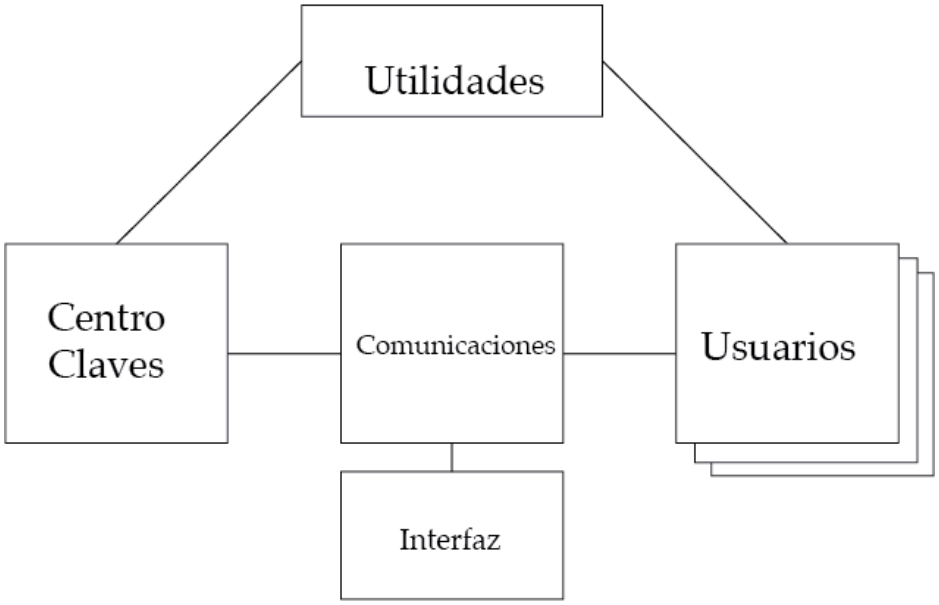

Figura 6.1: Relaciones entre las diferentes clases a implementar

6.3– Estructura de clases

Lo primero que debemos tener en cuenta a la hora de empezar a construir el programa son los diferentes elementos de los que va a estar compuesto. En la figura 6.1 puede verse un esquema de las relaciones entre clases. La implementación general va a tener cinco partes, que se definirán a continuación.

1. **Implementación del protocolo**

   El protocolo tiene una estructura general, que se implementará en dos clases: Centro de claves y Usuario. Estas dos clases tendrán como obetivo la reutilización de código, debido a que no están enfocadas al usuario final.

2. **Implementación de utilidades**

   Al estar trabajando con números grandes y operaciones criptográficas, es necesario que se implementen algunas rutinas para mejorar las operaciones elementales con las que se trabaja. También, para reutilizar código en muchos de los métodos, se definirá un apartado de utilidades dónde se unirán estas operaciones y métodos, permitiendo su acceso desde cualquier parte del código.

3. **Implementación de los métodos de comunicación**

Al tratarse de un protocolo que comunica a varios participantes, es necesaria la implementación de métodos que permitan el envío de mensajes entre ellos, pero que sea independiente de la implementación de los mismos.

Esto debe hacerse modularmente, de tal manera que permina la reutilización del código en proyectos posteriores. Es decir, si alguien quiere implementar este acuerdo de clave en un proyecto externo, sólo tendrá que reescribir los métodos relativos a la comunicación, siendo utilizable el resto del código.

4. **Generación de una interfaz orientada al usuario**

   Uno de los requisitos del proyecto es que tuviera un enfoque didáctico. Este protocolo es un protocolo casi invisible al usuario final, ya que el problema de acuerdo de clave es un paso intermedio en el proceso de asegurar una comunicación en grupo. Para un usuario estándar no debería importar cómo está sucediendo este acuerdo, siempre que se produzca. Nuestro objetivo, por otro lado, es mostrar cómo sucede esta comunicación y permitir a un estudiante examinar el intercambio de mensajes que ocurre entre los participantes.

   Por lo tanto es necesario el diseño de una interfaz acorde a dichas necesidades y su posterior implementación.

5. **Testeo** Por último, una clase para testear que las implementaciones que se están haciendo están libres de fallos y cumplen los requisitos establecidos.

6.4– Implementación del protocolo

Para la implementación del protocolo usamos como base la descripción hecha en el apartado 5.1 de este documento. Los elementos que aparecen mencionados en el protocolo se pueden dividir en dos tipos:

- **Centro de generación de claves, KGC**

  El KGC será el encargado de generar la clave y mantener un registro de los usuarios, por lo tanto se usará una clase que lo defina, la cual será llamada **KeyGenerationCenter**

- **Usuario**

  Cada partipante tendrá que almacenar sus propios valores temporales, así como garantizar una serie de requisitos de seguridad. Estos participantes se implementarán en otra clase, llamada **User**

La comunicación de los mensajes, como se ha indicado en el apartado anterior, no estará implementada aquí, ya que dependiendo del sistema

usado, funcionará mediante unos canales u otros.

Lo que sí hay que implementar es un soporte para la comunicación, en este caso que cada una de las clases posea la capacidad de enviar y recibir mensajes y modificar su estado interno en base a estos. Una estructura lógica para dicho esquema es una máquina de estados, que vaya cambiando de estado a medida que se van enviando mensajes. En la figura 6.2 se muestra un diagrama del funcionamiento de esta.

El cambio de estado se efectuará de la siguiente manera:

1. Entrada de mensaje nuevo: Procesamiento de los datos recibidos en ese mensaje
2. Comprobacion requisitos para el cambio de estado
3. Ejecución de las operaciones del nuevo estado
4. Envio de los mensajes correspondientes
5. Espera a la recepción de nuevos mensajes

Las ventajas de usar una máquina de estados como modelización del problema son claras; es fácil saber cuando un mensaje ha sido enviado en un momento erróneo y saber si existe un fallo con el protocolo.

6.4.1. Formato de mensaje

En la comunicación, se ha definido el formato de mensaje como una lista de elementos con el siguiente orden:

$$Mensaje = \{Tipo, Origen, Destino, Datos\} \tag{6.1}$$

Los campos usados en este mensaje son los siguientes:

- **Tipo**: Identificación del tipo de mensaje.

  Define el tipo de mensaje que se está enviando, lo cual permite saber el estado en el que está el participante que lo haya enviado.

- **Origen**: Identificador de emisor.

- **Destino**: Identificador de destino. Se pueden identificar tres tipos de destino:

  1. KGC: El mensaje irá dirigido al servidor de claves
  2. Usuario: El menaje irá dirigido a un usuario en particular

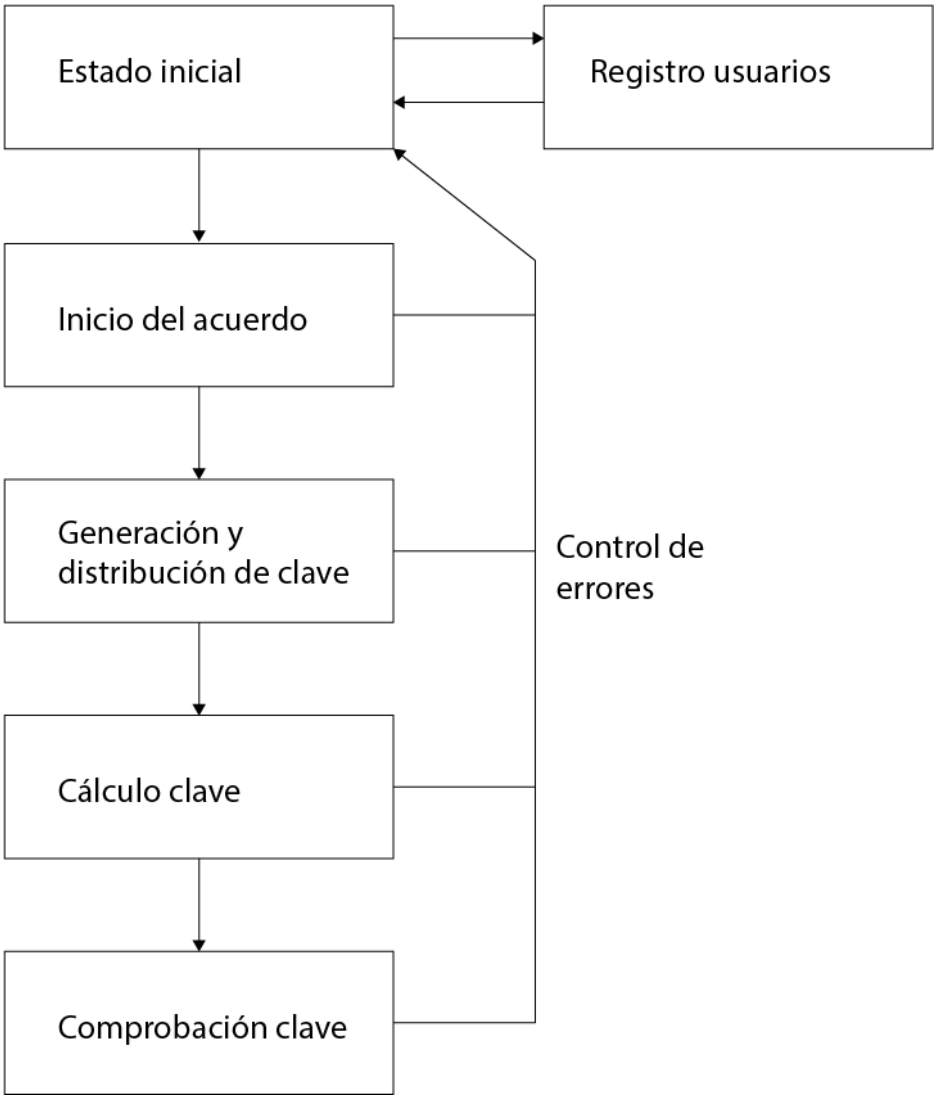

Figura 6.2: Diagrama de los estados por los que pasa el protocolo

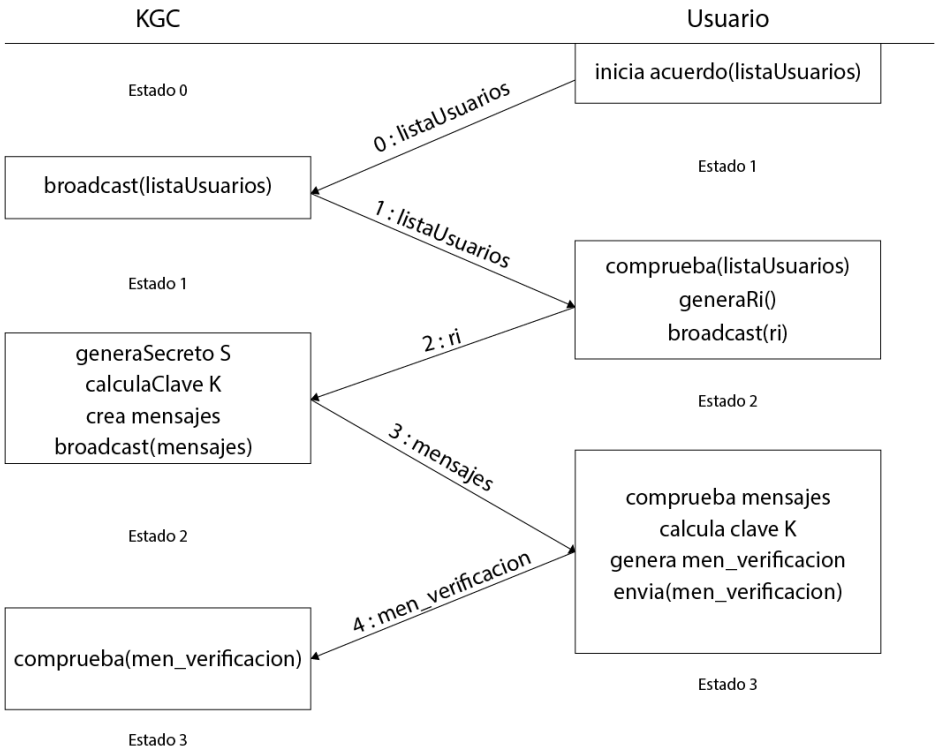

Figura 6.3: Diagrama de comunicación entre estados

3. Broadcast: Se enviarán copias de este mensaje a todos los participantes, incluyendo el servidor de claves, excepto al emisor del mismo.

   El destino broadcast es un destino especial que depende de la implementación de la capa de comunicaciones. Se podría también habilitar el soporte para que el destino pudieran ser múltiples usuarios, pero es más sencillo implementar esta orden.

6.4.2. Estructura del código de transmisión y recepción de mensajes

La tranmisión de mensajes que se va a exponer a continuación pertenece al protocolo de acuerdo de clave una vez los usuarios han sido registrados. La comunicación de la fase de registro será definida más adelante.

Las clases `KeyGenerationCenter` y `User` poseen dos métodos que se encargan de la interacción entre los mensajes y el funcionamiento de la maquina de estados interna, estos son:

1. `send_message()` - El cual se encarga de la creación de la estructura de datos del mensaje.

2. `receive_message()` - Se encarga de la recepción del mensaje y procesamiento de los datos que vienen en este. Al ser el mecanismo de entrada de la máquina de estados, se encargará del cambio de estado y del control de errores en la comunicación.

6.5– La clase `KeyGenerationCenter`

La clase que implementa el servidor de claves se puede dividir en tres partes:

- **Datos:** Almacenamiento de los datos a usar durante el protocolo
- **Comunicacion:** Métodos relativos al envío y la recepción de mensajes
- **Operaciones** Métodos a usar durante los cambios de estado

6.5.1. Datos

La clase `KeyGenerationCenter` debe almacenar las identidades de los participantes de la sesión a medida que estos se van registrando. También debe almacenar las subclaves generadas para estos y los datos necesarios del protocolo. Por último, se debe almacenar el estado interno en el que se encuentra en la máquina de estados, para saber los datos que es necesario

comprobar. Todo esto se implementa en el constructor de la clase, como se puede ver en código.

```
1         self.name = 'Kgc'  # Identificador
2         self.secret = 0    # Secreto, aun no generado
3         self.state = 0     # Estado
4
5         self.users = []       # Conjunto de usuarios registrados
6         self.randoms = dict() # Valores aleatorios de los users, ri
7         self.subkeys = dict() # Secreto mandado a cada usuario, si
8
9         # Parametros del intercambio
10        self.message = [] # Mensaje correspondiente al paso 3
11        self.auth = []  # Mensaje de autenticacion
12        self.k = 0      # Clave
```

Código 6.1: Atributos de la clase KGC

6.5.2. Comunicación

La clase está implementada de tal manera que para cambiar de estado, primero hay que enviar un mensaje mediante `receive_message()`, que recibirá el mensaje y procesará los datos, para luego llamar a `send_message()`, que comprobará si los objetivos del estado se han cumplido, cambiará al siguiente y enviará los mensajes necesarios.

Para `receive_message()`, primero se descompone el mensaje en partes, procesando su contenido. Después, dependiendo la identificación del mensaje y del estado del KGC, se procesarán los datos. Hasta que no se ejecute `send_message()` no se comprobará si esos datos son los correctos.

```
1     def receive_message(self, message):
2         msgId, msgSrc, msgDst, msgData = self.descomponeCabeceras(
              message)
3         if msgId == 0:
4             self.numUsers = len(msgData)
5             for n in msgData:
6                 self.addUser(n)
7         elif msgId == 2:
8             self.randoms[msgSrc] = msgData
9         elif msgId == 4:
10            self.checkHi(msgData,msgSrc)
11        else:
12            self.error(2) # Mensaje incorrecto
```

Código 6.2: Recepcion de mensajes del KGC

En la figura 6.5.2 se muestra la implementación de `send_message()` que genera los mensajes para ser enviados en el KGC. Se puede comprobar que de los cinco estados por los que pasa el protocolo, este solo envía mensajes en los tres primeros. Cuando la funcion `send_message()` sea llamada en un estado diferente a estos, no se enviará ningún dato.

Como se puede comprobar, es aquí donde se produce la creación de la estructura de datos del mensaje.

```
1  def send_message(self):
2      # Aqui se comprueba si todo es correcto y es el estado
3      if self.compruebaDatosDeEstado(self.state) == -1 :
4          return False
5      s = self.state
6      if s == 1:
7          msgdata = self.users
8          data = [1,'kgc','broadcast',msgdata]
9      elif s == 2:
10         msgdata = self.generaMensaje()
11         data = [3,'kgc','broadcast',msgdata]
12     elif s == 3:
13         data = []
14     return data
```

Código 6.3: Envio de mensajes del KGC

6.5.3. Operaciones

Las operaciones a relizar durante la ejecución del protocolo tienen lugar aquí. Estas usan código de la clase `Utilidades`, que se detallará mas adelante. Los diferentes métodos que se usan son los siguientes:

- `generateSubkey()`: Genera una parte del secreto s_i que se compartira durante el proceso de registro del participante.
- `generarMensajeUsuario(indice)`: Genera el valor M_i

$$M_i = \{g^{s_i+r_i}, u_i, H(u_i, g^{s_i+r_i}, s'_i, R_i)\} \tag{6.2}$$

- `generarMensaje()`: Devuelve todos los mensajes M_i, para $i = 1, 2..., n$
- `generateAuth()`: Genera el mensaje Auth:

$$Auth = H(K, g^{s_1+r_1}, ..., g^{s_n+r_n}, u_1, ..., u_n, r_1, ..., r_n) \tag{6.3}$$

- `generateHi(user)`: Calcula h_i:

$$h = H(s'_i, K', u_1, ..., u_n, r_1, ..., r_n) \tag{6.4}$$

- `checkHi(user)`: Comprueba que el valor calculado h_i es igual al valor recibido h'_i.

```
# Operaciones a realizar por el KGC
def generateSubKey()           # Genera la subclave un usuario.
def generarMensajeUsuario(indice): # Genera el mensaje Mi
def generaMensaje()            # Almacena todos los Mi para enviar
def generateAuth():            # Genera el mensaje Auth
def generateHi(user):          # Genera el Hi del ultimo paso
def checkHi(user):             # Comprueba si el Hi recibido
    concuerda
```

Código 6.4: Operaciones realizadas por el KGC

6.6– La clase `User`

La clase que implementa al usuario sigue el mismo esquema que la implementada para representar al KGC. Se divide en tres bloques:

- **Datos:** Almacenamiento de los datos a usar durante el protocolo
- **Comunicacion:** Métodos relativos al envío y la recepción de mensajes
- **Operaciones** Métodos a usar durante los cambios de estado

6.6.1. Datos

La clase `User` debe almacenar la subclave enviada por el KGC durante el proceso de registro. Debe almacenar también los datos intermedios del protocolo. Por último, se debe almacenar el estado interno en el que se encuentra en la máquina de estados, para saber los datos que es necesario comprobar. Todo esto se implementa de nuevo en el constructor de la clase.

```
    self.name  = name     # Id
    self.random = -1      # Valor aleatorio generado
    self.key   = -1       # Clave final
    self.subkey = -1      # si
    self.leader = leader  # Indica si es el lider
    self.msg   = []       # Almacena Mi y Auth

    # Valores del sistema
    self.publicUsers  = []
    self.publicRandoms = dict()
    self.publicValues = []

    # Estado del usuario.
```

```
14      self.state = 0
```

Código 6.5: Atributos de la clase User

6.6.2. Comunicación

La clase está implementada de tal manera que para cambiar de estado, primero hay que enviar un mensaje mediante `receive_message()`, que recibirá el mensaje y procesará los datos, para luego llamar a `send_message()`, que comprobará si los objetivos del estado se han cumplido, cambiará al siguiente y enviará los mensajes necesarios.

Para `receive_message()`, primero se descompone el mensaje en partes, procesando su contenido. Después, dependiendo la identificación del mensaje y del estado, se procesarán los datos. Hasta que no se ejecute `send_message()` no se comprobará si esos datos son los correctos.

```
1  def receive_message(self, message):
2      msgId, msgSrc, msgDst, msgData = self.descomponeCabeceras(
           message)
3      if msgId == 1:
4          if self.publicUsers == []:
5              self.publicUsers = msgData
6      elif msgId == 2:
7          self.recibeRandom(msgSrc,msgData)
8      elif msgId == 3:
9          self.recoverMsg(msgData)
10     else:
11         self.error(2)
```

Código 6.6: Recepcion de mensajes del usuario

En `send_message()` se genera los mensajes para ser enviados en al KGC y al resto de participantes. Eso ejecuta las operaciones necesarias para cada paso y envía los datos resultantes.

```
1      def send_message(self):
2          # Comprueba si los datos son correctos antes de enviarlos
3          if self.compruebaDatosDeUser(self.state) == -1:
4              return False
5          s = self.state
6          data = []
7          if s == 1:
8              if self.isLeader():
9                  data = [0,self.name,'kgc',self.publicUsers]
10             else:
11                 data = []
12         if s == 2:
```

```
13              msgdata = self.generateRandom()
14              self.recibeRandom(self.name,msgdata)
15              data = [2,self.name,'broadcast',msgdata]
16          if s == 3:
17              self.recoverKey()
18              msgdata = self.computeHi()
19              data = [4,self.name,'kgc',msgdata]
20          return data
```

Código 6.7: Envio de mensajes del usuario

6.6.3. Operaciones

Las operaciones a relizar durante la ejecución del protocolo tienen lugar aquí. Estas usan código de la clase `Utilidades`, que se detallará mas adelante. Los diferentes métodos que se usan son los siguientes:

- `generateRandom()`: Genera un valor aleatorio, para cada sesión
- `recibeRandom(user, random)`: Almacena el valor r_i de cada participante
- `recoverMsg(m)`: Divide el mensaje $m = \{M_1, M_2, ..., M_1, Auth\}$ en sus componentes y los almacena.
- `recoverKey()`: Calcula la clave final mediante:

$$K' = g^{s'_i} * g^{s_i+r_i}/g^{r_i} \tag{6.5}$$

- `generateAuth()`: Genera el mensaje Auth'

$$Auth' = H(K', g^{s_1+r_1}, ..., g^{s_n+r_n}, u_1, ..., u_n, r_1, ..., r_n) \tag{6.6}$$

- `compruebaAuth()`: Comprueba que Auth = Auth'
- `genH()`: Calcula h_i:

$$h'_i = H(u_i, g^{s_i+r_i}, s'_i, R_i) \tag{6.7}$$

- `compruebaH()`: Comprueba que el valor calculado h' es igual al valor recibido h que estaba dentro de M_i.
- `computeHi(user)`: Calcula h_i:

$$h = H(s'_i, K', u_1, ..., u_n, r_1, ..., r_n) \tag{6.8}$$

```
    # Metodos usados para generar ri y almacenar los valores
    # ri del resto de participantes
    def generateRandom()
    def recibeRandom(user, random)

    # Recuperacion del mensaje Mi, Auth
    def recoverMsg(m)
    # Calculo de la clave final
    def recoverKey()

    # Metodos de generacion/comprobacion de mensajes
    def generateAuth():
    def compruebaAuth()
    def genH()
    def compruebaH()
    def computeHi()
```

Código 6.8: Operaciones realizadas por el User

6.7– La clase `Comunicador`

Abreviada como clase `Comm`, es la que define un flujo de control en el protocolo y decide quien va a actuar en cada momento. También se encarga de transmitir los mensajes entre los diferentes participantes.

6.7.1. Retransmisión de mensajes

El mecanismo para la restransmisión de mensajes es sencillo

1. Comprueba que el mensaje esté bien formado
2. Almacena el mensaje en el registro de mensajes
3. Ejecuta los métodos `receive_message` de aquellos participantes que figuren en la etiqueta `destino`

   En caso de que esté marcado como `broadcast` se envía a todos los participantes

```
for rec in receivers:
    if msgSrc != rec.name and (msgDst == 'broadcast' or msgDst ==
        rec.name):
        rec.receive_message(mensaje)
```

Código 6.9: Enrutamiento de mensajes

6.7.2. Orden de actuación

En cada estado de la maquina de estados definida en el protocolo, cada participante enviará el mensaje que tenga asignado, empezando por el líder que inicia la comunicación.

La clase `Comm` va recorriendo todos los participantes y el KGC para indicarles que es su turno para enviar el mensaje correspondiente. Es necesario recordar que cuando un participante envia su mensaje, antes comprueba si todas las operaciones de ese estado han sido realizadas y los datos son correctos.

```
1     def bucle(self):
2         states = 3
3         for n in range(states):
4             for participant in self.participants:
5                 self.routeMsg(participant.send_message())
6             self.routeMsg(self.kgc.send_message())
```

Código 6.10: Enrutamiento de mensajes

6.8– Utilidades

La clase `Utilidades` es usada en todo el sistema para proveer al código de operaciones elementales que no necesiten estar repetidas a lo largo de las diferentes clases. Estas utilidades se pueden resumir en tres categorías:

6.8.1. Utilidades del sistema

Estas *utilidades del sistema* proveen del código necesario para realizar operaciones elementales con conjuntos, cadenas de texto y codificación de archivos.

6.8.2. Algoritmos resumen

Durante el acuerdo de clave en el protocolo elegido se usa en varias ocasiones una función resumen, definida como $H()$. A la hora de implementar este protocolo, siguiendo las directrices de modularidad, se ha elegido la función resumen SHA1, que satisface las condiciones de eficiencia y eficacia necesarias para este proyecto. La implementación elegida es la de la biblioteca estándar de Python (`hashlib.sha1`).

Como muestra la implementación en el código siguiente, es trivial cambiar el algoritmo de resumen para seleccionar otro. Todo el proyecto usa la clase `Utilidades`, por lo que cambiar el algoritmo de resumen usado

en todo el proyecto es factible cambiando sólo una función en esta clase.

```
1 # Funcion resumen usada durante el protocolo
2 def hs(listaElementos):
3     res = sha1() # Aqui se selecciona la funcion resumen deseada
4     for element in listaElementos:
5         res.update(encoder(element))
6     return res.hexdigest()
```

Código 6.11: Implementación de la función resumen

6.8.3. Operaciones matemáticas

Durante la implementación de las clases`KeyGenerationCenter` y `User` se ha usado una serie de operaciones matemáticas que han sido implementadas aquí para evitar la repetición de código. Las tres funciones implementadas han sido:

1. Algoritmo extendido de euclides: Necesario para calcular la inversa modular.

2. Inversa modular. Se usa para la recuperación de la clave por parte de los participantes

3. Cálculo de clave: Realiza la operación $g^x \ mod \ p$

6.8.4. Cálculo de clave

Como se indicó en el capítulo de conceptos básicos, la clave se calcula con la operación $g^x \ mod \ p$. Es importante encontrar unos valores para g y p que nos permitan una generación correcta de la clave.

La *Internet Engineering Task Force* (IETF) provee de una serie de artículos relacionados con la seguridad en la web y la criptografía. Uno de dichos artículos [KK03] habla sobre la generación de claves y los generadores necesarios para esta. Aporta diferentes valores g y p para generación de claves seguras.

En este caso se han elegido los valores correspondientes al grupo de 1536 bits. Para usar otros valores, sólo habría que cambiar las variables correspondientes al generador y el módulo en la implementación de *Utilidades*.

6.9– Testeo

Debido a que los requisitos del problema del acuerdo de clave son sencillos, la fase de testeo también es sencilla, pues el estado final del acuerdo de clave es aquel en el que todos los participantes se han puesto de acuerdo en una misma clave.

El código para el testeo de este apartado se encontrará en la clase `Comunicador`, pues es la que permite acceder a todos los objetos `User` y consultar la clave acordada.

```
1  # Testea que se produzca el intercambio de clave correctamente
       entre todos
2  def test(numUsers):
3      kgc = KeyGenerationCenter()
4      comm = Comm(kgc,numUsers)
5      # Efectua el intercambio de clave
6      comm.bucle()
7      # Comprueba que se haya efectuado convenientemente.
8      key = comm.kgc.k
9      for n in comm.participants:
10         assert n.key == key
11     return True
```

Código 6.12: Testeo del intercambio de clave

6.10– Interfaz gráfica

Para el desarrollo de la interfaz gráfica de escritorio hay unos requisitos que es necesario cumplir, que se pueden resumir en los siguientes puntos:

- **Soporte multiplataforma:** Dado que la aplicación pretende tener un enfoque didáctico, es necesario que esté disponible en la mayoría de sistemas con entornos de escritorio. Linux, Windows y Mac OS son los principales sistemas operativos de escritorios usados.
- **Integración con el código existente:** La aplicación debe estar integrada con el código ya escrito, por lo tanto es conveniente que esté en el mismo lenguaje que el código o posea una pasarela de enlace para trabajar con los datos que este mande.
- **Modularidad:** De la misma manera que el código, la aplicación debe ser modular, permitiendo a un desarrollador modificar o ampliar la interfaz sin tener que modificar el núcleo de la aplicación.

Vistos los requisitos, se ha decidido trabajar con la librería `wxPython` [Dun12], que ofrece gráficos multiplataforma en el mismo lenguaje en el que se ha desarrollado la aplicación.

6.10.1. Estructura de clases

Tal y como viene indicado en la documentación de wxPython [Dun12], la aplicación contendrá una clase principal que será la encargada de manejar toda la lógica de la aplicación, así como comunicarse con las clases antes implementadas relacionadas con el acuerdo de clave. Dicha clase se llama `MainWindow`, y es una subclase de `wx.Frame`.

El código siguiente contiene una estructura de dicha clase, que contendrá los métodos relativos a las acciones a ejecutar, los objetos de la clase comunicador y la división de ventanas.

```
1  class MainWindow(wx.Frame):
2
3      def __init__(self,parent,id,title):
4          wx.Frame.__init__(self,parent,-1,title)
5          # [...] Aqui se llama al resto de clases
6          self.p1 = ConfigurationWindow(self.splitter, -1, '')
7          self.p2 = MessageList(self.splitter, -1)
8          self.splitter.SplitVertically(self.p1, self.p2,330)
9
10     def Reset(self): # Elimina los valores en pantalla
11     def OnButton(self,e): # Presiona el boton iniciar
12     def OnSelectList(self, e): # Abre la ventana de inspeccion
13     # Cambia los datos de la ventana de inspeccion
14     def ChangeInspectorMsg(self, idn, src, dst, data):
```

Código 6.13: Atributos de la clase User

La ventana de la aplicación está dividida en dos mitades, más las ventanas emergentes. Cada uno de estos elementos posee una clase para su definición. En la figura 8.1 se ve la división de la ventana principal de la aplicación. La parte izquierda corresponde a la clase `ConfigurationWindow` y la derecha a `MessageList`. La estructura de estas clases es casi idéntica a la clase principal `MainWindow`.

Esto está diseñado así ya que uno de los requisitos es que la aplicación sea modular y ampliable. En caso de querer desarrollar una ventana nueva, solo hace falta crear un archivo nuevo con la clase correspondiente a la ventana, definir los elementos, atributos y comportamiento y llamar a esta clase desde la ventana principal `MainWindow`.

En este caso, para la ventana principal, se ha usado un Splitter, encajando en él las clases correspondientes a las dos subventanas, como se muestra en el código anterior.

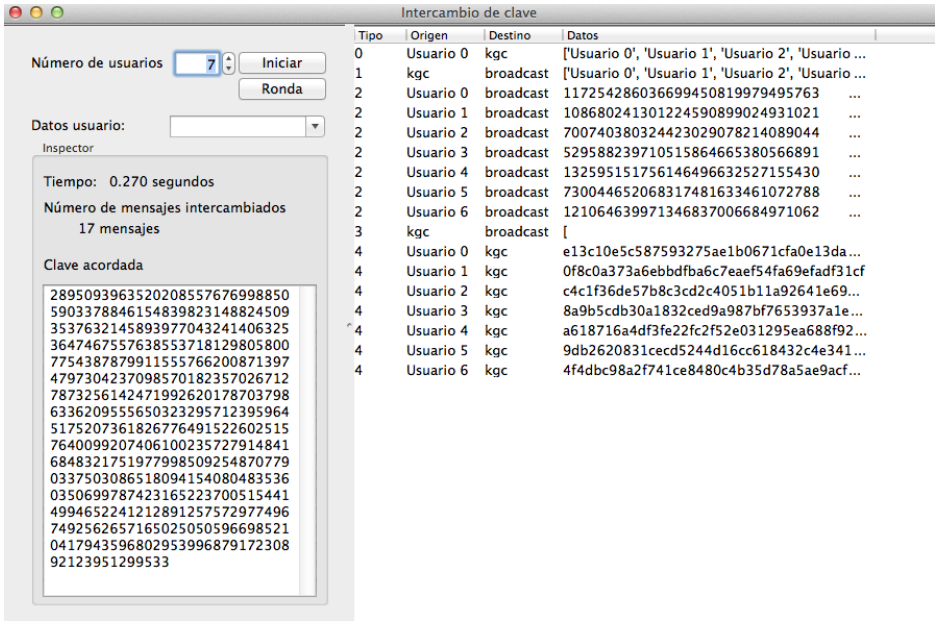

Figura 6.4: Ventana principal de la aplicación

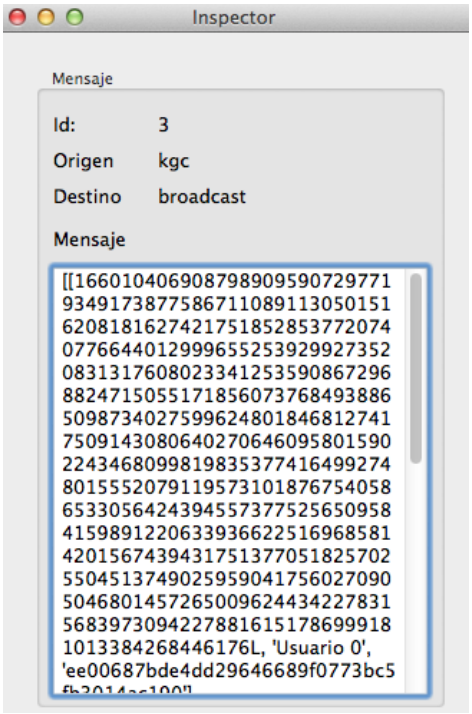

Figura 6.5: Inspector de mensajes

6.10.2. Ventanas flotantes

Se ha implementado también un inspector de datos, que se muestra en la figura 8.4, tanto para los mensajes como para los usuarios. En caso de hacer doble click en un mensaje, se mostrará una ventana con el origen, destino, tipo y datos del mensaje. Esto permite examinar en su totalidad los datos de un mensaje, cosa que la ventana principal no permite. En caso de inspeccionar un usuario, mostrará los datos relevantes a dicho usuario.

6.11– Creación de ejecutables

Para la creación de ejecutables multiplataforma, se ha decidido usar la aplicación `PyInstaller`. Dicha aplicación permite empaquetar en un archivo binario todas las bibliotecas necesarias para la ejecución del código en un sistema.

Esto significa que al crear un ejecutable para un sistema operativo dicho ejecutable contendrá todo lo necesario para su ejecución, incluyendo las bibliotecas gráficas (`wxPython` en este caso), la versión de `Python` correspondiente y los diferentes archivos de la aplicación.

Para su uso hay que configurar todo el sistema para que sea capaz de ejecutar la aplicación. Es decir, hay que instalar `PyInstaller, Python, wxPython` y los archivos relativos a la aplicación. Una vez esté todo configurado, hay que ejecutar el comando siguiente:

```
pyinstaller -Fw archivo.py
```

siendo `archivo.py` el archivo principal de la aplicación. Esto generará, como se ha indicado anteriormente, un ejecutable para el sistema en el que se ha creado.

Uno de los problemas de la creación de ejecutable es la disponibilidad de ejecutables para diferentes sistemas operativos. `PyInstaller` garantiza que el ejecutable funcionará en la versión del sistema operativo que esté corriendo, pero muchas veces no funciona en versiones anteriores o posteriores. Por lo tanto, es necesario que se busquen ejecutables para diferentes sistemas operativos y diferentes versiones. Esto ha sido posible gracias a `VirtualBox`, donde se han podido virtualizar las diferentes versiones de sistemas operativos para la creación de ejecutables.

6.12– Ampliación de código

Una de las ventajas de la estructura del código es su modularidad. Están totalmente separadas las capas de participantes (usuario y servidor), comunicación e interfaz. En caso de que otro desarrollador desee usar este proyecto para integrarlo en algún otro lugar, solo debe escoger los elementos que desee y encajarlos en el otro proyecto.

Las clases `KeyGenerationCenter` y `User`, por ejemplo, están diseñadas para funcionar mediante la recepción y envío de mensajes. En su diseño no importa como esté implementado el intercambio de mensajes. En este caso, al ser una aplicación didáctica, no ha sido necesario implementar una comunicación eficiente de mensajes, ya que toda comunicación toma lugar dentro de la misma máquina, pero podría darse el caso (y al hablar de acuerdo de clave es lo lógico) que cada participante se encuentre en una máquina diferente y por lo tanto sea necesario implementar los protocolos de comunicación en red. Para esto sólo haría falta modificar la capa de comunicaciones (figura 6.1), permaneciendo el resto inmutables.

CAPÍTULO 7

Conclusiones

7.1– Objetivos planteados y alcanzados

7.1.1. Resumen de los objetivos planteados

Los objetivos planteados durante este TFG han sido cinco, como ya se había comentado en el apartado de planificación:

1. Elección de un tema
2. Investigación de un artículo
3. Desarrollo e implementación de la aplicación
4. Creación de la memoria
5. Exposición oral

En el momento de finalización de esta memoria sólo se han completado los cuatro primeros objetivos, teniendo pendiende la exposición oral del trabajo junto con la presentación.

7.1.2. Objetivos alcanzados

Elección de un tema

Los tres temas en los que me centré cuando empecé a investigar el artículo eran el acuerdo de intercambio de claves en grupo, un análisis de una función resumen y el estudio de ataques de colisión a un tipo especial de función resumen. Acabé decidiendome por el acuerdo de clave ya que ví el tema más interesante y con más posibilidades de desarrollo.

Investigación de un artículo

Una vez acordé el artículo sobre el que iba a trabajar, investigué sobre las diferentes opciones de acuerdo de clave existentes y busque artículos que mencionaran el que ya había elegido. Este apartado fué un poco decepcionante, ya que en algunos artículos mencionaban problemas de seguridad en este acuerdo de clave, como se comenta en el análisis del protocolo. Aún así, sigue siendo un proyecto interesante por lo novedoso del protocolo y una gran meta a la hora de desarrollarlo.

Desarrollo e implementación de la aplicación

La implementación del acuerdo de clave ha sido uno de los aspectos más interesantes de este TFG, ya que me ha permitido llevar un concepto teórico a la práctica usando conocimientos que poseía de otras áreas, como las máquinas de estado. El reto de hacerlo totalmente modular para su posterior aprovechamiento ha marcado el camino del desarrollo, teniendolo siempre en cuenta.

Por otra parte, la aplicación gráfica ha supuesto un reto, ya que un protocolo de intercambio de clave no está hecho para el usuario final y es difícil implementar una interfaz para su comprensión. Creo que en la creación de la interfaz ya existe trabajo suficiente para un TFG, sin tener que analizar y desarrollar el protocolo, a la vez que implementarlo.

Por último, he comprendido la cantidad de trabajo necesaria para la exportación de una aplicación a diferentes sistemas operativos, ya sea Windows, Linux o Mac OS. En un principio pensaba usar `Kivy` como biblioteca gráfica, en vez de `wxPython`, ya que permite la apertura de la aplicación en dispositivos móviles, pero un fallo en la versión de escritorio me hizo abandonar el desarrollo, ya que no era compatible con algunas tarjetas gráficas, y el desarrollo de `Kivy` aún no es muy estable.

Creación de la memoria

En la creación de la memoria he analizado el protocolo de intercambio de clave y escrito sobre lo que he ido haciendo durante todo el proyecto. He conseguido un documento estructurado sobre el trabajo realizado durante el TFG.

El uso de LaTeX en la memoria requiere una mención especial, ya que nunca me había visto en la necesidad de usar LaTeX para un documento tan extenso. En ninguna asignatura de la carrera me han enseãdo a usar LaTeX y sería conveniente que en algún momento de los estudios de grado lo enseñaran, ya que es una gran herramienta para la documentación de

un proyecto, tanto por la estructura de su código como por la calidad del producto final.

7.2– Líneas de trabajo abiertas a partir de este TFG

Durante este TFG se han desarrollado conceptos que permiten que se trabaje mucho más en ellos:

- **Ataques al protocolo de intercambio de clave:** Como se desarrolló en el apartado de análisis del protocolo, existen algunas debilidades que podrían ser aprovechadas por un participante malicioso. Una posible línea de trabajo sería implementar estos ataques y analizar cómo de peligrosos serían en un entorno práctico.
- **Desarrollo de una interfaz completa:** Como se comentaba en el apartado de objetivos cumplidos, el desarrollo de una interfaz plantea una línea de trabajo bastante amplia, desde el análisis de objetivos de la misma bajo la disciplina *Interacción Persona-Ordenador* hasta la modificación del acuerdo de clave en tiempo real.
- **Uso del acuerdo de clave en otro proyecto**: Debido al carácter modular de este proyecto, es factible el uso del mismo para implementar un intercambio de clave en otro proyecto diferente.

7.3– Experimentación

Una de las ventajas de este protocolo es la cantidad de mensajes que es necesario enviar a la hora de acordar una clave. Por lo general, los protocolos de intercambio de clave centralizados necesitan realizar una ronda completa cada vez que se quiere actualizar la clave [CS05], sin embargo este protocolo no requiere enviar sub-claves nuevas a los participantes, ya que estos pueden continuar usando las que ya poseían.

El carácter centralizado de este acuerdo de clave, a diferencia de los distribuidos, hace que las operaciones y el tiempo utilizado aumenten linealmente con el número de participantes. En la figura 7.1 se puede comprobar como evoluciona el tiempo requerido para el intercambio de clave con el número de usuarios. La cantidad de mensajes intercambiados en el acuerdo de clave siempre será $3+2*n$, siendo n el número de participantes.

Por otra parte, se va a comparar este acuerdo de clave con otros acuerdos de clave centralizados, definidos en el artículo [CS05], en el cuadro 7.1.

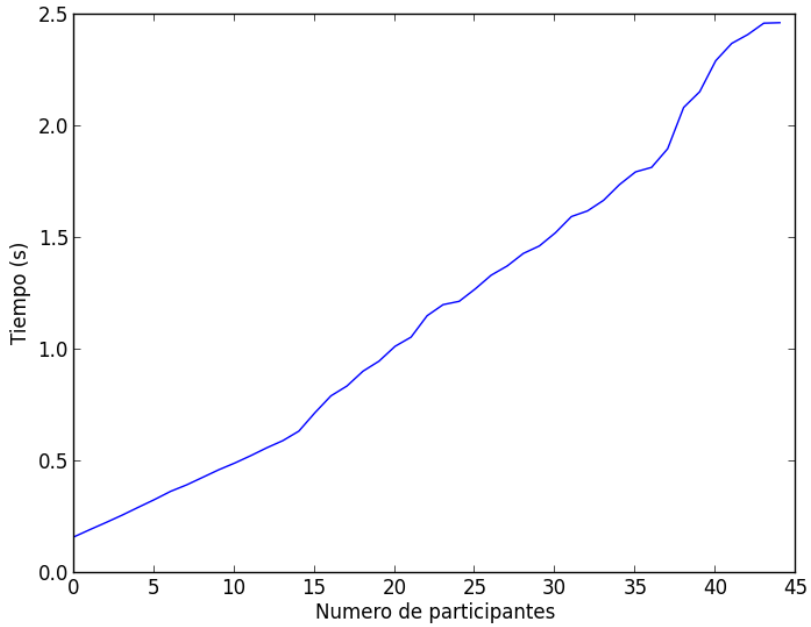

Figura 7.1: Evolución del tiempo con la cantidad de participantes

Protocolo	**CK**	**CU**	**Union**	**Abandono**
GKMP	$n+2$	3	$4n+2$	$2n$
LKH	$2n-1$	$log_2(n)+1$	$2n*log_2(n)-1$	$2log_2(n)$
OFT	$2n-1$	$log_2(n)+1$	$2n*log_2(n)+1$	$log_2(n)+1$
CFKM	$2n-1$	$n+1$	$n+1$	n
SUN	n	1	$2n+3$	$2n+3$

Cuadro 7.1: Comparativa con otros protocolos de intercambio de clave

Como se puede comprobar en el cuadro 7.1, el protocolo que se analiza en este TFG, el cual se ha denominado SUN se comporta muy bien en las dos primeras columnas, correspondientes a la cantidad de claves almacenadas por el KGC (CK) y a la cantidad de claves almacenadas por el usuario (CU).

Por otra parte, la cantidad de mensajes que es necesario intercambiar cuando se une un usuario nuevo (Union) y la cantidad de mensajes necesarios a intercambiar cuando un usario abandona. En este protocolo, al no quedar comprometidas las subclaves de cada usuario cuando uno de ellos abandona el acuerdo, y al no necesitar generar subclaves nuevas para todos los usuarios cuando uno nuevo entra, la cantidad de mensajes a enviar queda estable.

CAPÍTULO 8

Manual

8.1– Instalación

La aplicación pretende ser un ejecutable nativo, es decir, que no necesite instalación para su uso. Con ejecutar el archivo acuerdo.exe/app (dependiendo del sistema operativo en el que se encuentre) debería aparecer la pantalla de la aplicación.

En caso de querer modificar el código de la aplicación y ejecutarla, al final de este capítulo habrá una sección dedicada a ello.

8.2– Manual de uso

La aplicación consta de una ventana principal dividida en dos secciones, como se muestra en la figura 8.1. Se va a generar un acuerdo de clave y el usuario podrá visualizar los mensajes que se envían en dicho acuerdo.

8.2.1. Iniciando el acuerdo de clave

Para iniciar el acuerdo de clave, es necesario indicar la cantidad de usuarios que van a participar, para que la aplicación los genere y registre en el KGC.

En la esquina superior izquierda (figura 8.2) se puede indicar el número de participantes. Una vez estén indicados, hay que presionar el botón `Iniciar`.

Al presionar el botón `Iniciar`, si no ha aparecido ningún mensaje de error, debería llenarse la parte derecha de la pantalla con los mensajes

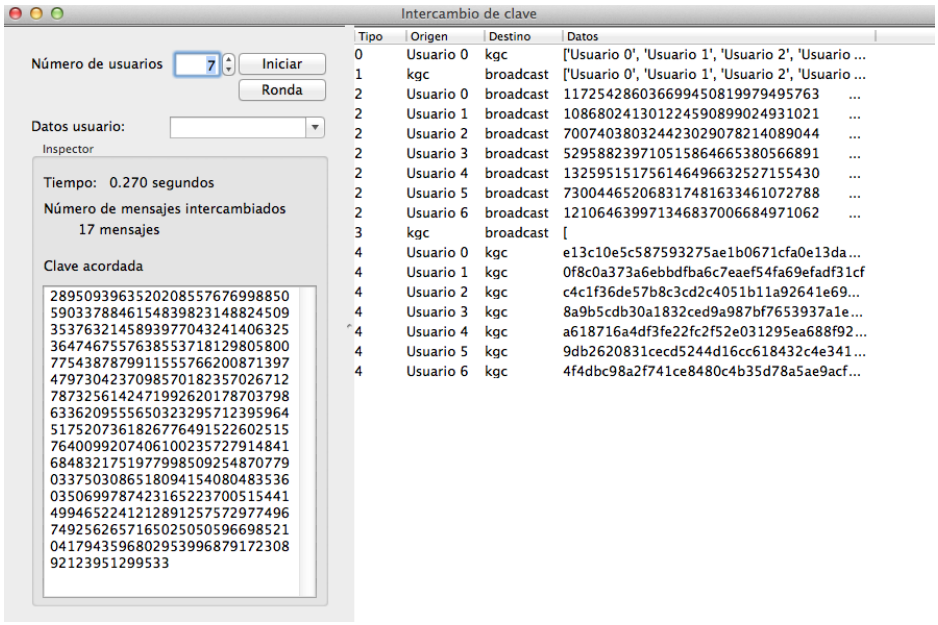

Figura 8.1: Ventana principal de la aplicación

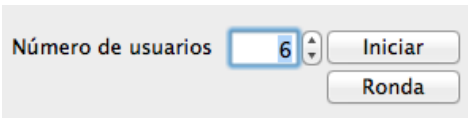

Figura 8.2: Botones de control de la aplicación

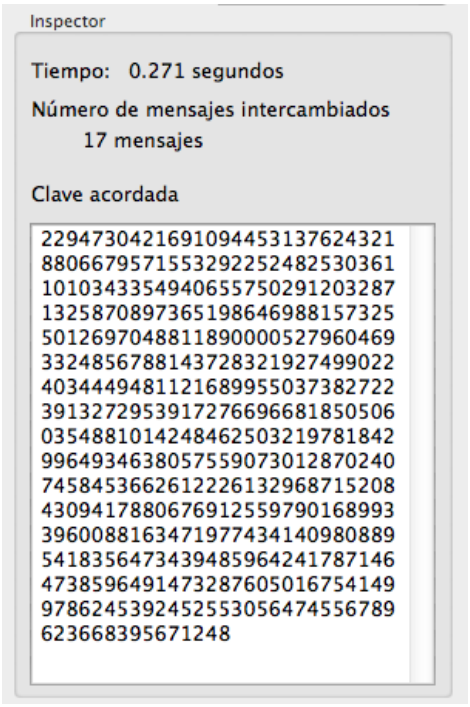

Figura 8.3: Datos del acuerdo de clave

que se han intercambiado en la aplicación, esto significa que el acuerdo de clave se ha completado. Para generar una clave nueva (o una ronda nueva) con los mismos usuarios y los mismos parámetros, se pulsa el botón `Ronda`.

Se pueden obsevar los datos en el inspector de la izquierda (figura 8.3). En este inspector viene indicado el tiempo que ha durado el acuerdo de clave, la cantidad de mensajes intercambiados y la clave final acordada.

8.2.2. Examinando los elementos

Para inspeccionar el contenido de cualquier mensaje se puede hacer doble-click en él, con lo que aparecerá una ventana (figura 8.4) que mostrará datos sobre dicho mensaje. En esta ventana vendrán los mismos datos que aparece en la lista, con la diferencia de un cuadro ampliado para los datos del mensaje. El contenido de algunos mensajes es largo, por lo que desde esta ventana se puede examinar con facilidad.

Por último, se pueden examinar los participantes en el acuerdo desde el menú `Datos usuario`. En ese desplegable se puede elegir un usuario para comprobar sus datos (figura 8.5). Dentro de esta ventana, se puede

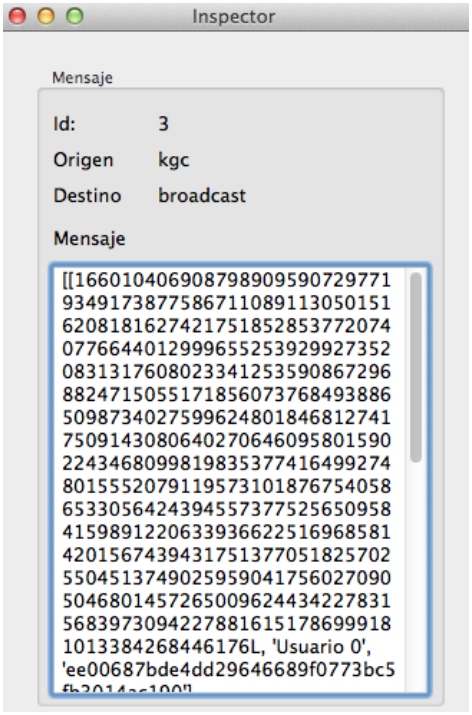

Figura 8.4: Ventana de inspección de mensajes

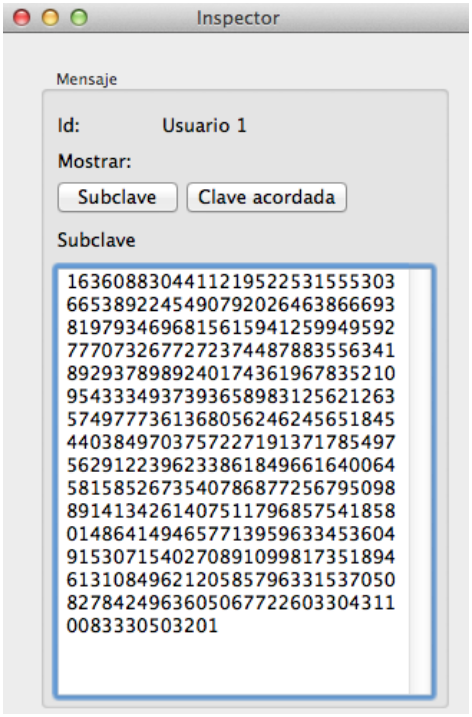

Figura 8.5: Ventana de inspección de usuarios

visualizar la clave acordada, que debería ser igual a la mostrada en la ventana principal, y la subclave perteneciente a dicho usuario, que debería permanecer sin alterar entre diferentes rondas.

8.3– Manual de desarrollo

Para modificar o reutilizar el código fuente de la aplicación hay que tener en cuenta que es un proyecto `Python`, y que cada una de las clases utilizadas se encuentra en un archivo diferente.

La versión de `Python` usada es la 2.7, pero debería funcionar en la 3, ya que no se usa ninguna de las nuevas funciones. Es necesario, una vez se tenga instalado `Python` 2.7, instalar `wxPython` para el desarrollo de la aplicación gráfica. Es recomendable hacerlo a través de la utilidad `pip`, mediante el comando

```
pip install wxpython
```

Por último, para ejecutar la aplicación, solo es necesario abrir el archivo interfaz.py con el intérprete de Python.

Bibliografía

[CS05] Challal, Yacine y Seba, Hamida, 'Group key management protocols: A novel taxonomy.' *International Journal of Information Technology, Volume 2, Number 1*, (2005).

[Dun12] Dunn, Robin, 'Documentation of the wxpython library.' *http://wxpython.org/Phoenix/docs/html/*, (2012).

[KK03] Kivinen, T. y Kojo, M., 'More modular exponential (modp) diffie-hellman groups for internet key exchange (ike) -.' *Force Internet Enigeering Task*, (2003).

[McC90] McCurley, KS., 'The disecrete logarithm problem.' *Proc. of Symposia in Applied Mathematics, vol 42*, (1990).

[MKW13] Mijin Kim, Namje Park y Won, Dongho, 'Cryptanalysis of an authenticated group key transfer protocol based on secret sharing.' *J.J. Park et al. (Eds.): GPC 2013, LNCS 7861, pp. 761–766*, (2013).

[Sha79] Shamir, Adi, 'How to share a secret.' *Communications of the ACM, Volume 22 Issue 11, Nov*, (1979).

[Sin00] Singh, Simon, 'Los códigos secretos.' -, (2000).

[Sun12] Sun, Yi, 'An authenticated group key transfer protocol based on secret sharing.' *2012 International Workshop in Information and Electronics Engineering (IWIEE)*, (2012).